sat-nms Software User Manual

API Reference Manual

Version 3.80.6-a -- 2025-12-03

SatService Gesellschaft für Kommunikationssysteme mbH
Hardstrasse 9
D-78256 Steisslingen

satnms-support@satservicegmbh.de

www.satnms.com
www.satservicegmbh.de
Tel +49 7738 99791-10

Table of Contents

1 API Reference

This document describes the sat-nms WebGUI backend server API which is used by the Web front end of the software and also may be used to control a sat-nms M&C in a programmatic way.

Quick access links to the main sections of the document:

- HTTPS Communication and OAuth2 Authentication
- API End Points
- WebSocket Communication
- Data Models
- Satellite Database

1.1 HTTPS Communication and OAuth2 Authentication

The sat-nms WebGUI backend server uses HTTPS for all communication. Unsecured HTTP calls are not accepted. In most cases a non-standard port number is used by the server, details about the port and also about the certificate provided by the server may vary for every installation of the software.

The backend server uses OAuth2 for authorization. It may be configured to accept JWTs issued by an external provider. In parallel the backend server provides a service for obtaining and renewing access tokens as well.

1.2 API End Points

This section describes the API end points provided by the sat-nms WebGUI backend server. The end point descriptions appear in an order which groups related functions. An alphabetic list of all end points is shown below.

- /api/v1/activebackend
- /api/v1/acutargets
- /api/v1/antennas
- /api/v1/applypreset
- /api/v1/backendinfo
- /api/v1/channels
- /api/v1/collectpreset
- /api/v1/dbstate
- /api/v1/dbversion
- /api/v1/debug
- /api/v1/devicesetup
- /api/v1/dframes
- /api/v1/documents
- /api/v1/drivers
- /api/v1/dscreens

- /api/v1/eventack
- /api/v1/eventreport
- /api/v1/filerecordings
- /api/v1/filerecordersettings
- /api/v1/globalSettings
- /api/v1/healthcheck
- /api/v1/inventory
- /api/v1/logout
- /api/v1/macros
- /api/v1/makewebsocket
- /api/v1/mnclist
- /api/v1/peek
- /api/v1/playmacro
- /api/v1/poke
- /api/v1/preferredprotocols
- /api/v1/presets
- /api/v1/presetvars
- /api/v1/devicevars
- /api/v1/protocols
- /api/v1/redundancy
- /api/v1/restart
- /api/v1/satdetails
- /api/v1/satellites
- /api/v1/satoperators
- /api/v1/schedule
- /api/v1/streamkeys
- /api/v1/subscribe
- /api/v1/tleimport
- /api/v1/thumbnail
- /api/v1/trackmem
- /api/v1/treeview
- /api/v1/uscreens
- /api/v1/user
- /api/v1/users
- /api/v1/userSettings
- /healthcheck
- /token

1.2.1 /token

The /token API call requests an access token from the server which can be used to authorize subsequent API calls.

All API calls except /token require a valid access token (JWT) to be passed with the header of the request as a "Authorization: Bearer ..." header or the server will abort with a 401 NOT AUTHENTICATED error.

Supported HTTP methods: POST

POST /token

Requests an access token for the user specified with the parameters added to this call. The parameter to this request must be passed as HTML form style parameters (application/x-www-form-urlencoded). The same end point is used to get a new and to renew an existing token.

Parameters:

Name	**Value**
username	The name of the user to log in. Must be a valid user name known to the sat-nms M&C application.
password	The password supplied by the user for login (clear text).
grant_type	One of 'password' or 'refresh_token'
refresh_token	A valid refresh token, making the backend to reply with a new version of the existing access token linked to this refresh token.

The *grant_type* parameter is mandatory, if it is set to 'password' the parameters*password* and *username* are requires as well. If *grant_type* is set to 'refresh_token', the parameter *refresh_token* is mandatory as well.

On success the API call returns a TokenReply document containing beside other information the access token and a refresh token to renew the access token.

Return Codes:

Code	**Description**
200	OK, login was successful. A TokenReply document is replied.
400	*grant_type* is neiter 'password' nor 'refresh_token'
401	Login failed. In this case a ApiError document is returned, describing details of the error. This happens either with *grant_type*='password' and invalid credentials or with *grant_type*='refresh_token' and an invalid refresh token.

1.2.2 /api/v1/logout

Logs out the currently logged in user.

Supported HTTP methods: POST

POST /api/v1/logout

Logs out the currently logged in user. Requires a valid access token to be passed with the request header to identify the session to be terminated. In fact, this call does not logout all open sessions of a given user, it terminates exactly one session which is referenced by the token.

Parameters: none

Return Codes:

Code	Description
204	OK, logout was successful. No payload data follows.

1.2.3 /healthcheck

This API call can be used to check if the backend server responds to API calls. It works without being logged in.

Supported HTTP methods: GET,HEAD,OPTIONS

GET /healthcheck

Returns an Dictionary document listing some basic status information about the backend server. As this API call is with public access, it returns only a limited set of information.

The Dictionary document returned contains the following keys with its values:

Key	Description
PRODUCT	The name of this product
WEBSITE	The SatService website URL
VERSION	The software version actually running
SERIALNO	The software serial number

Parameters: none

Return Codes:

Code	Description
200	OK, the Dictionary with the status information is returned as payload.

1.2.4 /api/v1/user

Delivers some information about the user currently logged in. Returns aUserProperties document describing the user. This document contains -- beside other information -- the *privilege level* of the user which must be known to decide which user interface elements must be shown in a read-only way because the user is not authorized to change these parameter (but he may read them if logged in).

This API end point also may be used with PUT to change the password of the actually logged in user.

Supported HTTP methods: GET, PUT, HEAD, OPTIONS

GET /api/v1/user

Returns a UserProperties document describing the currently logged in user.

Expected Payload: none

Replied Payload: a UserProperties document with information about the user who issued this call.

Parameters: none

Return Codes:

Code	Description
200	OK, returns a UserProperties document describing the currently logged in user. If no information about the currently logged in user is available, a dummy Document describing a user with the name 'unknown' and the privilege level '0' is returned.
401	not logged in. In this case a ApiError document is returned, describing details of the error.

PUT /api/v1/user

Changes the password of the user issuing this call. Returns the (updated)UserProperties for the currently logged in user.

Expected Payload: a UserProperties document containing the new password for the actual user. All other fields of the the document are ignored.

Replied Payload: a UserProperties document with the updated user information.

Parameters: none

Return Codes:

Code	Description
200	OK, returns a UserProperties with the updated data for the actually logged in user.

Code	Description
401	not logged in. In this case a ApiError document is returned, describing details of the error.

1.2.5 /api/v1/users

Reads or writes the list of users known to the backend. When editing the user list, the list must be read with a GET and later sent back with a PUT with all fields in each UserProperties record.

Supported HTTP methods: GET, PUT, HEAD, OPTIONS

GET /api/v1/users

Gets a UserList document containing the list of all known users.

Expected Payload: none

Replied Payload: a UserList document containing the list of all known users.

Parameters: none

Return Codes:

Code	Description
200	OK, returns a UserList containing the list of all known users.
401	not logged in. In this case a ApiError document is returned, describing details of the error.

PUT /api/v1/users

Send a UserList document containing the list of all known users to the backend. The UserList replaces the previous one. Already logged in users remain logged in, even if they are deleted from the list or if their password was changed.

Expected Payload: The UserList document to write.

Replied Payload: The UserList document echoed back.

Parameters: none

Return Codes:

Code	Description
200	OK, returns the written UserList echoed back.

Code	Description
400	cannot change the password. Probably the user logged in by LDAP. In this case a ApiError document is returned, describing details of the error.
401	not logged in. In this case a ApiError document is returned, describing details of the error.

1.2.6 /api/v1/mnclist

Delivers a list containing some information about the M&C systems managed by the backend. Returns a MncList document containing this list.

Supported HTTP methods: GET, HEAD, OPTIONS

GET /api/v1/mnclist

Returns a MncList document containing the list of managed M&C systems and their properties.

Parameters: none

Return Codes:

Code	Description
200	OK, returns a MncList document containing the list of managed M&C systems.
401	not logged in. In this case a ApiError document is returned, describing details of the error.

1.2.7 /api/v1/activebackend

Delivers an ActiveBackend document showing the name of the backend which is actually considered to be active for M&C redundancy switching. Also permits to change this using a PUT call.

Supported HTTP methods: GET, PUT, HEAD, OPTIONS

GET /api/v1/activebackend

Returns an ActiveBackend document showing the name of the backend which is actually considered to be active for M&C redundancy switching.

Parameters: none

Expected Payload: none.

Replied Payload: Returns an ActiveBackend document showing the name of the backend which is actually considered to be active for M&C redundancy switching.

Return Codes:

Code	Description
200	OK, returns the expected document
401	not logged in. In this case a ApiError document is returned, describing details of the error.

PUT /api/v1/activebackend

Sets the name of the of the backend which is actually considered to be active for M&C redundancy switching.

Parameters: none

Expected Payload: An ActiveBackend document containing the name of the backend which shall be considered to be active for M&C redundancy switching.

Replied Payload: Returns an ActiveBackend document showing the name of the backend which is actually considered to be active for M&C redundancy switching.

Return Codes:

Code	Description
200	OK, the name of the active backend has been updated. Returns the expected document
401	not logged in. In this case a ApiError document is returned, describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

1.2.8 /api/v1/peek

Reads ('peeks') the Parameter or Range definition addressed by {messageId} from the backend's message cache. If no such message is present in the cache, the M&C server is asked to deliver the message. The API call waits up to 3 seconds for the M&C server to answer, then a 404 error is returned. Peeking parameters is a very inefficient way to get a parameter value. It only should be used if a parameter value is required only once (without subsequent updates) or for testing purposes. Section WebSocket Communication describes the subscription model using STOMP/websockets which is the preferred way to read/update parameters in the UI.

The {messageId} supplied to /api/v1/peek must be a complete message identifier consisting of M&C-, device- and parameter-ID.

The document returned on success is either a Parameter or a Range, depending on the {messageId} supplied. For details about messages and message identifiers, see section

Message later in this document.

Supported HTTP methods: GET, HEAD, OPTIONS

GET /api/v1/peek/{messageId}

Returns a Message document describing the requested parameter.

Parameters:

Parameter	Description
messageId	The requested messageId. Must be a complete message identifier consisting of M&C-, device- and parameter-ID.

Return Codes:

Code	Description
200	OK, returns a Message document describing the requested parameter.
401	not logged in. In this case a ApiError document is returned, describing details of the error.
404	no parameter with the given messageId exists. In this case aApiError document is returned, describing details of the error.

1.2.9 /api/v1/poke

Writes ('pokes') a Parameter or Range message to the M&C server. The message to be written must be supplied as POST data.

Supported HTTP methods: POST, HEAD, OPTIONS

POST /api/v1/poke

Expects exactly one Parameter document as POST data. This creates and distributes the message on the M&C server addressed by the M&C identifier contained in the supplied message. This does not check if the specified destination exists. Messages sent to non-existing destinations are silently ignored, also result in a 201 response.

Parameters: none

Return Codes:

Code	Description
201	Success, echoes the message created.

Code	Description
401	not logged in. In this case a ApiError document is returned, describing details of the error.

1.2.10 /api/v1/makewebsocket

Creates a new websocket to connect to. A unique ID is generated for this websocket and returned as a WebsocketId document in the call's response body.

Supported HTTP methods: POST, HEAD, OPTIONS

POST /api/v1/makewebsocket

Creates a new websocket to connect to. A unique ID is generated for this websocket and returned as a WebsocketId document in the call's response body.

The WebsocketId is required to open the socket itself and for all subscribe / unsubscribe operations on this websocket. See section Websocket Communication for details about how to open a websocket.

Parameters: none

Return Codes:

Code	Description
201	Success, delivers the ID of the created websocket in the response body (data model WebsocketId).
401	not logged in. In this case a ApiError document is returned, describing details of the error.

1.2.11 /api/v1/subscribe

Subscribes or unsubscribes the delivery of one or more message IDs over the websocket connection. Whether to subscribe or unsubscribe is defined by the HTTP method.

Supported HTTP methods: POST, DELETE, HEAD, OPTIONS

POST /api/v1/subscribe/{websocketId}/{messageId}

Subscribes for the reception of messages with the given messageId over the websocket identified by websocketId. The requested message and all subsequent updates are sent over the websocket connection.

Returns a SubscribeList with the response body containing the message IDs which have been successfully subscribed (in this case of a single subscription the list contains either one or no message IDs).

The API call does not check if the remote M&C system can deliver a message with the given ID. Messages may become available later, this is a common situation.

Parameters:

Parameter	Description
websocketID	Identifies the websocket to send the message and subsequent updates to. See section Websocket Communication how to obtain a websocketID and how to open a websocket.
messageId	Identifies the message / parameter to subscribe. Must be a complete sat-nms message identifier including the M&C ID.

Return Codes:

Code	Description
201	Success, a SubscribeList is returned with the response body containing the (single) subscribed message ID.
401	Not logged in. In this case a ApiError document is returned, describing details of the error.
403	Invalid websocketId. The websocketId referenced in the URL either does not exist or the session which sent the request does not own this websocket. An empty SubscribeList is returned with the response body.

POST /api/v1/subscribe/{websocketId}

Subscribes for the reception of messages of multiple message IDs over a websocket. This variant of the API call expects a SubscribeList data object in the request body. This data object contains the message IDs to subscribe for.

The subscription requests are forwarded and are checked if successful one by one. The API call returns a SubscribeList in the response body containing the message IDs which have been successfully subscribed.

The API call does not check if the remote M&C system can deliver messages with the given IDs. Messages may become available later, this is a common situation.

Parameters:

Parameter	Description
websocketID	Identifies the websocket to send the message and subsequent updates to. See section Websocket Communication how to obtain a websocketID and how to open a websocket.

Return Codes:

Code	Description
201	Success, a SubscribeList is returned with the response body containing the subscribed message IDs.
401	Not logged in. In this case a ApiError document is returned, describing details of the error.
403	Invalid websocketId. The websocketId referenced in the URL either does not exist or the session which sent the request does not own this websocket. An empty SubscribeList is returned with the response body.

DELETE /api/v1/subscribe/{websocketId}/{messageId}

Unsubscribes the reception of messages with the given messageId over the websocket identified by websocketId.

Returns a SubscribeList with the response body containing the message IDs which have been successfully unsubscribed (in this case the list contains either one or no message IDs).

The API call does not check if there is a subscription pending for the given parameters. If a subscription is canceled which never has been done or which has been unsubscribed before, this is silently ignored.

Parameters:

Parameter	Description
websocketID	Identifies the websocket to unsubscribe. See section Websocket Communication how to obtain a websocketID and how to open a websocket.
messageId	Identifies the message / parameter to unsubscribe. Must be a complete sat-nms message identifier including the M&C ID.

Return Codes:

Code	Description
200	Success, a SubscribeList is returned with the response body containing the (single) unsubscribed message ID.
401	Not logged in. In this case a ApiError document is returned, describing details of the error.

Code	Description
403	Invalid websocketId. The websocketId referenced in the URL either does not exist or the session which sent the request does not own this websocket. An empty SubscribeList is returned with the response body.

DELETE /api/v1/subscribe/{websocketId}

Unsubscribes the reception of multiple message IDs over a websocket. This variant of the API call expects a SubscribeList data object in the request body. This data object contains the message IDs to unsubscribe.

The subscription requests are forwarded and are checked if successful one by one. The API call returns a SubscribeList in the response body containing the message IDs which have been successfully subscribed.

The API call does not check if there are subscriptions pending for the given message IDs. If a subscription is canceled which never has been done or which has been unsubscribed before, this is silently ignored.

Parameters:

Parameter	Description
websocketID	Identifies the websocket to unsubscribe. See section Websocket Communication how to obtain a websocketID and how to open a websocket.

Return Codes:

Code	Description
201	Success, a SubscribeList is returned with the response body containing the unsubscribed message IDs.
401	Not logged in. In this case a ApiError document is returned, describing details of the error.
403	Invalid websocketId. The websocketId referenced in the URL either does not exist or the session which sent the request does not own this websocket. An empty SubscribeList is returned with the response body.

1.2.12 /api/v1/uscreens

Gives access to the user defined screen definitions stored on the primary M&C. Screen definitions may be listed, read or written with this call.

Supported HTTP methods: GET, POST, HEAD, OPTIONS

GET /api/v1/uscreens

Returns a list of all available uscreens.

Parameters: none.

Expected Payload: none.

Replied Payload: Returns an ItemList containing the names of all available user screens. The list is read from the primary M&C.

Return Codes:

Code	Description
200	OK, returns an ItemList containing the names of all available user screens.
400	the primary M&C is offline, the list cannot be read. In this case aApiError document is returned, describing details of the error.
401	not logged in. In this case a ApiError document is returned, describing details of the error.

GET /api/v1/uscreens/{name}

Returns a ScreenDefinition document describing the requested screen. The screen definition is read from the primary M&C.

Parameters:

Parameter	Description
name	The name of the screen to load. Please note, that screen names are case sensitive, the name must match exactly to be loaded.

Expected Payload: none.

Replied Payload: Returns a ScreenDefinition document describing the requested screen.

Return Codes:

Code	Description
200	OK, returns a ScreenDefinition document describing the requested screen.
400	the primary M&C is offline, the screen cannot be read. In this case aApiError document is returned, describing details of the error.
401	not logged in. In this case a ApiError document is returned, describing details of the error.

Code	Description
404	no screen definition with the given name exists. In this case aApiError document is returned, describing details of the error.

POST /api/v1/uscreens/{name}

Stores a ScreenDefinition at the primary M&C. If the screen already exists, it gets overwritten.

Parameters:

Parameter	Description
name	The name of the screen to store. Screen definition names may consist of the characters A-Z, a-z, 0-9 and the hyphen character.

Expected Payload: The ScreenDefinition to store.

Replied Payload: Echoes back the written ScreenDefinition.

Return Codes:

Code	Description
200	OK, echoes back the written ScreenDefinition.
400	the primary M&C is offline, the screen cannot be written. In this case aApiError document is returned, describing details of the error.
401	not logged in. In this case a ApiError document is returned, describing details of the error.

DELETE /api/v1/uscreens/{name}

Deletes a ScreenDefinition at the primary M&C.

Parameters:

Parameter	Description
name	The name of the screen to delete. Screen definition names may consist of the characters A-Z, a-z, 0-9 and the hyphen character.

Expected Payload: none.

Replied Payload: none.

Return Codes:

Code	Description
204	OK, the screen has been deleted.
400	the primary M&C is offline, the screen cannot be deleted. In this case aApiError document is returned, describing details of the error.
401	not logged in. In this case a ApiError document is returned, describing details of the error.
404	not found. No screen of this name exists. In this case aApiError document is returned, describing details of the error.

1.2.13 /api/v1/dscreens

Gives access to the device screen definitions stored on the primary M&C. Screen definitions may be listed, read or written with this call.

Supported HTTP methods: GET, POST, HEAD, OPTIONS

GET /api/v1/dscreens

Returns a list of all available dscreens. The list is read from the primary M&C.

Parameters: none.

Expected Payload: none.

Replied Payload: Returns an ItemList containing the names of all available device screens.

Return Codes:

Code	Description
200	OK, returns an ItemList containing the names of all available device screens.
400	the primary M&C is offline, the list cannot be read. In this case aApiError document is returned, describing details of the error.
401	not logged in. In this case a ApiError document is returned, describing details of the error.

GET /api/v1/dscreens/{name}

Returns a ScreenDefinition document describing the requested screen. The screen is read from the primary M&C.

Parameters:

Parameter	Description

Parameter	Description
name	The name of the screen to load. Please note, that screen names are case sensitive, the name must match exactly to be loaded.

Expected Payload: none.

Replied Payload: Returns a ScreenDefinition document describing the requested screen.

Return Codes:

Code	Description
200	OK, returns a ScreenDefinition document describing the requested screen.
400	the primary M&C is offline, the screen cannot be read. In this case aApiError document is returned, describing details of the error.
401	not logged in. In this case a ApiError document is returned, describing details of the error.
404	no screen definition with the given name exists. In this case aApiError document is returned, describing details of the error.

POST /api/v1/dscreens/{name}

Stores a ScreenDefinition at the primary M&C. If the screen already exists, it gets overwritten.

Parameters:

Parameter	Description
name	The name of the screen to store. Screen definition names may consist of the characters A-Z, a-z, 0-9 and the hyphen character.

Expected Payload: The ScreenDefinition to store.

Replied Payload: Echoes back the written ScreenDefinition.

Return Codes:

Code	Description
200	OK, echoes back the written ScreenDefinition.
400	the primary M&C is offline, the screen cannot be written. In this case aApiError document is returned, describing details of the error.
401	not logged in. In this case a ApiError document is returned, describing details of the error.

DELETE /api/v1/dscreens/{name}

Deletes a ScreenDefinition at the primary M&C.

Parameters:

Parameter	Description
name	The name of the screen to delete. Screen definition names may consist of the characters A-Z, a-z, 0-9 and the hyphen character.

Expected Payload: none.

Replied Payload: none.

Return Codes:

Code	Description
204	OK, the screen has been deleted.
400	the primary M&C is offline, the screen cannot be deleted. In this case aApiError document is returned, describing details of the error.
401	not logged in. In this case a ApiError document is returned, describing details of the error.
404	not found. No screen of this name exists. In this case aApiError document is returned, describing details of the error.

1.2.14 /api/v1/dframes

Gives access to the device frame definitions stored on the primary M&C. Device frame definitions may be listed, read or written with this call.

Supported HTTP methods: GET, POST, HEAD, OPTIONS

GET /api/v1/dframes

Returns a list of all available device frame definitions. The list is read from the primary M&C.

Parameters: none.

Expected Payload: none.

Replied Payload: Returns an ItemList containing the names of all available device frame definitions.

Return Codes:

Code	Description
200	OK, returns an ItemList containing the names of all available device frame definitions.
400	the primary M&C is offline, the list cannot be read. In this case aApiError document is returned, describing details of the error.
401	not logged in. In this case a ApiError document is returned, describing details of the error.

GET /api/v1/dframes/{name}

Returns a DeviceFrameDefinition document describing the requested frame. The frame definition is read from the primary M&C.

Parameters:

Parameter	Description
name	The name of the frame definition to load. Please note, that frame names are case sensitive, the name must match exactly to be loaded.

Expected Payload: none.

Replied Payload: Returns a DeviceFrameDefinition document describing the requested frame.

Return Codes:

Code	Description
200	OK, returns a DeviceFrameDefinition document describing the requested frame.
400	the primary M&C is offline, the frame definition cannot be read. In this case a ApiError document is returned, describing details of the error.
401	not logged in. In this case a ApiError document is returned, describing details of the error.
404	no frame definition with the given name exists. In this case aApiError document is returned, describing details of the error.

POST /api/v1/dframes/{name}

Stores a DeviceFrameDefinition at the primary M&C. If the frame definition already exists, it gets overwritten.

Parameters:

Parameter	Description
name	The name of the frame to store. DeviceFrame definition names may consist of the characters A-Z, a-z, 0-9 and the hyphen character.

Expected Payload: The DeviceFrameDefinition to store.

Replied Payload: Echoes back the written DeviceFrameDefinition.

Return Codes:

Code	Description
200	OK, echoes back the written DeviceFrameDefinition.
400	the primary M&C is offline, the frame definition cannot be written. In this case a ApiError document is returned, describing details of the error.
401	not logged in. In this case a ApiError document is returned, describing details of the error.

DELETE /api/v1/dframe/{name}

Deletes a DeviceFrameDefinition at the primary M&C.

Parameters:

Parameter	Description
name	The name of the frame definition do delete. DeviceFrame definition names may consist of the characters A-Z, a-z, 0-9 and the hyphen character.

Expected Payload: none.

Replied Payload: none.

Return Codes:

Code	Description
204	OK, the device frame has been deleted.
400	the primary M&C is offline, the frame definition cannot be deleted. In this case a ApiError document is returned, describing details of the error.
401	not logged in. In this case a ApiError document is returned, describing details of the error.

Code	Description
404	not found. No device frame of this name exists. In this case aApiError document is returned, describing details of the error.

1.2.15 /api/v1/macros

Permits to load and save sat-nms macro definitions as well as to read the list of macros defined on a given M&C.

Supported HTTP methods: GET, PUT, DELETE, HEAD, OPTIONS

GET /api/v1/macros/{mncName}

Returns an ItemList document containing the list of macros defined on the given M&C.

Parameters:

Parameter	Description
mncName	The name of the M&C from which the macro list shall be loaded.

Return Codes:

Code	Description
200	OK, returns an ItemList containing the requested macro list.
401	not logged in. In this case a ApiError document is returned, describing details of the error.
404	no M&C with the given name exists. In this case a ApiError document is returned, describing details of the error.

GET /api/v1/macros/{mncName}/{macroName}

Returns a Macro document containing the requested macro.

Parameters:

Parameter	Description
mncName	The name of the M&C from which the macro shall be loaded.
macroName	The name of the macro to load.

Return Codes:

Code	Description
200	OK, returns a Macro containing the requested macro.
401	not logged in. In this case a ApiError document is returned, describing details of the error.
404	either M&C with the given name exists or at the known M&C there exists no macro with the requested name. In this case a ApiError document is returned, describing details of the error.

PUT /api/v1/macros/{mncName}/{macroName}

Saves a macro on the given M&C. Expects Macro document containing the macro text to save as request body. The *name* field in the Macro document will be ignored, the macro will be saved under the name given in the URL. As every PUT request, this will either overwrite an existing macro with this name or create a new macro definition at the M&C if no macro with this name exists.

Parameters:

Parameter	Description
mncName	The name of the M&C on which the macro shall be saved.
macroName	The name of the macro to save.

Return Codes:

Code	Description
200	OK, returns the Macro containing the saved macro. This will have the *name* field set the name under which the macro actually has been saved.
401	not logged in. In this case a ApiError document is returned, describing details of the error.
404	no M&C with the given name exists. In this case a ApiError document is returned, describing details of the error.

DELETE /api/v1/macros/{mncName}/{macroName}

Deletes a macro at the given M&C.

Parameters:

Parameter	Description

Parameter	Description
mncName	The name of the M&C on which the macro shall be deleted.
macroName	The name of the macro to delete.

Return Codes:

Code	Description
200	OK, returns the Macro containing the deleted macro.
401	not logged in. In this case a ApiError document is returned, describing details of the error.
404	either M&C with the given name exists or at the known M&C there exists no macro with the requested name. In this case a ApiError document is returned, describing details of the error.

1.2.16 /api/v1/playmacro

Executes a macro on a given M&C.

Supported HTTP methods: POST, HEAD, OPTIONS

POST /api/v1/playmacro/{mncName}

Executes the referenced macro. Expects the macro to play as aMacro body, delegates the request to the M&C.

Parameters:

Parameter	Description
mncName	The name of the M&C at which the macro shall be loaded and executed.

Return Codes:

Code	Description
200	OK, returns the Macro containing the executed macro.
401	not logged in. In this case a ApiError document is returned, describing details of the error.
404	the M&C with the given name does nit exist. In this case aApiError document is returned, describing details of the error.

POST /api/v1/playmacro/{mncName}/{macroName}

Executes the referenced macro. Expects no request body, simply delegates the request to the M&C.

Parameters:

Parameter	Description
mncName	The name of the M&C at which the macro shall be loaded and executed.
macroName	The name of the macro to execute.

Return Codes:

Code	Description
200	OK, returns the Macro containing the executed macro.
401	not logged in. In this case a ApiError document is returned, describing details of the error.
404	either M&C with the given name exists or at the known M&C there exists no macro with the requested name. In this case a ApiError document is returned, describing details of the error.

1.2.17 /api/v1/presets

Permits to load and save sat-nms device preset definitions as well as to read the list of presets defined on a given M&C.

Supported HTTP methods: GET, PUT, HEAD, OPTIONS

GET /api/v1/presets/{mncName}

Returns an ItemList document containing the list of preset directories found on the given M&C. This is the list of device drivers which have at least one preset saved on this M&C.

Parameters:

Parameter	Description
mncName	The name of the M&C from which the preset directory list shall be loaded.

Return Codes:

Code	Description

Code	Description
200	OK, returns an ItemList containing the requested preset directory list.
401	not logged in. In this case a ApiError document is returned, describing details of the error.
404	no M&C with the given name exists. In this case a ApiError document is returned, describing details of the error.

If the given M&C name exists, but there are no presets stored at all, the request returns 200 OK and an empty list.

GET /api/v1/presets/{mncName}/{driverName}

Returns an ItemList document containing the list of presets defined this device driver on the given M&C.

Parameters:

Parameter	Description
mncName	The name of the M&C from which the preset list shall be loaded.
driverName	The name of the device driver for which the preset list shall be loaded.

Return Codes:

Code	Description
200	OK, returns an ItemList containing the requested preset list.
401	not logged in. In this case a ApiError document is returned, describing details of the error.
404	no M&C with the given name exists. In this case a ApiError document is returned, describing details of the error.

If the given M&C name exists, but there are no presets stored for the driver type, the request returns 200 OK and an empty list.

GET /api/v1/presets/{mncName}/{driverName}/{presetName}

Returns a DevicePreset document containing the requested preset.

Parameters:

Parameter	Description

Parameter	Description
mncName	The name of the M&C from which the preset shall be loaded.
driverName	The name of the device driver for which the preset shall be loaded.
presetName	The name of the preset to load.

Return Codes:

Code	Description
200	OK, returns a DevicePreset containing the requested preset.
401	not logged in. In this case a ApiError document is returned, describing details of the error.
404	either M&C with the given name exists or at the known M&C there exists no preset with the requested driver/name combination. In this case a ApiError document is returned, describing details of the error.

PUT /api/v1/presets/{mncName}/{driverName}/{presetName}

Saves a preset on the given M&C. Expects DevicePreset document containing the preset text to save as request body. The *name* and *driver* fields in the DevicePreset document will be ignored, the preset will be saved under the driver / name given in the URL. As every PUT request, this will either overwrite an existing preset with this name or create a new preset definition at the M&C if no preset with this name exists.

Parameters:

Parameter	Description
mncName	The name of the M&C on which the preset shall be saved.
driverName	The name of the device driver for which the preset shall be saved.
presetName	The name of the preset to save.

Return Codes:

Code	Description
200	OK, returns the DevicePreset containing the saved preset. This will have the *name* and *driver* fields set the values under which the preset actually has been saved.

Code	Description
401	not logged in. In this case a ApiError document is returned, describing details of the error.
404	no M&C with the given name exists. In this case a ApiError document is returned, describing details of the error.

DELETE /api/v1/presets/{mncName}/{driverName}/{presetName}

Deletes a preset from the given M&C. Expects no request body.

Parameters:

Parameter	Description
mncName	The name of the M&C from which the preset shall be deleted.
driverName	The name of the device driver for which the preset shall be deleted.
presetName	The name of the preset to delete.

Return Codes:

Code	Description
200	OK, returns the DevicePreset containing the deleted preset.
401	not logged in. In this case a ApiError document is returned, describing details of the error.
404	no M&C with the given name exists or no device preset with that name / device driver could be found. In this case a ApiError document is returned, describing details of the error.

1.2.18 /api/v1/applypreset

Applies a preset on a given M&C.

Supported HTTP methods: POST, HEAD, OPTIONS

POST /api/v1/applypreset/{mncName}/{deviceName}

Applies a preset to the given device. Expects a DevicePreset as request body, then delegates the request to the M&C.

Parameters:

Parameter	Description
mncName	The name of the M&C at which the preset shall be applied.
deviceName	The name of the device to apply the preset to.

Return Codes:

Code	Description
200	OK, returns the DevicePreset containing the applied preset.
401	not logged in. In this case a ApiError document is returned, describing details of the error.
404	no M&C with the given name exists. In this case a ApiError document is returned, describing details of the error.

POST /api/v1/applypreset/{mncName}/{driverName}/{deviceName}/{presetName}

Applies the referenced preset to the given device. Expects no request body, simply delegates the request to the M&C.

Parameters:

Parameter	Description
mncName	The name of the M&C at which the preset shall be applied.
driverName	The name of the device driver for which the preset shall be applied.
deviceName	The name of the device to apply the preset to.
presetName	The name of the preset to apply.

Return Codes:

Code	Description
200	OK, returns the DevicePreset containing the applied preset.
401	not logged in. In this case a ApiError document is returned, describing details of the error.
404	either M&C with the given name exists or at the known M&C there exists no driver/device/preset with the requested name. In this case a ApiError document is returned, describing details of the error.

1.2.19 /api/v1/collectpreset

Collects a preset on a given M&C. This tells the referenced device to collect its settings into a preset and to store this preset in the M&C with the given name.

Supported HTTP methods: POST, HEAD, OPTIONS

POST /api/v1/collectpreset/{mncName}/{driverName}/{deviceName}/{presetName}

Collects a preset on a given M&C. This tells the referenced device to collect its settings into a preset and to store this preset in the M&C with the given name. Expects no request body, simply delegates the request to the M&C.

Parameters:

Parameter	Description
mncName	The name of the M&C at which the preset shall be collected.
driverName	The name of the device driver for which the preset shall be collected.
deviceName	The name of the device to collect the preset from.
presetName	The name of the preset to store.

Return Codes:

Code	Description
200	OK, returns the DevicePreset containing the collected preset.
401	not logged in. In this case a ApiError document is returned, describing details of the error.
404	either M&C with the given name exists or at the known M&C there exists no device with the requested name. In this case a ApiError document is returned, describing details of the error.

1.2.20 /api/v1/presetvars

Delivers the list of driver variable which may be stored in a preset for a given device driver.

Supported HTTP methods: GET, HEAD, OPTIONS

GET /api/vi/presetvars/{mncName}/{driverName}

Returns a PresetVars document containing the requested list of preset-storable variables. The backend reads the device driver *driverName.device* from the given M&C, compiles the driver text and selects the preset storable variables from the compiled list. The returned PresetVars document contains the Range definitions for all variables of the driver which may appear in a device preset. The *messageId* field in each Range object is set to the name of the variable.

Parameters:

Parameter	Description
mncName	The name of the M&C from which the variable list shall be loaded.
driverName	The name of the device driver for which the variable list shall be evaluated.

Return Codes:

Code	Description
200	OK, returns a PresetVars document containing the requested variable list.
401	not logged in. In this case a ApiError document is returned, describing details of the error.
404	either no M&C with the given name exists, or at the known M&C there exists no device driver with the given name. In this case a ApiError document is returned, describing details of the error.

1.2.21 /api/v1/devicevars

Delivers the list of all variables a device defines

Supported HTTP methods: GET, HEAD, OPTIONS

GET /api/vi/devicevars/{mncName}/{deviceName}

Returns a DeviceVars document containing the requested list of variables. The backend evaluates the driver for the given device, compiled the device driver and gets the variable list from this. The returned DeviceVars document contains the Range definitions for all variables of the driver which may appear in a device preset. The *messageId* field in each Range object is set to the name of the variable.

The API call returns the variable list as it exists at compile time at system startup. Changes made dynamically to the list of variables or to some variable ranges are not contained in the list

Parameters:

Parameter	Description
mncName	The name of the M&C from which the variable list shall be loaded.

Parameter	Description
deviceName	The name of the device for which the variable list shall be evaluated.

Return Codes:

Code	Description
200	OK, returns a PresetVars document containing the requested variable list.
401	not logged in. In this case a ApiError document is returned, describing details of the error.
404	either no M&C with the given name exists, or at the known M&C there exists no device driver with the given name. In this case a ApiError document is returned, describing details of the error.

1.2.22 /api/v1/schedule

Permits to load and save the macro scheduler's schedule at a given M&C.

When subscribing for a '.SYSTEM.schedule' parameter of a M&C, the front end receives a message which indicates 'an updated schedule is available' instead of sending the bulky schedule itself over the websocket connection.

After the front end received this message, it can use the /api/v1/schedule API call to retrieve the schedule as a Schedule data object.

Supported HTTP methods: GET, PUT, HEAD, OPTIONS

GET /api/v1/schedule/{mncName}

Returns a Schedule document containing the actual system schedule from this M&C.

Parameters:

Parameter	Description
mncName	The name of the M&C from which the schedule shall be loaded.

Return Codes:

Code	Description
200	OK, returns a Schedule document containing the requested schedule.

Code	Description
401	not logged in. In this case a ApiError document is returned, describing details of the error.
404	no M&C with the given name exists. In this case a ApiError document is returned, describing details of the error.

PUT /api/v1/schedule/{mncName}

Overwrites the system schedule at the given M&C. Expects a Schedule document containing new schedule to set.

Parameters:

Parameter	Description
mncName	The name of the M&C on which the preset shall be saved.

Return Codes:

Code	Description
200	OK, returns the Schedule containing the saved schedule.
401	not logged in. In this case a ApiError document is returned, describing details of the error.
404	no M&C with the given name exists. In this case a ApiError document is returned, describing details of the error.

1.2.23 /api/v1/healthcheck

This API call delivers some status information about the running backend server. Unlike the version with public access, this version of the health check contains the complete set of information.

Supported HTTP methods: GET,HEAD,OPTIONS

GET /api/v1/healthcheck

Returns an Dictionary document listing some status information about the backend server.

The Dictionary document returned contains the following keys with its values:

Key	Description
PRODUCT	The name of this product

Key	Description
WEBSITE	The SatService website URL
VERSION	The software version actually running
SERIALNO	The software serial number
UPTIME	The server uptime
MEMORY	The memory usage of the backend
SESSIONS	The number of open sessions
WEBSOCKS	The number of open (message) websocket connections

Parameters: none

Return Codes:

Code	Description
200	OK, the Dictionary with the status information is returned as payload.

1.2.24 /api/v1/dbstate

This API call delivers some status information about the backend's connection to the database server. It may be used by a frontend to evaluate if the backend actually is connected to the (read-only) backeup database and therefore no changes to the database content are allowed

Supported HTTP methods: GET,HEAD,OPTIONS

GET /api/v1/dbstate

Returns a SatDbState document listing some status information about the database connection.

Parameters: none

Return Codes:

Code	Description
200	OK, the SatDbState document with the status information is returned as payload.

1.2.25 /api/v1/dbversion

This API call delivers the version information about the database connected to the backend.

Supported HTTP methods: GET,HEAD,OPTIONS

GET /api/v1/dbversion

Returns a SatDbVersion document showing the actual version of the database and the version required by the backend.

Parameters: none

Return Codes:

Code	Description
200	OK, the SatDbState document with the status information is returned as payload.

1.2.26 /api/v1/satellites

This API endpoint gives access to various functions of the sat-nms satellite database, permitting to read, add, change or delete entries at the database

Supported HTTP methods: GET, POST, PATCH, DELETE, HEAD, OPTIONS**GET /api/v1/satellites**

Returns the list of stored satellites as an array ofSatDbSatellite data objects.

Parameters: none

Expected Payload: none

Replied Payload: The list of stored satellites as an array ofSatDbSatellite data objects.

Return Codes:

Code	Description
200	OK, returns a JSON document with all satellites and their attributes.
401	not logged in. In this case anApiError document is returned,describing details of the error.

GET /api/v1/satellites/{satelliteId}

Returns the basic properties of the satellite addressed by {satelliteId} as aSatDbSatellite data object.

Parameters:

Parameter	Description
satelliteId	Id of the satellite.

Expected Payload: none

Replied Payload: The requested SatDbSatellite data object.

Return Codes:

Code	Description
200	OK, returns a document describing the requested satellite.
401	not logged in. In this case an ApiError document is returned, describing details of the error.
404	satellite with the given *satelliteId* does not exists. In this case an ApiError document is returned, describing details of the error.

POST /api/v1/satellites

Creates a new satellite with the given attributes . The unique *satelliteId* is generated by the database backend and returned in the response document.

Parameters: none

Expected Payload: The SatDbSatellite data object to store. Any *satelliteId* definition in the source is ignored as this will be set by the database when the new entry is created. If other keys of the data object are missing, default values will be assigned these keys.

Replied Payload: The SatDbSatellite data object added to the database. The replied data object contains the *satelliteId* assigned to the newly created record.

Return Codes:

Code	Description
201	CREATED, the created SatDbSatellite data object is replied as payload
401	not logged in. In this case an ApiError document is returned,describing details of the error.
500	DB is not writable. In this case an ApiError document is returned,describing details of the error.

PATCH /api/v1/satellites/{satelliteId}

Updates an existing satellite with the given *satelliteId* from the attached SatDbSatellite data object.

Parameters:

Parameter	Description
satelliteId	Id of the satellite.

Expected Payload: The SatDbSatellite data object containing the fields to be updated in the database. If the data object contains a *satelliteId* key, the value is ignored. The database record is solely identified by *satelliteId* set in the URL, the *satelliteId* of the record is immutable. Keys not contained in the supplied data object are retained unchanged.

Replied Payload: The SatDbSatellite data object read from the database after the requested changes have been applied.

Return Codes:

Code	Description
200	OK, changes have been applied.
401	not logged in. In this case anApiError document is returned, describing details of the error.
404	satellite with the given *satelliteId* does not exists. In this case anApiError document is returned, describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

DELETE /api/v1/satellites/{satelliteId}

Remove a satellite from the database. This deletes the satellite's record from the satellite table and all records from other tables like beacons, TLEs etc which unambiguously depend on the deleted satellite.

Parameters:

Parameter	Description
satelliteId	Id of the satellite to delete

Expected Payload: none

Replied Payload: The SatDbSatellite data object which has been deleted.

Return Codes:

Code	Description
200	OK, the satellite has been deleted.
401	not logged in. In this case an ApiError document is returned, describing details of the error.
404	satellite with the given *satelliteId* does not exist. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case an ApiError document is returned,describing details of the error.

GET /api/v1/satellites/{satelliteId}/positions

Returns the list of satellite positions for a given satellite as an array of SatDbPosition data objects.

Parameters:

Parameter	Description
satelliteId	Id of the satellite.

Expected Payload: none

Replied Payload: The list of satellite positions as an array of SatDbPosition data objects. This array may be empty if there are no positions defined for this satellite.

Return Codes:

Code	Description
200	OK, returns a JSON document with the requested information
401	not logged in. In this case an ApiError document is returned, describing details of the error.
404	satellite with the given *satelliteId* does not exists. In this case an ApiError document is returned, describing details of the error.

GET /api/v1/satellites/{satelliteId}/beacons

Returns the list of satellite beacons for a given satellite as an array of SatDbBeacon data objects.

Parameters:

Parameter	Description
satelliteId	Id of the satellite.

Expected Payload: none

Replied Payload: The list of satellite beacons as an array ofSatDbBeacon data objects. This array may be empty if there are no beacons defined for this satellite.

Return Codes:

Code	Description
200	OK, returns a JSON document with the requested information
401	not logged in. In this case anApiError document is returned, describing details of the error.
404	satellite with the given *satelliteId* does not exists. In this case anApiError document is returned, describing details of the error.

GET /api/v1/satellites/{satelliteId}/beacons/{beaconId}

Returns the information stored for a particular satellite beacon addressed by {satelliteId} / {beaconId} as a SatDbBeacon data object.

Parameters:

Parameter	Description
satelliteId	Id of the satellite.
beaconId	Id of the beacon.

Expected Payload: none

Replied Payload: The requested SatDbBeacon data object.

Return Codes:

Code	Description
200	OK, returns a document describing the requested satellite.
401	not logged in. In this case anApiError document is returned, describing details of the error.

Code	Description
404	no satellite satellite beacon exists with the given *satelliteId* / *beaconId*. In this case an ApiError document is returned, describing details of the error.

POST /api/v1/satellites/{satelliteId}/beacons

Creates a new satellite beacon with the given attributes. {satelliteId} specifies the satellite for which the beacon shall be created. The unique *beaconId* is generated by the database backend and returned in the response document.

Parameters:

Parameter	Description
satelliteId	Id of the satellite.

Expected Payload: The SatDbBeacon data object to store. Any *beaconId* definition in the source is ignored as this will be set by the database when the new entry is created. The same applies to *satellite_Id*, the newly created SatDbBeacon data object will be linked to the satellite specified in the URL. If other keys of the data object are missing, default values will be assigned these keys.

Replied Payload: The SatDbBeacon data object added to the datbase. The replied data object contains the *beaconId* assigned to the newly created record.

Return Codes:

Code	Description
201	CREATED, the created SatDbBeacon data object is replied as payload
401	not logged in. In this case anApiError document is returned,describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

PATCH /api/v1/satellites/{satelliteId}/beacons/{beaconId}

Updates an existing satellite beacon with the given *satelliteId* / *beaconId* from the attached SatDbBeacon data object.

Parameters:

Parameter	Description

Parameter	Description
satelliteId	Id of the satellite.
beaconId	Id of the beacon.

Expected Payload: The SatDbBeacon data object containing the fields to be updated in the database. If the data object contains values for *beaconId* and / or *satelliteId*, these values are ignored. *beaconId* and *satelliteId* are immutable once a record has been created. Keys not contained in the supplied data object are retained unchanged.

Replied Payload: The SatDbBeacon data object read from the datbase after the requested changes have been applied.

Return Codes:

Code	Description
200	OK, changes have been applied.
401	not logged in. In this case anApiError document is returned, describing details of the error.
404	no satellite satellite beacon exists with the given *satelliteId* / *beaconId*. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

DELETE /api/v1/satellites/{satelliteId}/beacons/{beaconId}

Remove a satellite beacon from the database.

Parameters:

Parameter	Description
satelliteId	Id of the satellite the beacon belongs to.
beaconId	Id of the beacon to delete.

Expected Payload: none

Replied Payload: none

Return Codes:

Code	Description
204	NO CONTENT, the beacon has been deleted.
401	not logged in. In this case an ApiError document is returned,describing details of the error.
404	no satellite satellite beacon exists with the given *satelliteId* / *beaconId*. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case an ApiError document is returned,describing details of the error.

DELETE /api/v1/satellites/{satelliteId}/beacons

Remove all satellite beacons which belong to the given *satelliteId*.

Parameters:

Parameter	Description
satelliteId	Id of the satellite

Return Codes:

Code	Description
204	NO CONTENT, all beacons for the selected satellite have been deleted.
401	not logged in. In this case an ApiError document is returned,describing details of the error.
404	satellite with the given *satelliteId* does not exist. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case an ApiError document is returned,describing details of the error.

GET /api/v1/satellites/{satelliteId}/tcchans

Returns the list of satellite TC channels for a given satellite as an array of SatDbTc data objects.

Parameters:

Parameter	Description

Parameter	Description
satelliteId	Id of the satellite.

Expected Payload: none

Replied Payload: The list of satellite TC channels as an array ofSatDbTc data objects. This array may be empty if there are no TC channels defined for this satellite.

Return Codes:

Code	Description
200	OK, returns a JSON document with the requested information
401	not logged in. In this case anApiError document is returned, describing details of the error.
404	satellite with the given *satelliteId* does not exists. In this case anApiError document is returned, describing details of the error.

GET /api/v1/satellites/{satelliteId}/tcchans/{tcId}

Returns the information stored for a particular satellite TC channel addressed by {satelliteId} / {tcId} as a SatDbTc data object.

Parameters:

Parameter	Description
satelliteId	Id of the satellite.
tcId	Id of the TC channel.

Expected Payload: none

Replied Payload: The requested SatDbTc data object.

Return Codes:

Code	Description
200	OK, returns a document describing the requested satellite channel.
401	not logged in. In this case anApiError document is returned, describing details of the error.

Code	Description
404	no satellite TC channel exists with the given *satelliteId* / *tcId*. In this case an ApiError document is returned, describing details of the error.

POST /api/v1/satellites/{satelliteId}/tcchans

Creates a new satellite TC channel with the given attributes. {satelliteId} specifies the satellite for which the TC channel shall be created. The unique *tcId* is generated by the database backend and returned in the response document.

Parameters:

Parameter	Description
satelliteId	Id of the satellite.

Expected Payload: The SatDbTc data object to store. Any *tcId* definition in the source is ignored as this will be set by the database when the new entry is created. The same applies to *satelliteId*, the newly created SatDbTc data object will be linked to the satellite specified in the URL. If other keys of the data object are missing, default values will be assigned these keys.

Replied Payload: The SatDbTc data object added to the datbase. The replied data object contains the *tcId* assigned to the newly created record.

Return Codes:

Code	Description
201	CREATED, the created SatDbTc data object is replied as payload
401	not logged in. In this case an ApiError document is returned,describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

PATCH /api/v1/satellites/{satelliteId}/tcchans/{tcId}

Updates an existing satellite TC channel with the given *satelliteId* / *tcId* from the attached SatDbTc data object.

Parameters:

Parameter	Description
satelliteId	Id of the satellite.

Parameter	Description
tcId	Id of the TC channel.

Expected Payload: The SatDbTc data object containing the fields to be updated in the database. If the data object contains values for *tcId* and / or *satelliteId*, these values are ignored. *tcId* and *satelliteId* are immutable once a record has been created. Keys not contained in the supplied data object are retained unchanged.

Replied Payload: The SatDbTc data object read from the datbase after the requested changes have been applied.

Return Codes:

Code	Description
200	OK, changes have been applied.
401	not logged in. In this case an ApiError document is returned, describing details of the error.
404	no satellite satellite TC channel exists with the given *satelliteId* / *tcId*. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case an ApiError document is returned,describing details of the error.

DELETE /api/v1/satellites/{satelliteId}/tcchans/{tcId}

Remove a satellite TC channel from the database.

Parameters:

Parameter	Description
satelliteId	Id of the satellite the TC channel belongs to.
tcId	Id of the TC channel to delete.

Expected Payload: none

Replied Payload: none

Return Codes:

Code	Description

Code	Description
204	NO CONTENT, the TC channel has been deleted.
401	not logged in. In this case an ApiError document is returned,describing details of the error.
404	no satellite satellite TC channel exists with the given *satelliteId* / *tcId*. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case an ApiError document is returned,describing details of the error.

DELETE /api/v1/satellites/{satelliteId}/tcchans

Remove all satellite TC channels which belong to the given *satelliteId*.

Parameters:

Parameter	Description
satelliteId	Id of the satellite

Return Codes:

Code	Description
204	NO CONTENT, all TC channels for the selected satellite have been deleted.
401	not logged in. In this case an ApiError document is returned,describing details of the error.
404	satellite with the given *satelliteId* does not exist. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case an ApiError document is returned,describing details of the error.

GET /api/v1/satellites/{satelliteId}/tmchans

Returns the list of satellite TM channels for a given satellite as an array of SatDbTm data objects.

Parameters:

Parameter	Description

Parameter	Description
satelliteId	Id of the satellite.

Expected Payload: none

Replied Payload: The list of satellite TM channels as an array ofSatDbTm data objects. This array may be empty if there are no tm channels defined for this satellite.

Return Codes:

Code	Description
200	OK, returns a JSON document with the requested information
401	not logged in. In this case anApiError document is returned, describing details of the error.
404	satellite with the given *satelliteId* does not exists. In this case anApiError document is returned, describing details of the error.

GET /api/v1/satellites/{satelliteId}/tmchans/{tmId}

Returns the information stored for a particular satellite TM channel addressed by {satelliteId} / {tmId} as a SatDbTm data object.

Parameters:

Parameter	Description
satelliteId	Id of the satellite.
tmId	Id of the TM channel.

Expected Payload: none

Replied Payload: The requested SatDbTm data object.

Return Codes:

Code	Description
200	OK, returns a document describing the requested satellite channel.
401	not logged in. In this case anApiError document is returned, describing details of the error.

Code	Description
404	no satellite TM channel exists with the given *satelliteId* / *tmId*. In this case an ApiError document is returned, describing details of the error.

POST /api/v1/satellites/{satelliteId}/tmchans

Creates a new satellite TM channel with the given attributes. {satelliteId} specifies the satellite for which the tm channel shall be created. The unique *tmId* is generated by the database backend and returned in the response document.

Parameters:

Parameter	Description
satelliteId	Id of the satellite.

Expected Payload: The SatDbTm data object to store. Any *tmId* definition in the source is ignored as this will be set by the database when the new entry is created. The same applies to *satelliteId*, the newly created SatDbTm data object will be linked to the satellite specified in the URL. If other keys of the data object are missing, default values will be assigned these keys.

Replied Payload: The SatDbTm data object added to the datbase. The replied data object contains the *tmId* assigned to the newly created record.

Return Codes:

Code	Description
201	CREATED, the created SatDbTm data object is replied as payload
401	not logged in. In this case an ApiError document is returned,describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

PATCH /api/v1/satellites/{satelliteId}/tmchans/{tmId}

Updates an existing satellite TM channel with the given *satelliteId* / *tmId* from the attached SatDbTm data object.

Parameters:

Parameter	Description
satelliteId	Id of the satellite.

Parameter	Description
tmId	Id of the TM channel.

Expected Payload: The SatDbTm data object containing the fields to be updated in the database. If the data object contains values for *tmId* and / or *satelliteId*, these values are ignored. *tmId* and *satelliteId* are immutable once a record has been created. Keys not contained in the supplied data object are retained unchanged.

Replied Payload: The SatDbTm data object read from the datbase after the requested changes have been applied.

Return Codes:

Code	Description
200	OK, changes have been applied.
401	not logged in. In this case anApiError document is returned, describing details of the error.
404	no satellite satellite TM channel exists with the given *satelliteId* / *tmId*. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

DELETE /api/v1/satellites/{satelliteId}/tmchans/{tmId}

Remove a satellite TM channel from the database.

Parameters:

Parameter	Description
satelliteId	Id of the satellite the TM channel belongs to.
tmId	Id of the TM channel to delete.

Expected Payload: none

Replied Payload: none

Return Codes:

Code	Description

Code	Description
204	NO CONTENT, the TM channel has been deleted.
401	not logged in. In this case an ApiError document is returned,describing details of the error.
404	no satellite satellite TM channel exists with the given *satelliteId* / *tmId*. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case an ApiError document is returned,describing details of the error.

DELETE /api/v1/satellites/{satelliteId}/tmchans

Remove all satellite TM channels which belong to the given *satelliteId*.

Parameters:

Parameter	Description
satelliteId	Id of the satellite

Return Codes:

Code	Description
204	NO CONTENT, all TM channels for the selected satellite have been deleted.
401	not logged in. In this case an ApiError document is returned,describing details of the error.
404	satellite with the given *satelliteId* does not exist. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case an ApiError document is returned,describing details of the error.

GET /api/v1/satellites/{satelliteId}/i11params

Returns the list of I11 parameter sets for a given satellite as an array ofSatDbI11Data data objects.

Parameters:

Parameter	Description

Parameter	Description
satelliteId	Id of the satellite.

Expected Payload: none

Replied Payload: The list of I11 parameter sets as an array ofSatDbI11Data data objects. This array may be empty if there are no I11 parameter sets defined for this satellite.

Return Codes:

Code	Description
200	OK, returns a JSON document with the requested information
401	not logged in. In this case anApiError document is returned, describing details of the error.
404	satellite with the given *satelliteId* does not exists. In this case anApiError document is returned, describing details of the error.

GET /api/v1/satellites/{satelliteId}/i11params/{i11Id}

Returns the information stored for a particular I11 parameter set addressed by {satelliteId} / {i11Id} as a SatDbI11Data data object.

Parameters:

Parameter	Description
satelliteId	Id of the satellite.
i11Id	Id of the I11 parameter set.

Expected Payload: none

Replied Payload: The requested SatDbI11Data data object.

Return Codes:

Code	Description
200	OK, returns a document describing the requested satellite.
401	not logged in. In this case anApiError document is returned, describing details of the error.

Code	Description
404	no satellite I11 parameter set exists with the given *satelliteId* / *i11Id*. In this case an ApiError document is returned, describing details of the error.

POST /api/v1/satellites/{satelliteId}/i11params

Creates a new I11 parameter set with the given attributes. {satelliteId} specifies the satellite for which the I11 parameter set shall be created. The unique *i11Id* is generated by the database backend and returned in the response document.

Parameters:

Parameter	Description
satelliteId	Id of the satellite.

Expected Payload: The SatDbI11Data data object to store. Any *i11Id* definition in the source is ignored as this will be set by the database when the new entry is created. The same applies to *satellite_Id*, the newly created SatDbI11Data data object will be linked to the satellite specified in the URL. If other keys of the data object are missing, default values will be assigned these keys.

Replied Payload: The SatDbI11Data data object added to the datbase. The replied data object contains the *i11Id* assigned to the newly created record.

Return Codes:

Code	Description
201	CREATED, the created SatDbI11Data data object is replied as payload
401	not logged in. In this case an ApiError document is returned,describing details of the error.
500	DB is not writable. In this case an ApiError document is returned,describing details of the error.

PATCH /api/v1/satellites/{satelliteId}/i11params/{i11Id}

Updates an existing I11 parameter set with the given *satelliteId* / *i11Id* from the attached SatDbI11Data data object.

Parameters:

Parameter	Description
satelliteId	Id of the satellite.

Parameter	Description
i11Id	Id of the I11 parameter set.

Expected Payload: The SatDbI11Data data object containing the fields to be updated in the database. If the data object contains values for *i11Id* and / or *satelliteId*, these values are ignored. *i11Id* and *satelliteId* are immutable once a record has been created. Keys not contained in the supplied data object are retained unchanged.

Replied Payload: The SatDbI11Data data object read from the datbase after the requested changes have been applied.

Return Codes:

Code	Description
200	OK, changes have been applied.
401	not logged in. In this case anApiError document is returned, describing details of the error.
404	no satellite I11 parameter set exists with the given*satelliteId* / *i11Id*. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

DELETE /api/v1/satellites/{satelliteId}/i11params/{i11Id}

Remove a I11 parameter set from the database.

Parameters:

Parameter	Description
satelliteId	Id of the satellite the I11 parameter set belongs to.
i11Id	Id of the I11 parameter set to delete.

Expected Payload: none

Replied Payload: none

Return Codes:

Code	Description

Code	Description
204	NO CONTENT, the I11 parameter set has been deleted.
401	not logged in. In this case an ApiError document is returned,describing details of the error.
404	no satellite I11 parameter set exists with the given *satelliteId* / *i11Id*. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case an ApiError document is returned,describing details of the error.

DELETE /api/v1/satellites/{satelliteId}/i11params

Remove all I11 parameter sets which belong to the given *satelliteId*.

Parameters:

Parameter	Description
satelliteId	Id of the satellite

Return Codes:

Code	Description
204	NO CONTENT, all I11 parameter sets for the selected satellite have been deleted.
401	not logged in. In this case an ApiError document is returned,describing details of the error.
404	satellite with the given *satelliteId* does not exist. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case an ApiError document is returned,describing details of the error.

GET /api/v1/satellites/{satelliteId}/tleparams

Returns the list of TLE parameter sets for a given satellite as an array ofSatDbTLEData data objects.

Parameters:

Parameter	Description

Parameter	Description
satelliteId	Id of the satellite.

Expected Payload: none

Replied Payload: The list of TLE parameter sets as an array ofSatDbTLEData data objects. This array may be empty if there are no TLE parameter sets defined for this satellite. As there may be actually only one TLE parameter set for each satellite, the array may contain only zero or one elements.

Return Codes:

Code	Description
200	OK, returns a JSON document with the requested information
401	not logged in. In this case anApiError document is returned, describing details of the error.
404	satellite with the given *satelliteId* does not exists. In this case anApiError document is returned, describing details of the error.

GET /api/v1/satellites/{satelliteId}/tleparams/{noradNumber}

Returns the information stored for a particular TLE parameter set addressed by {satelliteId} / {noradNumber} as a SatDbTLEData data object.

Parameters:

Parameter	Description
satelliteId	Id of the satellite.
noradNumber	Id / norad number of the TLE parameter set.

Expected Payload: none

Replied Payload: The requested SatDbTLEData data object.

Return Codes:

Code	Description
200	OK, returns a document describing the requested satellite.

Code	Description
401	not logged in. In this case an ApiError document is returned, describing details of the error.
404	no satellite TLE parameter set exists with the given *satelliteId / noradNumber*. In this case an ApiError document is returned, describing details of the error.

POST /api/v1/satellites/{satelliteId}/tleparams

Creates a new TLE parameter set with the given attributes. {satelliteId} specifies the satellite for which the TLE parameter set shall be created. The unique *noradNumber* is taken from the *noradNumber* field in the satellite's master data and returned in the response document. Hence it is important that the *noradNumber* field in the satellite's master data has been set before to the correct value.

As there may be only TLE parameter set assigned to a satellite, this POST method behaves in a special way: If there already exists a TLE parameter set for this satellite, it will be silently overwritten by the POST call. The call proceeds, stores the new data and returns a 201 return code. This behavior ensures, that not more than one TLE parameter set can be assigned to a satellite.

Parameters:

Parameter	Description
satelliteId	Id of the satellite.

Expected Payload: The SatDbTLEData data object to store. Any *noradNumber* definition in the source is ignored as this will be taken from the *noradNumber* field in the satellite's master data when the new entry is created. The same applies to *satellite_Id*, the newly created SatDbTLEData data object will be linked to the satellite specified in the URL. If other keys of the data object are missing, default values will be assigned these keys.

Replied Payload: The SatDbTLEData data object added to the datbase. The replied data object contains the *noradNumber* assigned to the newly created record.

Return Codes:

Code	Description
201	CREATED, the created SatDbTLEData data object is replied as payload
401	not logged in. In this case an ApiError document is returned,describing details of the error.
500	DB is not writable. In this case an ApiError document is returned,describing details of the error.

PATCH /api/v1/satellites/{satelliteId}/tleparams/{noradNumber}

Updates an existing TLE parameter set with the given*satelliteId* / *noradNumber* from the attached SatDbTLEData data object.

Parameters:

Parameter	Description
satelliteId	Id of the satellite.
noradNumber	Id of the TLE parameter set.

Expected Payload: The SatDbTLEData data object containing the fields to be updated in the database. If the data object contains values for *noradNumber* and / or *satelliteId*, these values are ignored. *noradNumber* and *satelliteId* are immutable once a record has been created. Keys not contained in the supplied data object are retained unchanged.

Replied Payload: The SatDbTLEData data object read from the datbase after the requested changes have been applied.

Return Codes:

Code	Description
200	OK, changes have been applied.
401	not logged in. In this case anApiError document is returned, describing details of the error.
404	no satellite TLE parameter set exists with the given*satelliteId* / *noradNumber*. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

DELETE /api/v1/satellites/{satelliteId}/tleparams/{noradNumber}

Remove a TLE parameter set from the database.

Parameters:

Parameter	Description
satelliteId	Id of the satellite the TLE parameter set belongs to.
noradNumber	Id of the TLE parameter set to delete.

Expected Payload: none

Replied Payload: none

Return Codes:

Code	Description
204	NO CONTENT, the TLE parameter set has been deleted.
401	not logged in. In this case an ApiError document is returned,describing details of the error.
404	no satellite TLE parameter set exists with the given *satelliteId / noradNumber*. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case an ApiError document is returned,describing details of the error.

DELETE /api/v1/satellites/{satelliteId}/tleparams

Remove all TLE parameter sets which belong to the given *satelliteId*.

Parameters:

Parameter	Description
satelliteId	Id of the satellite

Return Codes:

Code	Description
204	NO CONTENT, all TLE parameter sets for the selected satellite have been deleted.
401	not logged in. In this case an ApiError document is returned,describing details of the error.
404	satellite with the given *satelliteId* does not exist. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case an ApiError document is returned,describing details of the error.

DELETE /api/v1/satellites/{satelliteId}/ttracks

In general, the table track ('ttrack') API calls are located at /api/v1/antennas. The only call located at /api/v1/satellites is to temove all table track data sets which belong to the given *satelliteId* (for all antennas).

Parameters:

Parameter	Description
satelliteId	Id of the satellite

Return Codes:

Code	Description
204	NO CONTENT, all table track data sets for the selected satellite have been deleted.
401	not logged in. In this case anApiError document is returned,describing details of the error.
404	satellite with the given *satelliteId* does not exist. In this case anApiError document is returned, describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

1.2.27 /api/v1/satoperators

This API endpoint gives access to various functions of the sat-nms satoperator database table, permitting to read, add, change or delete entries at the database

Supported HTTP methods: GET, POST, PATCH, DELETE, HEAD, OPTIONS

The API generally does not permit PATCH or DELETE operations on entire tables, this to protect the database from being accidentally be deleted. Even POST operations only work with single data objects.

GET /api/v1/satoperators

Returns the list of stored satoperators as an array ofSatDbSatOperator data objects.

Parameters: none

Expected Payload: none

Replied Payload: The list of stored satoperators as an array ofSatDbSatOperator data objects.

Return Codes:

Code	Description
200	OK, returns a JSON document with all satoperators and their attributes.

Code	Description
401	not logged in. In this case an ApiError document is returned,describing details of the error.

GET /api/v1/satoperators/{operatorId}

Returns the basic properties of the satoperator addressed by {operatorId} as a SatDbSatOperator data object.

Parameters:

Parameter	Description
operatorId	Id of the satoperator.

Expected Payload: none

Replied Payload: The requested SatDbSatOperator data object.

Return Codes:

Code	Description
200	OK, returns a document describing the requested satoperator.
401	not logged in. In this case an ApiError document is returned, describing details of the error.
404	satoperator with the given *operatorId* does not exists. In this case an ApiError document is returned, describing details of the error.

POST /api/v1/satoperators

Creates a new satoperator with the given attributes. The unique *operatorId* is generated by the database backend and returned in the response document.

Parameters: none

Expected Payload: The SatDbSatOperator data object to store. Any *operatorId* definition in the source is ignored as this will be set by the database when the new entry is created. If other keys of the data object are missing, default values will be assigned these keys.

Replied Payload: The SatDbSatOperator data object added to the datbase. The replied data object contains the *operatorId* assigned to the newly created record.

Return Codes:

Code	Description
201	CREATED, the created SatDbSatOperator data object is replied as payload
401	not logged in. In this case an ApiError document is returned,describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

PATCH /api/v1/satoperators/{operatorId}

Updates an existing satellite operator with the given *operatorId* from the attached SatDbSatOperator data object.

Parameters:

Parameter	Description
operatorId	Id of the satoperator.

Expected Payload: The SatDbSatOperator data object containing the fields to be updated in the database. If the data object contains an *operatorId* key, the value is ignored. The database record is solely identified by *operatorId* set in the URL, the *operatorId* of the record is immutable. Keys not contained in the supplied data object are retained unchanged.

Replied Payload: The SatDbSatOperator data object read from the datbase after the requested changes have been applied.

Return Codes:

Code	Description
200	OK, changes have been applied.
401	not logged in. In this case an ApiError document is returned, describing details of the error.
404	the satellite operator with the given *operatorId* does not exists. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

DELETE /api/v1/satoperators/{operatorId}

Remove a satoperator from the database. This deletes the satoperator's record from the satoperator table and removes all references to the deleted satoperator in the *satellites* table.

Parameters:

Parameter	Description
operatorId	Id of the satoperator to delete

Expected Payload: none

Replied Payload: none

Return Codes:

Code	Description
204	NO CONTENT, the satellite has been deleted.
401	not logged in. In this case anApiError document is returned, describing details of the error.
404	the satellite operator with the given *operatorId* does not exists. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

1.2.28 /api/v1/channels

This API endpoint gives access to various functions of the*channels* database table, permitting to read, add, change or delete entries at the database. This database table holds all satellite channels known in the satnms database.

Supported HTTP methods: GET, POST, PATCH, DELETE, HEAD, OPTIONS

The API generally does not permit PATCH or DELETE operations on entire tables, this to protect the database from being accidentally be deleted. Even POST operations only work with single data objects.

GET /api/v1/channels/

Returns the list of all satellite channel records as an array ofChannelData data objects.

Parameters: none

Expected Payload: none

Replied Payload: The list of satellite channel records as an array ofChannelData data objects.

Return Codes:

Code	Description
200	OK, returns a JSON document with the requested information
401	not logged in. In this case an ApiError document is returned, describing details of the error.

GET /api/v1/channels/ip

Returns the list of all 'IP' type satellite channel records as an array ofChannelData data objects.

Parameters: none

Expected Payload: none

Replied Payload: The list of satellite channel records as an array ofChannelData data objects. This array may be empty if there are no such channel records in the database..

Return Codes:

Code	Description
200	OK, returns a JSON document with the requested information
401	not logged in. In this case an ApiError document is returned, describing details of the error.

GET /api/v1/channels/asi

Returns the list of all 'ASI' type satellite channel records as an array ofChannelData data objects.

Parameters: none

Expected Payload: none

Replied Payload: The list of satellite channel records as an array ofChannelData data objects. This array may be empty if there are no such channel records in the database.

Return Codes:

Code	Description
200	OK, returns a JSON document with the requested information
401	not logged in. In this case an ApiError document is returned, describing details of the error.

GET /api/v1/channels/sat

Returns the list of all 'SAT' type satellite channel records as an array ofChannelData data objects.

Parameters: none

Expected Payload: none

Replied Payload: The list of satellite channel records as an array ofChannelData data objects. This array may be empty if there are no such channel records in the database.

Return Codes:

Code	Description
200	OK, returns a JSON document with the requested information
401	not logged in. In this case anApiError document is returned, describing details of the error.

GET /api/v1/channels/sat/{satelliteId}

Returns the list of 'SAT' type satellite channel records for a given satellite as an array of ChannelData data objects.

Parameters:

Parameter	Description
satelliteId	Id of the satellite.

Expected Payload: none

Replied Payload: The list of satellite channel records as an array ofChannelData data objects. This array may be empty if there are no channel records defined for this satellite.

Return Codes:

Code	Description
200	OK, returns a JSON document with the requested information
401	not logged in. In this case anApiError document is returned, describing details of the error.
404	satellite with the given *satelliteId* does not exists. In this case anApiError document is returned, describing details of the error.

GET /api/v1/channels/{channelId}

Returns the information stored for a particular channel record addressed by {channelId} as a ChannelData data object.

Parameters:

Parameter	Description
channelId	Id of the channel record.

Expected Payload: none

Replied Payload: The requested ChannelData data object.

Return Codes:

Code	Description
200	OK, returns a document describing the requested satellite.
401	not logged in. In this case an ApiError document is returned, describing details of the error.
404	no satellite channel record exists with the given *channelId*. In this case an ApiError document is returned, describing details of the error.

POST /api/v1/channels

Creates a new channel record with the given attributes.

Parameters: none

Expected Payload: The ChannelData data object to store. The channel definition must conform the following conditions:

- The *decoderInput* field must be set to one of 'SAT', 'IP' or 'ASI' to define the type of channel.
- The *name* field must be set to a value which is unique for the given satellite or the API call will decline to create the database record.
- 'SAT' type records must contain a valid *satelliteId*
- Any *channelId* definition in the source is ignored as this will be set by the database when the new entry is created.

Replied Payload: The ChannelData data object added to the datbase. The replied data object contains the *channelId* assigned to the newly created record.

Return Codes:

Code	Description

Code	Description
201	CREATED, the created ChannelData data object is replied as payload
400	BAD_REQUEST if the supplied ChannelData does not conform the conditions listed above.
401	not logged in. In this case an ApiError document is returned,describing details of the error.
500	DB is not writable. In this case an ApiError document is returned,describing details of the error.

PATCH /api/v1/channels/{channelId}

Updates an existing channel record with the given *channelId* from the attached ChannelData data object.

Parameters:

Parameter	Description
channelId	Id of the channel record.

Expected Payload: The ChannelData data object containing the fields to be updated in the database. Fields not contained in the supplied data object are retained unchanged. The channel definition must conform the following conditions:

- If it changes the *name* of the record, this must be set to a value which is unique for the given satellite or the API call will decline to create the database record.
- 'SAT' type records must contain a valid *satelliteId*
- Any *channelId* definition in the source is ignored as the *channelId* is immutable once a record has been created.

Replied Payload: The ChannelData data object read from the database after the requested changes have been applied.

Return Codes:

Code	Description
200	OK, changes have been applied.
400	BAD_REQUEST if the supplied ChannelData does not conform the conditions listed above.
401	not logged in. In this case an ApiError document is returned, describing details of the error.

Code	Description
404	no satellite channel record exists with the given *channelId*. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

DELETE /api/v1/channels/{channelId}

Remove a channel record from the database.

Parameters:

Parameter	Description
channelId	Id of the channel record to delete.

Expected Payload: none

Replied Payload: none

Return Codes:

Code	Description
204	NO CONTENT, the channel record has been deleted.
401	not logged in. In this case an ApiError document is returned, describing details of the error.
404	no satellite channel record exists with the given *channelId*. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

DELETE /api/v1/channels/sat/{satelliteId}

Remove all channel records which belong to the given *satelliteId*.

Parameters:

Parameter	Description
satelliteId	Id of the satellite

Return Codes:

Code	Description
204	NO CONTENT, all channel records for the selected satellite have been deleted.
401	not logged in. In this case an ApiError document is returned, describing details of the error.
404	satellite with the given *satelliteId* does not exist. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case an ApiError document is returned,describing details of the error.

1.2.29 /api/v1/streamkeys

This API endpoint gives access to various functions of the*streamkeys* database table, permitting to read, add, change or delete entries at the database. This database table holds all BISS-1/BISS-E keys and SRT passphrases known in the sat-nms database.

Supported HTTP methods: GET, POST, PATCH, DELETE, HEAD, OPTIONS

The API generally does not permit PATCH or DELETE operations on entire tables, this to protect the database from being accidentally be deleted. Even POST operations only work with single data objects.

GET /api/v1/streamkeys

Returns the list of stored stream keys as an array ofStreamKeyData data objects.

Parameters: none

Expected Payload: none

Replied Payload: The list of stored stream keys as an array ofStreamKeyData data objects.

Return Codes:

Code	Description
200	OK, returns a JSON document with all streamkeys and their attributes.
401	not logged in. In this case an ApiError document is returned,describing details of the error.
500	DB is not writable. In this case an ApiError document is returned,describing details of the error.

GET /api/v1/streamkeys/{keyId}

Returns the basic properties of the stream key addressed by {keyId} as aStreamKeyData data object.

Parameters:

Parameter	Description
keyId	Id of the stream key.

Expected Payload: none

Replied Payload: The requested StreamKeyData data object.

Return Codes:

Code	Description
200	OK, returns a document describing the requested stream key.
401	not logged in. In this case anApiError document is returned, describing details of the error.
404	stream key with the given *keyId* does not exists. In this case anApiError document is returned, describing details of the error.

POST /api/v1/streamkeys

Creates a new stream key with the given attributes. The unique*keyId* is generated by the database backend and returned in the response document.

Parameters: none

Expected Payload: The StreamKeyData data object to store. Any *keyId* definition in the source is ignored as this will be set by the database when the new entry is created. The *name* field in the supplied payload must be unique over the entire table or the API call will decline to create the database record.

Replied Payload: The StreamKeyData data object added to the database. The replied data object contains the *keyId* assigned to the newly created record.

Return Codes:

Code	Description
201	CREATED, the created StreamKeyData data object is replied as payload
400	BAD_REQUEST. Either the *name* supplied with the request is not unique or the*key* does not conform to the format restrictions for the given *keyType*

Code	Description
401	not logged in. In this case an ApiError document is returned,describing details of the error.
500	DB is not writable. In this case an ApiError document is returned,describing details of the error.

PATCH /api/v1/streamkeys/{keyId}

Updates an existing stream key with the given *keyId* from the attached StreamKeyData data object.

Parameters:

Parameter	Description
keyId	Id of the stream key.

Expected Payload: The StreamKeyData data object containing the fields to be updated in the database. If the data object contains an *keyId* key, the value is ignored. The database record is solely identified by *keyId* set in the URL, the *keyId* of the record is immutable. Keys not contained in the supplied data object are retained unchanged. The *name* field in the supplied payload must be unique over the entire table or the API call will decline to modify the database record.

Replied Payload: The StreamKeyData data object read from the database after the requested changes have been applied.

Return Codes:

Code	Description
200	OK, changes have been applied.
400	BAD_REQUEST. Either the *name* supplied with the request is not unique or the *key* does not conform to the format restrictions for the given *keyType*
401	not logged in. In this case an ApiError document is returned, describing details of the error.
404	the stream key with the given *keyId* does not exists. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case an ApiError document is returned,describing details of the error.

DELETE /api/v1/streamkeys/{keyId}

Remove a stream key from the database.

Parameters:

Parameter	Description
keyId	Id of the stream key to delete

Expected Payload: none

Replied Payload: none

Return Codes:

Code	Description
204	NO CONTENT, the satellite has been deleted.
401	not logged in. In this case anApiError document is returned, describing details of the error.
404	the stream key with the given *keyId* does not exists. In this case anApiError document is returned, describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

1.2.30 /api/v1/antennas

This API endpoint gives access to various functions of the sat-nms antenna database table, permitting to read, add, change or delete entries at the database

Supported HTTP methods: GET, POST, PATCH, DELETE, HEAD, OPTIONS

The API generally does not permit PATCH or DELETE operations on entire tables, this to protect the database from being accidentally be deleted. Even POST operations only work with single data objects.

GET /api/v1/antennas

Returns the list of stored antennas as an array ofSatDbAntenna data objects.

Parameters: none

Expected Payload: none

Replied Payload: The list of stored antennas as an array ofSatDbAntenna data objects.

Return Codes:

Code	Description
200	OK, returns a JSON document with all antennas and their attributes.
401	not logged in. In this case an ApiError document is returned,describing details of the error.

GET /api/v1/antennas/{antennaId}

Returns the basic properties of the antenna addressed by {antennaId} as aSatDbAntenna data object.

Parameters:

Parameter	Description
antennaId	Id of the antenna.

Expected Payload: none

Replied Payload: The requested SatDbAntenna data object.

Return Codes:

Code	Description
200	OK, returns a document describing the requested antenna.
401	not logged in. In this case an ApiError document is returned, describing details of the error.
404	antenna with the given *antennaId* does not exists. In this case an ApiError document is returned, describing details of the error.

POST /api/v1/antennas

Creates a new antenna with the given attributes. The unique *antennaId* is generated by the database backend and returned in the response document.

Parameters: none

Expected Payload: The SatDbAntenna data object to store. Any *antennaId* definition in the source is ignored as this will be set by the database when the new entry is created. If other keys of the data object are missing, default values will be assigned these keys.

Replied Payload: The SatDbAntenna data object added to the database. The replied data object contains the *antennaId* assigned to the newly created record.

Return Codes:

Code	Description
201	CREATED, the created SatDbAntenna data object is replied as payload
401	not logged in. In this case an ApiError document is returned,describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

PATCH /api/v1/antennas/{antennaId}

Updates an existing antenna with the given *antennaId* from the attached SatDbAntenna data object.

Parameters:

Parameter	Description
antennaId	Id of the antenna.

Expected Payload: The SatDbAntenna data object containing the fields to be updated in the database. If the data object contains an *antennaId* key, the value is ignored. The database record is solely identified by *antennaId* set in the URL, the *antennaId* of the record is immutable. Keys not contained in the supplied data object are retained unchanged.

Replied Payload: The SatDbAntenna data object read from the database after the requested changes have been applied.

Return Codes:

Code	Description
200	OK, changes have been applied.
401	not logged in. In this case an ApiError document is returned, describing details of the error.
404	antenna with the given *antennaId* does not exists. In this case anApiError document is returned, describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

DELETE /api/v1/antennas/{antennaId}

Remove a antenna from the database. This deletes the antenna's record from the antenna table and all records from the *satpositions* table which refer to the deleted antenna.

Parameters:

Parameter	Description
antennaId	Id of the antenna to delete

Expected Payload: none

Replied Payload: none

Return Codes:

Code	Description
204	NO CONTENT, the satellite has been deleted.
401	not logged in. In this case anApiError document is returned, describing details of the error.
404	antenna with the given *antennaId* does not exists. In this case anApiError document is returned, describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

GET /api/v1/antennas/{antennaId}/positions

Returns the list of satellite positions for a given antenna as an array ofSatDbPosition data objects.

Parameters:

Parameter	Description
antennaId	Id of the antenna.

Expected Payload: none

Replied Payload: The list of satellite positions as an array ofSatDbPosition data objects. This array may be empty if there are no satellite positions defined for this antenna.

Return Codes:

Code	Description
200	OK, returns a JSON document with the requested information

Code	Description
401	not logged in. In this case an ApiError document is returned, describing details of the error.
404	the antenna with the given *antennaId* does not exists. In this case an ApiError document is returned, describing details of the error.

GET /api/v1/antennas/{antennaId}/positions/{positionId}

Returns the information stored for a particular satellite position addressed by {antennaId} / {positionId} as a SatDbPosition data object.

Parameters:

Parameter	Description
antennaId	Id of the antenna.
positionId	Id of the satellite position.

Expected Payload: none

Replied Payload: The requested SatDbPosition data object.

Return Codes:

Code	Description
200	OK, returns a document describing the requested antenna.
401	not logged in. In this case an ApiError document is returned, describing details of the error.
404	no antenna satellite position exists with the given *antennaId / positionId*. In this case an ApiError document is returned, describing details of the error.

ST /api/v1/antennas/{antennaId}/positions

Creates a new satellite position with the given attributes. {antennaId} specifies the antenna for which the satellite position shall be created. The unique *positionId* is generated by the database backend and returned in the response document.

Parameters:

Parameter	Description

Parameter	Description
antennaId	Id of the antenna.

Expected Payload: The SatDbPosition data object to store. Any *positionId* definition in the source is ignored as this will be set by the database when the new entry is created. The same applies to *antennaId*, the newly created SatDbPosition data object will be linked to the antenna specified in the URL. If other keys of the data object are missing, default values will be assigned these keys.

Replied Payload: The SatDbPosition data object added to the database. The replied data object contains the *positionId* assigned to the newly created record.

Return Codes:

Code	Description
201	CREATED, the created SatDbPosition data object is replied as payload
401	not logged in. In this case an ApiError document is returned,describing details of the error.

PATCH /api/v1/antennas/{antennaId}/positions/{positionId}

Updates an existing satellite position with the given *antennaId* / *positionId* from the attached SatDbPosition data object.

Parameters:

Parameter	Description
antennaId	Id of the antenna.
positionId	Id of the satellite position.

Expected Payload: The SatDbPosition data object containing the fields to be updated in the database. If the data object contains values for *positionId* and / or *antennaId*, these values are ignored. *positionId* and *antennaId* are immutable once a record has been created. Keys not contained in the supplied data object are retained unchanged.

Replied Payload: The SatDbPosition data object read from the database after the requested changes have been applied.

Return Codes:

Code	Description

Code	Description
200	OK, changes have been applied.
401	not logged in. In this case an [ApiError](#) document is returned, describing details of the error.
404	no antenna satellite position exists with the given *antennaId* / *positionId*. In this case an [ApiError](#) document is returned, describing details of the error.
500	DB is not writable. In this case an [ApiError](#) document is returned,describing details of the error.

DELETE /api/v1/antennas/{antennaId}/positions/{positionId}

Remove a satellite position from the database.

Parameters:

Parameter	Description
antennaId	Id of the antenna the satellite position belongs to.
positionId	Id of the satellite position to delete.

Expected Payload: none

Replied Payload: none

Return Codes:

Code	Description
204	NO CONTENT, the satellite position has been deleted.
401	not logged in. In this case an [ApiError](#) document is returned,describing details of the error.
404	no antenna satellite position exists with the given *antennaId* / *positionId*. In this case an [ApiError](#) document is returned, describing details of the error.
500	DB is not writable. In this case an [ApiError](#) document is returned,describing details of the error.

DELETE /api/v1/antennas/{antennaId}/positions

Remove all satellite positions which belong to the given *antennaId*.

Parameters:

Parameter	Description
antennaId	Id of the antenna

Return Codes:

Code	Description
204	NO CONTENT, all satellite positions for the selected antenna have been deleted.
401	not logged in. In this case anApiError document is returned,describing details of the error.
404	antenna with the given *antennaId* does not exist. In this case anApiError document is returned, describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

GET /api/v1/antennas/{antennaId}/ttracks

Returns the list of table track data sets for a given antenna as an array ofSatDbTableTracking data objects.

Parameters:

Parameter	Description
antennaId	Id of the antenna.

Expected Payload: none

Replied Payload: The list of table track data sets as an array ofSatDbTableTracking data objects. This array may be empty if there are no table track data sets defined for this antenna.

Return Codes:

Code	Description
200	OK, returns a JSON document with the requested information
401	not logged in. In this case anApiError document is returned, describing details of the error.
404	antenna with the given *antennaId* does not exists. In this case anApiError document is returned, describing details of the error.

GET /api/v1/antennas/{antennaId}/{satelliteId}/ttracks

Returns the list of table track data sets for a given satellite/antenna combination as an array of SatDbTableTracking data objects.

Parameters:

Parameter	Description
antennaId	Id of the antenna.
satelliteId	Id of the satellite.

Expected Payload: none

Replied Payload: The list of table track data sets as an array ofSatDbTableTracking data objects. This array may be empty if there are no table track data sets defined for this satellite.

Return Codes:

Code	Description
200	OK, returns a JSON document with the requested information
401	not logged in. In this case anApiError document is returned, describing details of the error.
404	satellite with the given *satelliteId* does not exists. In this case anApiError document is returned, describing details of the error.

GET /api/v1/antennas/{antennaId}/{satelliteId}/ttracks/{ttId}

Returns the information stored for a particular table track data set addressed by {antennaId} / {satelliteId} / {ttId} as a SatDbTableTracking data object.

Parameters:

Parameter	Description
antennaId	Id of the antenna.
satelliteId	Id of the satellite.
ttId	Id of the table track data set.

Expected Payload: none

Replied Payload: The requested SatDbTableTracking data object.

Return Codes:

Code	Description
200	OK, returns a document describing the requested satellite.
401	not logged in. In this case an ApiError document is returned, describing details of the error.
404	no satellite table track data set exists with the given *satelliteId* / *ttId*. In this case an ApiError document is returned, describing details of the error.

POST /api/v1/antennas/{antennaId}/{satelliteId}/ttracks

Creates a new table track data set with the given attributes. {antennaId} / {satelliteId} specifies the satellite / antenna combination for which the table track data set shall be created. The unique *ttId* is generated by the database backend and returned in the response document.

Parameters:

Parameter	Description
antennaId	Id of the antenna.
satelliteId	Id of the satellite.

Expected Payload: The SatDbTableTracking data object to store. Any *ttId* definition in the source is ignored as this will be set by the database when the new entry is created. The same applies to *satellite_Id*, the newly created SatDbTableTracking data object will be linked to the satellite specified in the URL. If other keys of the data object are missing, default values will be assigned these keys.

Replied Payload: The SatDbTableTracking data object added to the datbase. The replied data object contains the *ttId* assigned to the newly created record.

Return Codes:

Code	Description
201	CREATED, the created SatDbTableTracking data object is replied as payload
401	not logged in. In this case an ApiError document is returned,describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

PATCH /api/v1/antennas/{antennaId}/{satelliteId}/ttracks/{ttId}

Updates an existing table tracking data set with the given*antennaId* / *satelliteId* / *ttId* from the attached SatDbTableTracking data object.

Parameters:

Parameter	Description
antennaId	Id of the antenna.
satelliteId	Id of the satellite.
ttId	Id of the table track data set.

Expected Payload: The SatDbTableTracking data object containing the fields to be updated in the database. If the data object contains values for any of *ttId*, *satelliteId* or *antennaId*, these values are ignored. *ttId*, *satelliteId* and *antennaId* are immutable once a record has been created. Keys not contained in the supplied data object are retained unchanged.

Replied Payload: The SatDbTableTracking data object read from the datbase after the requested changes have been applied.

Return Codes:

Code	Description
200	OK, changes have been applied.
401	not logged in. In this case anApiError document is returned, describing details of the error.
404	no satellite table track data set exists with the given*satelliteId* / *ttId*. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

DELETE /api/v1/antennas/{antennaId}/{satelliteId}/ttracks/{ttId}

Remove a table track data set from the database.

Parameters:

Parameter	Description
antennaId	Id of the antenna.
satelliteId	Id of the satellite the table track data set belongs to.

Parameter	Description
ttId	Id of the table track data set to delete.

Expected Payload: none

Replied Payload: none

Return Codes:

Code	Description
204	NO CONTENT, the table track data set has been deleted.
401	not logged in. In this case an ApiError document is returned,describing details of the error.
404	no satellite table track data set exists with the given *satelliteId* / *ttId*. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case an ApiError document is returned,describing details of the error.

DELETE /api/v1/antennas/{antennaId}/{satelliteId}/ttracks

Remove all table track data sets which belong to the given *satelliteId* / *antennaId* combination.

Parameters:

Parameter	Description
satelliteId	Id of the satellite
antennaId	Id of the antenna.

Return Codes:

Code	Description
204	NO CONTENT, all table track data sets for the selected satellite have been deleted.
401	not logged in. In this case an ApiError document is returned,describing details of the error.
404	satellite with the given *satelliteId* does not exist. In this case an ApiError document is returned, describing details of the error.

Code	Description
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

DELETE /api/v1/antennas/{antennaId}/ttracks

Remove all table track data sets which belong to the given antenna.

Parameters:

Parameter	Description
antennaId	Id of the antenna

Return Codes:

Code	Description
204	NO CONTENT, all table track data sets for the selected satellite have been deleted.
401	not logged in. In this case anApiError document is returned,describing details of the error.
404	satellite with the given *satelliteId* does not exist. In this case anApiError document is returned, describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

GET /api/v1/antennas/{antennaId}/beacon-atten/{beaconId}

Returns the attenuation value addressed by {antennaId} / {beaconId} as a SatDbBeaconAttenuation data object.

Parameters:

Parameter	Description
antennaId	Id of the antenna.
beaconId	Id of the beacon.

Expected Payload: none

Replied Payload: The requested SatDbBeaconAttenuation data object. If no attenuation has been stored for this antennaId/beaconId combination, an empty SatDbBeaconAttenuation data object with the attenuation value set to 0.0 is returned.

Return Codes:

Code	Description
200	OK, returns a document describing the requested satellite.
400	BAD_REQUEST, antennaId or beaconId could not be parsed to a numeric identifier. In this case an ApiError document is returned, describing details of the error.
401	not logged in. In this case an ApiError document is returned, describing details of the error.

PUT /api/v1/antennas/{antennaId}/beacon-atten/{beaconId}

Sets the attenuation value for an {antennaId} / {beaconId} combination. Returns the attenuation set as a SatDbBeaconAttenuation data object. If no attenuation record exists for this antenna / beacon combination a new one is created. If there is already an attenuation record for this combination, the existing one will be overwritten.

Parameters:

Parameter	Description
antennaId	Id of the antenna.
beaconId	Id of the beacon.

Expected Payload: The attenuation to set as a SatDbBeaconAttenuation data object. Only the attenuation is read from this data object, any IDs passed to this API function are ignored. The IDs are defined solely by the URL parameters.

Replied Payload: The SatDbBeaconAttenuation data object with the updated attenuation .

Return Codes:

Code	Description
200	OK, returns a document describing the requested satellite.
400	BAD_REQUEST, antennaId or beaconId could not be parsed to a numeric identifier. In this case an ApiError document is returned, describing details of the error.
401	not logged in. In this case an ApiError document is returned, describing details of the error.

Code	Description
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

PATCH /api/v1/antennas/{antennaId}/beacon-atten/{beaconId}

Sets the attenuation value for an {antennaId} / {beaconId} combination. Returns the attenuation set as a SatDbBeaconAttenuation data object.

Parameters:

Parameter	Description
antennaId	Id of the antenna.
beaconId	Id of the beacon.

Expected Payload: The attenuation to set as a SatDbBeaconAttenuation data object. Only the attenuation is read from this data object, any IDs passed to this API function are ignored. The IDs are defined solely by the URL parameters.

Replied Payload: The SatDbBeaconAttenuation data object with the updated attenuation .

Return Codes:

Code	Description
200	OK, returns a document describing the requested satellite.
400	BAD_REQUEST, antennaId or beaconId could not be parsed to a numeric identifier. In this case an ApiError document is returned, describing details of the error.
401	not logged in. In this case an ApiError document is returned, describing details of the error.
404	no attenuation record exists with the given *satelliteId* / *beaconId*. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

DELETE /api/v1/antennas/{antennaId}/beacon-atten/{beaconId}

Deletes the attenuation value addressed by {antennaId} / {beaconId}.

Parameters:

Parameter	Description
antennaId	Id of the antenna.
beaconId	Id of the beacon.

Expected Payload: none

Replied Payload: none.

Return Codes:

Code	Description
204	OK, the record has been deleted
400	BAD_REQUEST, antennaId or beaconId could not be parsed to a numeric identifier. In this case an ApiError document is returned, describing details of the error.
401	not logged in. In this case an ApiError document is returned, describing details of the error.
404	no attenuation record exists with the given *satelliteId* / *beaconId*. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case an ApiError document is returned,describing details of the error.

GET /api/v1/antennas/{antennaId}/tc-atten/{tcId}

Returns the attenuation values addressed by {antennaId} / {tcId} as a SatDbTcAttenuation data object. If no attenuation has been stored for this antennaId/tcId combination, an empty SatDbTcAttenuation data object with the attenuation values set to 0.0 is returned.

Parameters:

Parameter	Description
antennaId	Id of the antenna.
tcId	Id of the tc channel.

Expected Payload: none

Replied Payload: The requested SatDbTcAttenuation data object.

Return Codes:

Code	Description
200	OK, returns a document describing the requested satellite.
400	BAD_REQUEST, antennaId or tcId could not be parsed to a numeric identifier. In this case an ApiError document is returned, describing details of the error.
401	not logged in. In this case an ApiError document is returned, describing details of the error.

PUT /api/v1/antennas/{antennaId}/tc-atten/{tcId}

Sets the attenuation values for an {antennaId} / {tcId} combination. Returns the attenuation set as a SatDbTcAttenuation data object. If no attenuation record exists for this antenna / tc combination a new one is created. If there is already an attenuation record for this combination, the existing one will be overwritten.

Parameters:

Parameter	Description
antennaId	Id of the antenna.
tcId	Id of the tc channel.

Expected Payload: The attenuation to set as a SatDbTcAttenuation data object. Only the attenuation values are read from this data object, any IDs passed to this API function are ignored. The IDs are defined solely by the URL parameters.

Replied Payload: The SatDbTcAttenuation data object with the updated attenuation .

Return Codes:

Code	Description
200	OK, returns a document describing the requested satellite.
400	BAD_REQUEST, antennaId or tcId could not be parsed to a numeric identifier. In this case an ApiError document is returned, describing details of the error.
401	not logged in. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case an ApiError document is returned,describing details of the error.

PATCH /api/v1/antennas/{antennaId}/tc-atten/{tcId}

Modifies the attenuation values for an {antennaId} / {tcId} combination. Returns the attenuation set as a SatDbTcAttenuation data object.

Parameters:

Parameter	Description
antennaId	Id of the antenna.
tcId	Id of the tc channel.

Expected Payload: One or both attenuation values to set in aSatDbTcAttenuation data object. Only the attenuation values are read from this data object, any IDs passed to this API function are ignored. The IDs are defined solely by the URL parameters.

Replied Payload: The SatDbTcAttenuation data object with the updated attenuation.

Return Codes:

Code	Description
200	OK, returns a document describing the requested satellite.
400	BAD_REQUEST, antennaId or tcId could not be parsed to a numeric identifier. In this case an ApiError document is returned, describing details of the error.
401	not logged in. In this case anApiError document is returned, describing details of the error.
404	no attenuation record exists with the given *satelliteId* / *tcId*. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

DELETE /api/v1/antennas/{antennaId}/tc-atten/{tcId}

Deletes the attenuation values addressed by {antennaId} / {tcId}.

Parameters:

Parameter	Description
antennaId	Id of the antenna.
tcId	Id of the tc channel.

Expected Payload: none

Replied Payload: none.

Return Codes:

Code	Description
204	OK, the record has been deleted
400	BAD_REQUEST, antennaId or tcId could not be parsed to a numeric identifier. In this case an ApiError document is returned, describing details of the error.
401	not logged in. In this case an ApiError document is returned, describing details of the error.
404	no attenuation record exists with the given *satelliteId / tcId*. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case an ApiError document is returned,describing details of the error.

GET /api/v1/antennas/{antennaId}/tm-atten/{tmId}

Returns the attenuation values addressed by {antennaId} / {tmId} as a SatDbTmAttenuation data object.

Parameters:

Parameter	Description
antennaId	Id of the antenna.
tmId	Id of the tm channel.

Expected Payload: none

Replied Payload: The requested SatDbTmAttenuation data object. If no attenuation has been stored for this antennaId/tcId combination, an empty SatDbTmAttenuation data object with the attenuation values set to 0.0 is returned.

Return Codes:

Code	Description
200	OK, returns a document describing the requested satellite.
400	BAD_REQUEST, antennaId or tmId could not be parsed to a numeric identifier. In this case an ApiError document is returned, describing details of the error.

Code	Description
401	not logged in. In this case an ApiError document is returned, describing details of the error.

PUT /api/v1/antennas/{antennaId}/tm-atten/{tmId}

Sets the attenuation values for an {antennaId} / {tmId} combination. Returns the attenuation values set as a SatDbTmAttenuation data object. If no attenuation record exists for this antenna / tm combination a new one is created. If there is already an attenuation record for this combination, the existing one will be overwritten.

Parameters:

Parameter	Description
antennaId	Id of the antenna.
tmId	Id of the tm channel.

Expected Payload: The attenuation to set as a SatDbTmAttenuation data object. Only the attenuation values are read from this data object, any IDs passed to this API function are ignored. The IDs are defined solely by the URL parameters.

Replied Payload: The SatDbTmAttenuation data object with the updated attenuation .

Return Codes:

Code	Description
200	OK, returns a document describing the requested satellite.
400	BAD_REQUEST, antennaId or tmId could not be parsed to a numeric identifier. In this case an ApiError document is returned, describing details of the error.
401	not logged in. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case an ApiError document is returned,describing details of the error.

PATCH /api/v1/antennas/{antennaId}/tm-atten/{tmId}

Modifies the attenuation values for an {antennaId} / {tmId} combination. Returns the attenuation set as a SatDbTmAttenuation data object.

Parameters:

Parameter	Description
antennaId	Id of the antenna.
tmId	Id of the tm channel.

Expected Payload: One or both attenuation values to set in aSatDbTmAttenuation data object. Only the attenuation values are read from this data object, any IDs passed to this API function are ignored. The IDs are defined solely by the URL parameters.

Replied Payload: The SatDbTmAttenuation data object with the updated attenuation.

Return Codes:

Code	Description
200	OK, returns a document describing the requested satellite.
400	BAD_REQUEST, antennaId or tmId could not be parsed to a numeric identifier. In this case an ApiError document is returned, describing details of the error.
401	not logged in. In this case anApiError document is returned, describing details of the error.
404	no attenuation record exists with the given*satelliteId / tmId*. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

DELETE /api/v1/antennas/{antennaId}/tm-atten/{tmId}

Deletes the attenuation values addressed by {antennaId} / {tmId}.

Parameters:

Parameter	Description
antennaId	Id of the antenna.
tmId	Id of the tm channel.

Expected Payload: none

Replied Payload: none.

Return Codes:

Code	Description
204	OK, the record has been deleted
400	BAD_REQUEST, antennaId or tmId could not be parsed to a numeric identifier. In this case an ApiError document is returned, describing details of the error.
401	not logged in. In this case an ApiError document is returned, describing details of the error.
404	no attenuation record exists with the given *satelliteId* / *tmId*. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case an ApiError document is returned,describing details of the error.

1.2.31 /api/v1/satdetails

This API endpoint reports all relevant data for a satellite in a compact form. It has been added for convenience, unburdening the front end programmer from the need to gather this information from different database tables using multiple API calls.

Supported HTTP methods: GET, HEAD, OPTIONS

GET /api/v1/satdetails/{satelliteId}

Returns detailed properties of the satellite addressed by {satelliteId} as aSatDbSatDetails data object.

Parameters:

Parameter	Description
satelliteId	Id of the satellite.

Expected Payload: none

Replied Payload: The requested SatDbSatDetails data object.

Return Codes:

Code	Description
200	OK, returns a document describing the requested satellite.
401	not logged in. In this case an ApiError document is returned, describing details of the error.

Code	Description
404	satellite with the given *satelliteId* does not exists. In this case anApiError document is returned, describing details of the error.

1.2.32 /api/v1/tleimport

This API endpoint lets you upload and import a list of TLE parameter sets from a data file. The expected file format is compatible to the well known 'geo.txt' file from celestrak.com. The file must contain three lines of text for each satellite, the firs line contains the satellite name, followed by two lines in TLE format. The norad number contained in both lines of the TLE format is used to identify for which satellite each record shall be used. The satellite name is treated as a comment only.

Supported HTTP methods: POST, HEAD, OPTIONS

POST /api/v1/tleimport

Expects a SatDbTleImport document containing the file to import along with some additional parameters.

Parameters: none

Expected Payload: The SatDbTleImport document containing the file to import and the import mode to use.

Replied Payload: none, except for errors.

Return Codes:

Code	Description
204	OK, the file has been imported to the database. Do data is returned.
400	The file could not be parsed. In this case anApiError document is returned, describing details of the error.
401	not logged in. In this case anApiError document is returned, describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

1.2.33 /api/v1/eventreport

This API endpoint lets you create reports from the M&C system event database. The M&C systems managed by the backend share a common event database which store all faults, warnings and informational messages occurring in the particular M&C systems.

A report from the event database is created by sending anEventReportRequest document with a POST call to this API endpoint. The EventReportRequest defines the time span to read and some filter settings which allow to read a set of messages with a specific content.

The API call responds with a EventReport document containing the requested event report.

Supported HTTP methods: POST, HEAD, OPTIONS

POST /api/v1/eventreport

Expects an EventReportRequest document containing the time span to read and the report filter settings.

Parameters: none

Expected Payload: The EventReportRequest document containing the time span to read and the report filter settings.

Replied Payload: The EventReport document containing the requested report.

Return Codes:

Code	Description
200	OK, the request has been processed, the report generated is replied.
500	The event database does not reply (internal error). In this case anApiError document is returned, describing details of the error.
401	not logged in. In this case an ApiError document is returned, describing details of the error.

1.2.34 /api/v1/eventack

The sat-nms event database provides a "not yet acknowledged" flag with each event in the database. This flag is initially set for all faults. This API endpoint gives access to the "acknowledge" function which clears this flag for a single event or for all events which have the flag set.

To acknowledge on or all events, the front end has to send anEventAck document with a POST call to this API end point. The EventAck document defines if all events or a particular one shall be acknowledged.

Supported HTTP methods: POST, HEAD, OPTIONS

POST /api/v1/eventack

Expects an EventAck document which defines if all events or a particular one shall be acknowledged.

Parameters: none

Expected Payload: The EventAck document which defines if all events or a particular one shall be acknowledged.

Replied Payload: none, except for errors.

Return Codes:

Code	Description
204	OK, the request has been processed. Do data is returned. There is no error checking, acknowledging a non-existing event or an event which does not require acknowledging is not treated as a fault.
500	The event database does not reply (internal error). In this case anApiError document is returned, describing details of the error.
401	not logged in. In this case an ApiError document is returned, describing details of the error.

1.2.35 /api/v1/trackmem

When subscribing for a '.state.mode4' parameter of a SatService ACU-ODM, the front end receives a message which indicates 'a new tracking history for this antenna is available' instead of sending the bulky tracking history itself over the websocket connection.

After the front end received this message, it can use the /api/v1/trackmem API call to retrieve the full tracking history as a TrackingHistory data object.

Supported HTTP methods: GET, HEAD, OPTIONS

GET /api/v1/trackmem/{odmName}

Parameters:

Parameter	Description
odmName	The device name of the ODM for which the tracking history shall be read. This must be a fully qualified name including name of the M&C where the ODM is managed.

Expected Payload: none

Replied Payload: A TrackingHistory document containing the tracking history of the referenced antenna controller.

Return Codes:

Code	Description

Code	Description
200	OK, the requested TrackingHistory document is returned.
400	Either the requested device does not exist or the device has not yet sent a tracking history. This may happen if the front end did no subscribe for the 'state.mode4' of the ODM before or if the ODM is not operational. In this case an ApiError document is returned, describing details of the error.
401	not logged in. In this case an ApiError document is returned, describing details of the error.

1.2.36 /api/v1/treeview

This API end point lets you retrieve the treeview data structure as a whole or partially by requesting only a branch of the tree. The API also provides a PUT method for this end point which lets you replace the user defined part of the tree for a given M&C name.

Supported HTTP methods: GET, PUT, HEAD, OPTIONS

GET /api/v1/treeview

Gets the complete tree definition for all M&C systems configured in the backend. Includes the auto-generated and the user defined parts of the tree definition

Parameters: none

Expected Payload: none

Replied Payload: A TreeViewNode document containing the complete tree view tree, starting with the ROOT node of the tree.

Return Codes:

Code	Description
200	OK, the requested TreeViewNode document is returned.
401	Not logged in. In this case an ApiError document is returned, describing details of the error.

GET /api/v1/treeview/{mncName}

Gets the tree definition for a given M&C system. Includes the user defined part of the tree definition as well as the device list for this M&C system.

Parameters:

Parameter	Description

Parameter	Description
mncName	The name of the M&C for which the tree definition shall be read.

Expected Payload: none

Replied Payload: A TreeViewNode document containing the tree branch for the given M&C, starting with the MNCNODE node for this M&C.

Return Codes:

Code	Description
200	OK, the requested TreeViewNode document is returned.
404	The requested M&C system does not exist or is actually offline. In this case an ApiError document is returned, describing details of the error.
401	Not logged in. In this case an ApiError document is returned, describing details of the error.

GET /api/v1/treeview/{mncName}/{subsystemName}

Gets the tree definition for a given M&C subsystem.

Parameters:

Parameter	Description
mncName	The name of the M&C for which the tree definition shall be read.
subsystemName	The name of the subsystem to retrieve. This subsystem name also may address nested subsystems, e.g. `SUBSYS-1.SUB_SYS-2` denotes a subsystem `SUSBSYS-2` contained in `SUBSYS-1`

Expected Payload: none

Replied Payload: A TreeViewNode document containing the tree branch for the given M&C subsystem, starting with the SUBSYSTEM node for this M&C subsystem.

Return Codes:

Code	Description
200	OK, the requested TreeViewNode document is returned.

Code	Description
404	Either the requested M&C system does not exist or is actually offline or no subsystem with the given name exists in this M&C system. In this case an ApiError document is returned, describing details of the error.
401	Not logged in. In this case an ApiError document is returned, describing details of the error.

PUT /api/v1/treeview/{mncName}

Replaces the tree definition for a given M&C system with a new version supplied by the client. Echoes the tree definition for this M&C system after the request has been processed.

Parameters:

Parameter	Description
mncName	The name of the M&C for which the tree definition shall be replaced.

Expected Payload: A TreeViewNode document containing the user defined part tree branch for the given M&C, starting with the MNCNODE node for this M&C. Only the SUBSYSTEM children of this node are processed, if the TReeViewNode has a DEVICELIST child, this is ignored.

Replied Payload: A TreeViewNode document containing the tree branch for the given M&C, starting with the MNCNODE node for this M&C. This is the tree view branch after the request has been processed, it includes the DEVICELIST for this M&C.

Return Codes:

Code	Description
200	OK, the requested TreeViewNode document is returned.
404	The requested M&C system does not exist or is actually offline. In this case an ApiError document is returned, describing details of the error.
401	Not logged in. In this case an ApiError document is returned, describing details of the error.

1.2.37 /api/v1/debug

This API end point lets you send debug commands to the (already opened) debug terminal of a M&C.

Supported HTTP methods: POST, HEAD, OPTIONS

POST /api/v1/debug/{mncName}

Sends one line containing a debug command to the debug terminal of the given M&C. The websocket connection for the debug terminal must already be open to make the command work. Echoes the original command after the request has been processed.

Parameters:

Parameter	Description
mncName	The name of the M&C to send the debug command to.

Expected Payload: A DebugMessage document containing the line with the command to send.

Replied Payload: A copy of the DebugMessage which has been sent.

Return Codes:

Code	Description
200	OK, the echoed DebugMessage document is returned.
404	The requested M&C system does not exist, is actually offline or another user is already connected to the debug terminal at this M&C. In this case an ApiError document is returned, describing details of the error.
401	Not logged in. In this case an ApiError document is returned, describing details of the error.

1.2.38 /api/v1/redundancy

This API end point lets you control or interrogate the redundancy properties for redundant M&Cs.

Supported HTTP methods: GET, PUT, HEAD, OPTIONS

GET /api/v1/redundancy/{mncName}

Queries the state of the redundancy control logic for the given M&C

Parameters:

Parameter	Description
mncName	The name of the M&C to query.

Expected Payload: none.

Replied Payload: A RedundancyState object reporting the redundancy properties of the M&C.

Return Codes:

Code	Description
200	OK, a RedundancyState document is returned.
400	The requested M&C system is not configured as a redundant pair. In this case an ApiError document is returned, describing details of the error.
404	The requested M&C system does not exist. In this case anApiError document is returned, describing details of the error.
401	Not logged in. In this case an ApiError document is returned, describing details of the error.

GET /api/v1/redundancy

Queries the state of the redundancy control logic for all redundant M&Cs.

Parameters: none.

Expected Payload: none.

Replied Payload: A RedundancySummary object reporting the redundancy properties of all redundant M&Cs.

Return Codes:

Code	Description
200	OK, a RedundancySummary document is returned.
401	Not logged in. In this case an ApiError document is returned, describing details of the error.

PUT /api/v1/redundancy/{mncName}

Sends a redundancy command to either switch a redundant M&C pair or to enable/disable the M&C redundancy.

Parameters:

Parameter	Description
mncName	The name of the M&C to send the redundancy command to.

Expected Payload: A RedundancyCmd document containing the command to send.

Replied Payload: A copy of the RedundancyCmd which has been sent.

Return Codes:

Code	Description
200	OK, the echoed RedundancyCmd document is returned.
404	The requested M&C system does not exist. In this case anApiError document is returned, describing details of the error.
401	Not logged in. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

1.2.39 /api/v1/restart

This API end point lets you send a restart command to a given M&C.

Supported HTTP methods: PUT, HEAD, OPTIONS

PUT /api/v1/restart/{mncName}

Sends a restart command to the given M&C.

Parameters:

Parameter	Description
mncName	The name of the M&C to send the redundancy command to.

Expected Payload: A RestartCmd document with *cmd* set to 'RESTART'. With other values of *cmd* the restart command is ignored.

Replied Payload: A copy of the RestartCmd which has been sent.

Return Codes:

Code	Description
200	OK, the echoed RestartCmd document is returned.
400	Either the RestartCmd passed to the call did not contain cmd=RESTART or the requested M&C system is offline and cannot be restarted. In this case an ApiError document is returned, describing details of the error.

Code	Description
404	The requested M&C system does not exist. In this case anApiError document is returned, describing details of the error.
401	Not logged in. In this case an ApiError document is returned, describing details of the error.

1.2.40 /api/v1/devicesetup

This API end point lets read and write the device setup for a specific M&C.

Writing a device setup as a whole to the M&C requires a M&C restart to make the M&C read and interpret the changed setup. Hence this is to be considered a legacy operation which will be replaced by API calls to modify the device configuration of the M&C 'on the fly' without restarting it, as soon as these API calls are available.

Supported HTTP methods: GET, PUT, HEAD, OPTIONS

GET /api/v1/devicesetup/{mncName}

Get the device setup for the given M&C.

Parameters:

Parameter	Description
mncName	The name of the M&C to read the device setup from.

Expected Payload: none.

Replied Payload: A DeviceSetup document describing the device configuration of the given M&C.

Return Codes:

Code	Description
200	OK, a DeviceSetup document is returned.
404	The requested M&C system does not exist. In this case anApiError document is returned, describing details of the error.
401	Not logged in. In this case an ApiError document is returned, describing details of the error.

PUT /api/v1/devicesetup/{mncName}

Sends a new device configuration to the given M&C. The configuration will be saved as 'vlc.setup' on this M&C, replacing the existing configuration. The M&C will use this configuration after the next restart (see /api/v1/restart).

Parameters:

Parameter	Description
mncName	The name of the M&C to send the device setup to.

Expected Payload: The DeviceSetup document containing the new / modified device configuration.

Replied Payload: Echoes back the DeviceSetup which has been sent.

Return Codes:

Code	Description
200	OK, the echoed DeviceSetup document is returned.
404	The requested M&C system does not exist. In this case anApiError document is returned, describing details of the error.
401	Not logged in. In this case an ApiError document is returned, describing details of the error.

1.2.41 /api/v1/drivers

This API end point returns a list of the available device drivers for a specific M&C.

Supported HTTP methods: GET, HEAD, OPTIONS

GET /api/v1/drivers

Get the list of the available device drivers. The list is fetched from the primary M&C.

Parameters: none

Expected Payload: none.

Replied Payload: An ItemList document containing the names of all device drivers available.

Return Codes:

Code	Description
200	OK, an ItemList document is returned.

Code	Description
400	The primary M&C is offline, cannot be queried. In this case anApiError document is returned, describing details of the error.
401	Not logged in. In this case an ApiError document is returned, describing details of the error.

1.2.42 /api/v1/protocols

This API end point returns a list of the available device communication protocols. The list is fetched from the primary M&C.

Supported HTTP methods: GET, HEAD, OPTIONS

GET /api/v1/protocols

Get the list of all available device communication protocols. The list is fetched from the primary M&C.

Parameters: none.

Expected Payload: none.

Replied Payload: An ItemList document containing the names of all device communication protocols available.

Return Codes:

Code	Description
200	OK, an ItemList document is returned.
400	The primary M&C is offline, cannot be queried. In this case anApiError document is returned, describing details of the error.
401	Not logged in. In this case an ApiError document is returned, describing details of the error.

1.2.43 /api/v1/preferredprotocols

This API end point returns a list of the preferred device communication protocols a particular device driver uses. The list is fetched from the primary M&C.

Supported HTTP methods: GET, HEAD, OPTIONS

GET /api/v1/preferredprotocols/{driverName}

Get the list of the preferred device communication protocols a particular device driver uses.

Actually a ***sat-nms*** devices driver defines exactly one preferred protocol, hence this list will always have exactly one element. Future ***sat-nms*** versions may permit to define multiple preferred protocols for a driver.

Parameters:

Parameter	Description
driverName	The name of the device driver to check.

Expected Payload: none.

Replied Payload: An ItemList document containing the names of the preferred device communication protocols of the given driver.

Return Codes:

Code	Description
200	OK, an ItemList document is returned.
400	The primary M&C is offline, cannot be queried. In this case anApiError document is returned, describing details of the error.
404	The device driver this call refers to does not exist. In this case anApiError document is returned, describing details of the error.
401	Not logged in. In this case an ApiError document is returned, describing details of the error.

1.2.44 /api/v1/filerecordings

This API end point returns the recorded data for a File-Recorder device

Supported HTTP methods: GET, HEAD, OPTIONS

GET /api/v1/filerecordings/{mncName}/{deviceName}

Reads the recorded data from the given File-Recorder device

Parameters:

Parameter	Description
mncName	The name of the M&C to read the data from
deviceName	The name of the File-Recorder device to read the data from

Expected Payload: none.

Replied Payload: The recorded data in CSV format (text/plain). Columns are separated by comma characters, the first column contains the time stamp in 'YYYY-MM-DD HH:MM:SS' format (UTC). All other columns contain the recorded data. Example:

```
2023-02-16 09:12:38,28.3,,,
2023-02-16 09:13:39,28.4,,,
2023-02-16 09:14:39,28.5,,,
2023-02-16 09:15:38,28.4,,,
2023-02-16 09:16:38,28.5,,,
2023-02-16 09:17:38,28.3,,,
2023-02-16 09:18:38,28.4,,,
2023-02-16 09:19:38,28.4,,,
```

Return Codes:

Code	Description
200	OK, the requested file recorder data is returned.
400	The M&C is offline, cannot be queried. In this case anApiError document is returned, describing details of the error.
401	Not logged in. In this case an ApiError document is returned, describing details of the error.
404	Either the referred M&C or the device does not exists or the device is not a File-Recorder device. In this case an ApiError document is returned, describing details of the error.

1.2.45 /api/v1/filerecordersettings

Reads and stores the diagram settings for a given file recorder device.

Supported HTTP methods: GET, PUT, HEAD, OPTIONS

GET /api/v1/filerecordersettings/{mncName}/{deviceName}

Reads the diagram settings for a given file recorder device.

Parameters:

Parameter	Description
mncName	The name of the M&C to read the data from
deviceName	The name of the File-Recorder device to read the data from

Expected Payload: none.

Replied Payload: The diagram settings as a FRViewProperties document.

Return Codes:

Code	Description
200	OK, the requested file recorder data is returned.
400	The M&C is offline, cannot be queried. In this case anApiError document is returned, describing details of the error.
401	Not logged in. In this case an ApiError document is returned, describing details of the error.
404	Either the referred M&C or the device does not exists or the device is not a File-Recorder device. In this case an ApiError document is returned, describing details of the error.

PUT /api/v1/filerecordersettings/{mncName}/{deviceName}

Stores the diagram settings for a given file recorder device.

Parameters:

Parameter	Description
mncName	The name of the M&C to store the data to
deviceName	The name of the File-Recorder device to store the data for

Expected Payload: The diagram settings to store as a FRViewProperties document.

Replied Payload: Echoes back the stored data (after storing it at the destination M&C and reading it back from there).

Return Codes:

Code	Description
200	OK, the echoed back diagram settings document is returned.
400	The M&C is offline, data cannot be stored. In this case anApiError document is returned, describing details of the error.
401	Not logged in. In this case an ApiError document is returned, describing details of the error.

Code	Description
404	Either the referred M&C or the device does not exists or the device is not a File-Recorder device. In this case an ApiError document is returned, describing details of the error.

1.2.46 /api/v1/acutargets

Reads the targets / satellites stored on a given ACU.

Supported HTTP methods: GET, HEAD, OPTIONS

GET /api/v1/acutargets/{mncName}/{deviceName}

Reads the targets / satellites stored on a given ACU.

Parameters:

Parameter	Description
mncName	The name of the M&C to read the data from
deviceName	The name of the device to read the data from. This may be either the name of the ACU device itself or the name of the Antenna-Management device controlling the ACU

Expected Payload: none.

Replied Payload: The target list of the queries ACU as a AcuTargetList document.

Return Codes:

Code	Description
200	OK, the requested file recorder data is returned.
400	The M&C is offline, cannot be queried. In this case anApiError document is returned, describing details of the error.
401	Not logged in. In this case an ApiError document is returned, describing details of the error.
404	Either the referred M&C or the device does not exists or the device does not provide a target list. In this case an ApiError document is returned, describing details of the error.

1.2.47 /api/v1/inventory

This API endpoint is used to inspect or modify the backend's inventory of items / devices.

Supported HTTP methods: GET, POST, PATCH, HEAD, OPTIONS

GET /api/v1/inventory/{itemId}

Gets the inventory item with the given ID.

Parameters:

Parameter	Description
itemId	The itemId of the item to get

Expected Payload: none.

Replied Payload: An InventoryItem record with the given ID

Return Codes:

Code	Description
200	OK, the requested InventoryItem document is returned.
401	Not logged in. In this case an ApiError document is returned, describing details of the error.
404	No InventoryItem with this ID exists or the backend does not support the inventory database. In this case an ApiError document is returned, describing details of the error.

GET /api/v1/inventory/{itemId}/devices

Gets the list of InventoryDevice objects mapped to the given inventory item.

Parameters:

Parameter	Description
itemId	The itemId of the item to get

Expected Payload: none.

Replied Payload: An array of InventoryDevice records mapped to the item with the given ID

Return Codes:

Code	Description
200	OK, the requested InventoryDevice Array is returned.

Code	Description
401	Not logged in. In this case an ApiError document is returned, describing details of the error.
404	No InventoryItem with this ID exists or the backend does not support the inventory database. In this case an ApiError document is returned, describing details of the error.

GET /api/v1/inventory/{itemId}/devices/{deviceId}

Gets the InventoryDevice objejct with the given ID combination.

Parameters:

Parameter	Description
itemId	The itemId of the item to get
deviceId	The deviceId of the device to get

Expected Payload: none.

Replied Payload: An InventoryDevice record with the given ID combination.

Return Codes:

Code	Description
200	OK, the requested InventoryDevice document is returned.
401	Not logged in. In this case an ApiError document is returned, describing details of the error.
404	No InventoryItem with this ID exists or the backend does not support the inventory database. In this case an ApiError document is returned, describing details of the error.

POST /api/v1/inventory

Creates a new InventoryItem entry in the database.

Parameters: none.

Expected Payload:An InventoryItem record containing the fields of the record to create. The following rules apply:

- Fields which are null or missing are set to 'null' in the record.
- The field 'itemId' is ignored. The POST command creates a new record, the database

assigns a unique itemId to the created record.
- The 'modified' field is set to the time when the command is executed, a 'modified' time specified in the supplied data is ignored.

Replied Payload: The created InventoryItem record

Return Codes:

Code	Description
200	OK, the modified InventoryItem document is returned.
401	Not logged in. In this case anApiError document is returned, describing details of the error.
404	No InventoryItem with this ID exists or the backend does not support the inventory database. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

POST /api/v1/inventory/{itemId}/devices

Creates a new InventoryDevice entry in the database and attaches this to the given item.

Parameters:

Parameter	Description
itemId	The itemId of the item to add the new device to

Expected Payload:An InventoryDevice record containing the fields of the record to create. The following rules apply:

- Fields which are null or missing are set to 'null' in the record.
- The field 'deviceId' is ignored. The POST command creates a new record, the database assigns a unique deviceId to the created record.
- The field 'itemId' is ignored. The POST command always takes the the 'itemId' from the URL and sets this in the created record.
- The 'modified' field is set to the time when the command is executed, a 'modified' time specified in the supplied data is ignored.

Replied Payload: The created InventoryItem record

Return Codes:

Code	Description

Code	Description
200	OK, the modified InventoryDevice document is returned.
401	Not logged in. In this case an ApiError document is returned, describing details of the error.
404	No InventoryItem with this ID exists or the backend does not support the inventory database. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

PATCH /api/v1/inventory/{itemId}

Modifies the inventory item with the given ID.

Parameters:

Parameter	Description
itemId	The itemId of the item to modify

Expected Payload:An InventoryItem record containing the fields to modify. The following rules apply:

- Fields which are null or missing are not changed in the record.
- To set a field to 'null' set it to an empty string.
- The field 'itemId' is ignored. The PATCH command modifies the record specified in the URI, the 'itemId' ofthis record never changes.
- The 'modified' field is set to the time when the command is executed, a 'modified' time specified in the supplied data is ignored.

Replied Payload: The InventoryItem record after modification

Return Codes:

Code	Description
200	OK, the modified InventoryItem document is returned.
401	Not logged in. In this case an ApiError document is returned, describing details of the error.
404	No InventoryItem with this ID exists or the backend does not support the inventory database. In this case an ApiError document is returned, describing details of the error.

Code	Description
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

PATCH /api/v1/inventory/{itemId}/devices/{deviceId}

Modifies the InventoryDevice object with the given ID combination.

Parameters:

Parameter	Description
itemId	The itemId of the item to modify
deviceId	The deviceId of the device to modify

Expected Payload:An InventoryDevice record containing the fields to modify. The following rules apply:

- Fields which are null or missing are not changed in the record.
- To set a field to 'null' set it to an empty string.
- The field 'deviceId' is ignored. The PATCH command modifies the record specified in the URI, the 'deviceId' of this record never changes.
- The 'modified' field is set to the time when the command is executed, a 'modified' time specified in the supplied data is ignored.

Replied Payload: The InventoryDevice record after modification

Return Codes:

Code	Description
200	OK, the modified InventoryDevice document is returned.
401	Not logged in. In this case an ApiError document is returned, describing details of the error.
404	No InventoryItem with this ID exists or the backend does not support the inventory database. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

DELETE /api/v1/inventory/{itemId}

Deletes the inventory item with the given ID. As a side effect, allInventoryDevice and InventoryLogEntry objects referring to this inventory item are deleted as well.

Parameters:

Parameter	Description
itemId	The itemId of the item to delete

Expected Payload: none.

Replied Payload: none.

Return Codes:

Code	Description
204	OK, the addressed InventoryItem has been deleted.
401	Not logged in. In this case an ApiError document is returned, describing details of the error.
404	No InventoryItem with this ID exists or the backend does not support the inventory database. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case an ApiError document is returned,describing details of the error.

DELETE /api/v1/inventory/{itemId}/devices/{deviceId}

Deletes the InventoryDevice obkect with the given ID combination.

Parameters:

Parameter	Description
itemId	The itemId of the item to delete
deviceId	The deviceId of the device to delete

Expected Payload: none.

Replied Payload: none.

Return Codes:

Code	Description
204	OK, the addressed InventoryDevice has been deleted.

Code	Description
401	Not logged in. In this case an ApiError document is returned, describing details of the error.
404	No InventoryItem with this ID exists or the backend does not support the inventory database. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

POST /api/v1/inventory/list

Gets a list of inventory items matching the filter information sent with the request.

Parameters: none

Expected Payload: An InventoryItem document containing the filter to be applied. Any fields in the document which are missing or null are intepreted as a wildcard. Specified fields must *all* match to include the item in the returned list. If a 'modified' timestamp is given in the filter description, it is interpreted as 'must be modified at this time or later'.

Replied Payload: An array of InventoryItem records fulfilling the filter spec. The list is sorted by modification time, oldest first. If no records match the filter, an empty list is returned.

Return Codes:

Code	Description
200	OK, the requested InventoryItem array is returned.
401	Not logged in. In this case an ApiError document is returned, describing details of the error.
404	The backend does not support the inventory database. In this case anApiError document is returned, describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

GET /api/v1/inventory/log/{entryId}

Gets the inventory log entry with the given ID.

Parameters:

Parameter	Description

Parameter	Description
entryId	The entryId of the log entry to get

Expected Payload: none.

Replied Payload: An InventoryLogEntry record with the given ID

Return Codes:

Code	Description
200	OK, the requested InventoryLogEntry document is returned.
401	Not logged in. In this case an ApiError document is returned, describing details of the error.
404	No InventoryLogEntry with this ID exists or the backend does not support the inventory database. In this case an ApiError document is returned, describing details of the error.

POST /api/v1/inventory/log

Adds a new InventoryLogEntry to the database.

Parameters: none.

Expected Payload:An InventoryLogEntry record containing the fields of the record to create. The following rules apply:

- log entries cannot be modified / rewritten. All fields must be supplied with their desired values.
- The field 'entryId' is ignored. The POST command creates a new record, the database assigns a unique itemId to the created record.
- The 'created' field is set to the time when the command is executed, a 'created' time specified in the supplied data is ignored.

Replied Payload: The created InventoryLogEntry record

Return Codes:

Code	Description
200	OK, the modified InventoryLogEntry document is returned.
401	Not logged in. In this case an ApiError document is returned, describing details of the error.

Code	Description
404	No InventoryLogEntry with this ID exists or the backend does not support the inventory database. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

POST /api/v1/inventory/log/list

Gets a list of inventory log entries matching the filter information sent with the request.

Parameters: none

Expected Payload: An InventoryLogEntry document containing the filter to be applied. Any fields in the document which are missing or null are intepreted as a wildcard. Specified fields must *all* match to include the entry in the returned list. If a 'modified' timestamp is given in the filter description, it is interpreted as 'must be created at this time or later'.

Replied Payload: An array ofInventoryLogEntry records fulfilling the filter spec. The list is sorted by creation time, oldest first.

Return Codes:

Code	Description
200	OK, the requested InventoryLogEntry array is returned.
401	Not logged in. In this case anApiError document is returned, describing details of the error.
404	The backend does not support the inventory database. In this case anApiError document is returned, describing details of the error.

1.2.48 /api/v1/documents

This API endpoint is used to inspect or modify the backend's list of documents.

Supported HTTP methods: GET, POST, PATCH, HEAD, OPTIONS

GET /api/v1/documents/{documentId}

Gets the documents item with the given ID.

Parameters:

Parameter	Description

Parameter	Description
documentId	The documentId of the item to get

Expected Payload: none.

Replied Payload: An DocumentItem record with the given ID. The DocumentItem does not contain its 'fileContent' field, use the '/api/v1/documents/content/' API call to retrieve the document content.

Return Codes:

Code	Description
200	OK, the requested DocumentItem document is returned.
401	Not logged in. In this case an ApiError document is returned, describing details of the error.
404	No DocumentItem with this ID exists or the backend does not support the inventory database. In this case an ApiError document is returned, describing details of the error.

POST /api/v1/documents

Creates a new DocumentItem entry in the database.

Parameters: none.

Expected Payload:An DocumentItem record containing the fields of the record to create. The following rules apply:

- Fields which are null or missing are set to 'null' in the record.
- The field 'documentId' is ignored. The POST command creates a new record, the database assigns a unique documentId to the created record.
- The 'modified' field is set to the time when the command is executed, a 'modified' time specified in the supplied data is ignored.

Replied Payload: The created DocumentItem record. The DocumentItem does not contain its 'fileContent' field, use the '/api/v1/documents/content/' API call to retrieve the document content.

Return Codes:

Code	Description
200	OK, the modified DocumentItem document is returned.

Code	Description
400	The file already exists. In this case the file is not written and anApiError document is returned, describing details of the error.
401	Not logged in. In this case an ApiError document is returned, describing details of the error.
404	No DocumentItem with this ID exists or the backend does not support the inventory database. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

PATCH /api/v1/documents/{documentId}

Modifies the documents item with the given ID.

Parameters:

Parameter	Description
documentId	The documentId of the item to modify

Expected Payload:An DocumentItem record containing the fields to modify. The following rules apply:

- Fields which are null or missing are not changed in the record.
- To set a field to 'null' set it to an empty string.
- The field 'documentId' is ignored. The PATCH command modifies the record specified in the URI, the 'documentId' of this record never changes.
- The 'modified' field is set to the time when the command is executed, a 'modified' time specified in the supplied data is ignored.
- If the field 'fileContent' is contained, the stored file gets rewritten with this content.
- If 'fileName' and 'fileContent' are changed at the same time, the old file is deleted before writing the new one.

Replied Payload: The DocumentItem record after modification. The DocumentItem does not contain its 'fileContent' field, use the '/api/v1/documents/content/' API call to retrieve the document content.

Return Codes:

Code	Description
200	OK, the modified DocumentItem document is returned.

Code	Description
400	A file with the new name already exists. In this case the file is not written and an ApiError document is returned, describing details of the error.
401	Not logged in. In this case an ApiError document is returned, describing details of the error.
404	No DocumentItem with this ID exists or the backend does not support the documents database. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

DELETE /api/v1/documents/{documentId}

Deletes the documents item with the given ID. This also deletes the file this record refers to.

Parameters:

Parameter	Description
documentId	The documentId of the item to delete

Expected Payload: none.

Replied Payload: none.

Return Codes:

Code	Description
204	OK, the addressed DocumentItem has been deleted.
401	Not logged in. In this case an ApiError document is returned, describing details of the error.
404	No DocumentItem with this ID exists or the backend does not support the inventory database. In this case an ApiError document is returned, describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

GET /api/v1/documents/content/{documentId}

Gets the file content of the document item with the given ID.

Parameters:

Parameter	Description
documentId	The documentId of the item to get

Expected Payload: none.

Replied Payload: The file content of the data file this document record refers to.

Return Codes:

Code	Description
200	OK, the requested file is returned.
401	Not logged in. In this case an ApiError document is returned, describing details of the error.
404	No DocumentItem with this ID exists or the backend does not support the inventory database. In this case an ApiError document is returned, describing details of the error.

POST /api/v1/documents/list

Gets a list of documents items matching the filter information sent with the request.

Parameters: none

Expected Payload: An DocumentItem document containing the filter to be applied. Any fields in the document which are missing or null are intepreted as a wildcard. Specified fields must *all* match to include the item in the returned list. If a 'modified' timestamp is given in the filter description, it is interpreted as 'must be modified at this time or later'. The 'fileContent' field is ignored as a filter.

Replied Payload: An array of DocumentItem records fulfilling the filter spec. The list is sorted by file name, 'a' to 'z'. The DocumentItem records do not contain their 'fileContent' field, this API call is intended to list the items, not to to execute a bulk read of the documents.

Return Codes:

Code	Description
200	OK, the requested DocumentItem array is returned.
401	Not logged in. In this case an ApiError document is returned, describing details of the error.

Code	Description
404	The backend does not support the inventory database. In this case anApiError document is returned, describing details of the error.

1.2.49 /api/v1/thumbnail

This API call fetches a thumnail image from the backend. Thumbnail images are cached in the backend. If you subscribe for a thumbnail image via the websocket, the frontend only receives a messages which tells that a new thumbnail picture is available. The image itself then must be fetched with the `/api/v1/thumbnail` API call.

Supported HTTP methods: GET, HEAD, OPTIONS

GET /api/v1/thumbnail/{messageId}

Gets the thumbnail image for the given message ID (e.g. 'MNC-1.IRD-3.thumbnail') Please note, that the messageId must be subscribed or the API call will return a 404 Not Found because the backens has not subscribes for the thumbnail message at the M&C

Parameters:

Parameter	Description
messageId	The full message ID of the thumbnail parameter of the device

Expected Payload: none.

Replied Payload: An Thumbnail document containing the thumbnail image.

Return Codes:

Code	Description
200	OK, the requested Thumbnail document is returned.
401	Not logged in. In this case an ApiError document is returned, describing details of the error.
404	Not found. In this case an ApiError document is returned, describing details of the error.

1.2.50 /api/v1/backendinfo

This API endpoint gives the backend information for example name and role.

Supported HTTP methods: GET

GET /api/v1/backendinfo

Returns the BackendInfo data objects.

Parameters: none

Expected Payload: none

Replied Payload: BackendInfo data objects

Return Codes:

Code	Description
200	OK, returns a JSON document.
401	not logged in. In this case an ApiError document is returned,describing details of the error.

1.2.51 /api/v1/globalSettings

This API endpoint is used to get and or save global settings for the users.

Supported HTTP methods: GET, PATCH, HEAD, OPTIONS

GET /api/v1/globalSettings

Gets the saved settings if not Backend should retrun default values with setting keys.

Parameters: none

Expected Payload: none

Replied Payload: The list of stored and default settings as an array ofUISettings data objects.

Return Codes:

Code	Description
200	OK, returns a JSON document.
401	not logged in. In this case an ApiError document is returned,describing details of the error.

PATCH /api/v1/globalSettings

Save setting in the Database. If *setting key* already exists it will overwrite the value.

Parameters: none

Expected Payload: The UISettings data object to store.

Replied Payload: The list of stored and default settings as an array ofUISettings data objects.

Return Codes:

Code	Description
201	OK, returns a JSON document.
401	not logged in. In this case an ApiError document is returned,describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

DELETE /api/v1/globalSettings/{id}

Deletes the global settings with the given id.

Parameters:

Parameter	Description
id	The id of the settings to delete

Expected Payload: none.

Replied Payload: none.

Return Codes:

Code	Description
202	ACCEPTED, Return Object of UISettings.
401	Not logged in. In this case an ApiError document is returned, describing details of the error.
404	No user setting with this username exists . In this case anApiError document is returned, describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

1.2.52 /api/v1/userSettings

This API endpoint is used to save or get user settings for the users.

Supported HTTP methods: GET, PATCH, HEAD, OPTIONS

GET /api/v1/userSettings

Gets the saved settings in data base if not Backend should retrun default values with setting keys.

Parameters: none

Expected Payload: none.

Replied Payload: The list of stored and default settings as an array ofUISettings data objects.

Return Codes:

Code	Description
200	OK, returns a JSON document.
401	not logged in. In this case anApiError document is returned,describing details of the error.

PATCH /api/v1/userSettings

Save user setting in the Database.

Parameters: NONE

Expected Payload: The UISettings data object to store.

Replied Payload: The list of stored and default settings as an array ofUISettings data objects.

Return Codes:

Code	Description
201	OK, returns a JSON document with all user settings.
401	not logged in. In this case anApiError document is returned,describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

DELETE /api/v1/userSettings/{id}

Deletes the user settings with the given id.

Parameters:

Parameter	Description

Parameter	Description
id	The id of the settings to delete

Expected Payload: none.

Replied Payload: none.

Return Codes:

Code	Description
202	ACCEPTED, Return Object of UISettings.
401	Not logged in. In this case an ApiError document is returned, describing details of the error.
404	No user setting with this username exists . In this case anApiError document is returned, describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

DELETE /api/v1/userSettings

Deletes the user settings with username. Username is take from token by the backend.

Parameters: none

Expected Payload: none.

Replied Payload: none.

Return Codes:

Code	Description
202	ACCEPTED, Return list of UISettings array list.
401	Not logged in. In this case an ApiError document is returned, describing details of the error.
404	No user setting with this username exists . In this case anApiError document is returned, describing details of the error.
500	DB is not writable. In this case anApiError document is returned,describing details of the error.

1.3 WebSocket Communication

The API provides a websocket at /msgs to update volatile data asynchronously. The data to be sent over the websocket is controlled with a subscription model. With the sat-nms Software, everything is about device parameters: Every device parameter has a unique ID, changes of this parameters are transported as messages having the id and the value of this parameter as properties. The websocket is used one-way: There are only messages sent from the backend server to the front end. The /api/v1/subscribe API call controls which messages / parameters shall be send over the websocket.

1.3.1 Creating WebSockets

Creating a websocket id a two-step procedure: First a websocket ID must be obtained. This is done using the /api/v1/makewebsocket API call. This call prepares a websocket which can be connected to. It returns an ID for the newly prepared websocket. The client must remember this ID during the session, it is required for all subsequent operations on this websocket.

The second step is to connect to this websocket. The URL for these websockets is**/msgs**, all connections share this. To identify which websocket to open, a query parameter defining the websocket ID must added to the HTTP URI when opening the websocket:

```
/msgs&satnmsWebsocketId=XXXXX
```

XXXXX has to be replaced by the 5-character websocket ID obtained with the /api/v1/makewebsocket API call.

The now connected websocket does not receive any data.

1.3.2 Subscribing For Messages

To receive data over the websocket, the data must be subscribed for. With the sat-nms Software, each device parameter and each type/range description has its unique message ID. For each message ID an explicit subscription must be done to get this parameter and its updates. The /api/v1/subscribe API call does this, either for a single message ID or for a list of IDs. Subscriptions are made for a certain websocket instance, hence the ID of the websocket must be supplied with the subscribe call.

Subscribed messages are sent asynchronously, each time when a change of the parameter is detected. It is not guaranteed, that a subscribe results immediately in a message sent over the websocket. Parameters may be in the state "unknown", specially in the first minutes after starting a M&C. Such a parameter will be sent when the first value becomes available.

The /api/v1/subscribe API call is also is used to unsubscribe messages using the DELETE HTTP method. It is recommended to unsubscribe messages for which there are no more updates needed.

Closing the websocket automatically unsubscribes all pending subscriptions. There is no need to do this explicitly before closing the session.

1.3.3 Interpreting WebSocket Data

The websocket transports text messages containing single JSON data objects. These objects all are Message objects. Parsing and interpreting the Message objects can be done along the*type* field in the message. The value of this field determines the data type and further content of the message:

Token	Message Type
VALUE	Parameter, contains an update for a parameter value
RANGE	Range, contains a data type an valid range definition for a parameter
PONG	This could be an echoed message used by the front end as a heartbeat (see Echo / Heartbeat Function below)

Range messages contain an additional *dataType* field which specifies the data type of the parameter belonging to this range definition. Depending on *dataType*, there will me more fields in the range message, specifying the valid range of a numeric parameter, the available choices for an enumeration and more. The table below list all defined data types:

Token	Data Type
INTEGER	IntegerRange
HEX	HexRange
FLOAT	DoubleRange
BOOLEAN	BooleanRange
ALARM	AlarmFlagRange
CHOICE	EnumRange
OBJECT	ObjectRange

1.3.4 Echo / Heartbeat Function

"Ping" messages on websocket protocol level sent to the websocket are answered with a "Pong" message. This can be used to implement a heartbeat mechanism at protocol level.

At application level this is possible as well. The backend's websocket implementation echoes all text messages received from the front end back to it. It is recommended to use Message data objects for this and setting *type* to a value not used by the backend.

1.3.5 Closing Websockets

When a websocket is closed, the backend server deletes all subscriptions made for this websocket. The websocket cannot be re-opened, the websocket ID obtained for the closed websocket becomes invalid.

When there is no websocket opened in a session for 120 seconds, the backend server deletes the session, this is the way the backend server handles the case when the operator closes the browser window. *(not yet implemented)*

1.4 Debug Terminal Communication

The backend provides a relay mechanism to access the debug terminal at each M&C connected to the backend. While sending commands to the debug terminal is done with an API call, the messages showing in the terminal are sent to the front end through a websocket. The following paragraphs give an overview about this.

1.4.1 Opening A Debug Terminal

To open a debug terminal, the front end opens a websocket at the URI defined for this. It receives every line to be shown in the terminal window as one message on the websocket. The M&C allows only one client to connect to the debug terminal at a time, opening the websocket may fail if either the M&C is down or if another user is already connected to the debug terminal (either from the Web-UI or from a Java client).

The URI to connect to the debug terminal is `/debug&mncName=XXXXX`

Where XXXXX has to be replaced with the name of the M&C to connect to. Once opened, the websocket transports every line from the debug terminal as a DebugMessage data object to the front end.

1.4.2 Closing A Debug Terminal

While the websocket for a debug terminal is opened, this prevents other users from using this feature. Hence every debug terminal websocket should be closed by the front end as soon as it is no longer needed. Closing the websocket also closes the data connection the M&C debug terminal port and by this frees this resource.

1.4.3 Sending Debug Commands

Sending commands to the debug terminal is done with the/api/v1/debug API call. This also takes a DebugMessage data object payload, carrying the line to be entered at the debug terminal. /api/v1/debug API calls require a debug terminal websocket to the referenced M&C being already opened. The will fail if the websocket for this M&C is not open because this means that also the data connection to the M&C is not present.

1.4.4 M&C Redundancy Management

The backend provides a redundancy switching mechanism for M&C systems connected to it. A redundant pair of M&C systems share the M&C name, but the M&C systems have different IP addresses. The redundancy works asymmetrically (there is one primary and one backup M&C) and one-way (redundancy switch will never happen from backup to primary).

If the communication to the primary M&C breaks and cannot be re-established for some time, the backend stops the M&C service on the primary machine and starts it on the backup machine. From now on the backend connects to the IP of backup M&C to control it.

All configuration about this is done in the 'backend.properties' file. The second IP address for the backup M&C and parameters how long the backend shall retry to connect to the primary M&C before it does the switch over to the backup M&C.

The actual state of the redundancy for each M&C (pair) is reported through the/api/v1/mnclist API call. Each MncInfo record in the returned list defines a 'redundancy' field which tells about the actual state.

1.4.4.1 Redundancy Commands

While the general aspects of the redundancy are statically configured in the 'backend.properties' file, the backend provides a /api/v1/redundancy API call to control if the primary or the backup machine of a redundant pair shall be used. There are two commands which may be sent with this API call.

command	description
RESET	tells the backend to switch back to primary M&C. If the backend is already connected to the primary M&C, or if no redundancy is configured for this M&C, the 'RESET' command has no effect. If the backend cannot connect to the primary M&C for some time, it tries to switch over to the backup machine.
FORCE	forces a switch over to the backup M&C - even if the communication with the primary one works fine. Has no effect if no redundancy is configured or if the backend is already connected to the backup machine.

1.5 Data Models

This section describes the data models used as payload with the API calls mentioned above.

The data model descriptions appear in an order which groups related functions. An alphabetic list of all data models is listed below

- ActiveBackend
- AcuTarget
- AcuTargetList
- AlarmFlagRange
- ApiError
- ArrowElement
- AzElElement
- BackendInfo
- StreamKeyData
- BooleanRange

- ButtonElement
- ChannelData
- ChartElement
- ColorDefinition
- DatabaseVersion
- DebugMessage
- DeviceDefinition
- DeviceElement
- DeviceFrameDefinition
- DeviceFrameTab
- DevicePreset
- DeviceSetup
- DeviceThreadDefinition
- Dictionary
- DisplayElement
- DocumentItem
- DocumentList
- DoubleRange
- EditButtonElement
- EnumRange
- EventAck
- EventInfo
- EventLog
- EventReportRequest
- EventReport
- FaultElement
- FaultInfo
- FaultList
- FrameElement
- FRViewProperties
- FRViewTraceDescription
- FRViewPreset
- FRViewTraceScaling
- GaugeElement
- HexRange
- IconElement
- IntegerRange
- InventoryDevice
- InventoryItem
- InventoryLogEntry
- ItemList
- KeyValPair
- LatchingButtonElement
- LockButtonElement
- MCPElement
- Macro
- Message

- MncInfo
- MncList
- ObjectRange
- ParameterButtonElement
- ParameterElement
- Parameter
- PresetValue
- PresetVars
- DeviceVars
- RadioButtonElement
- Range
- RectElement
- RedundancyCmd
- RedundancyState
- RedundancySummary
- RestartCmd
- SatDbAntenna
- SatDbBeacon
- SatDbBeaconAttenuation
- SatDbI11Data
- SatDbPosition
- SatDbSatDetails
- SatDbSatOperator
- SatDbSatellite
- SatDbTLEData
- SatDbTableTracking
- SatDbTc
- SatDbTcAttenuation
- SatDbTleImport
- SatDbTm
- SatDbTmAttenuation
- SatDbState
- ScheduleEvent
- Schedule
- ScreenDefinition
- ScreenElement
- SpectrumElement
- SpectrumTrace
- StringRange
- SubscribeList
- SwitchElement
- TakeButtonElement
- TargetListElement
- TextElement
- Thumbnail
- TokenReply
- TrackingHistory

- TrackingPoint
- TreeViewNode
- UISettings
- UserList
- UserProperties
- WebsocketId
- XYChartElement

1.5.1 TokenReply

The TokenReply data model contains the data returned from a successful login or token refresh (API end point /token). The fields defined in this data model follow the OAuth2 recommendations.

Data Model

Key	Type	Value
access_token	String	The access token (JWT) to be added as 'Authorization: Bearer ...' header field with subsequent API calls.
refresh_token	String	The refresh token (JWT) to be used to refresh an expired access token with the /token API end point and *grant_type* set to 'refresh_token'.
token_type	String	Always 'JWT'
expires	Number	The number of seconds the access_token is valid

1.5.2 UserProperties

The UserProperties document contains some information about the user currently logged in. It is returned as a reply to the GET /api/v1/user call.

Data Model

Key	Type	Value
userName	String	The user (login) name.
fullName	String	The user's full name.
password	String	The user's password (encrypted).
groupList	String	A comma separated list of group identifiers this user is allowed to access.
privilege	Number	The user's privilege level (1..150)

Remarks

- The user's privilege level is very important for the front end as it specifies which user interface elements may be accessed by the user. With the sat-nms WebGUI the privilege level mechanism replaces the commonly used 'roles'. This works the following way:
    - Each active user interface element like an entry field, a drop down box has a privilege level defined. Predefined screens like standard device screens have these level set to standard values, with user defined screens each user interface element may be configured to an individual privilege level.
    - If the privilege level of the user is higher or equal to the level of the user interface element, he may change the parameter controlled by this element.
    - If the privilege level of the user is lower, he gets the actual value of the parameter displayed, but he may not change this value.
- The privilege level has a range from 1 to 150, commonly used values are 100 for standard operator access and 150 for admin privileges.

1.5.3 UserList

The UserList document contains the complete list of users known to the backend. It is returned as a reply to the GET /api/v1/users call.

Data Model

Key	Type	Value
users	Array	Contains the list of known users as an array ofUserProperties objects

1.5.4 ApiError

The backend server returns an ApiError document if something went wrong, i.e. The HTTP return code is not in the 200 range.

Data Model

Key	Type	Value
timestamp	String	The time and date when the event happened.
status	Number	The HTTP status code returned.
error	String	The textual description of *status*.
message	String	A message describing the reason of the error.
path	String	The path part of the URL which caused the error.

Remarks

- The *timestamp* has a format like `2020-01-16T09:40:48.079+0000`

- The fields *error* and *message* contain the same text in most cases.

1.5.5 MncList

The MncList is used with the /api/v1/mnclist API call to report the list of M&C systems managed by the backend.

Data Model

Key	Type	Value
mncList	Array	contains an array of MncInfo objects, each describing the properties of one M&C system manged by the backend. The first entry of the list is called the primary M&C, it is used to get the login data and the live event log. This (and only this) M&C has "primary" set true.

1.5.6 MncInfo

The MncInfo data model contains the properties of one entry in a MncList. It is used with the /api/v1/mnclist API call to report the list of M&C systems managed by the backend.

Data Model

Key	Type	Value
id	String	The ID / name of the M&C. This ID is used to identify the M&C in messages. See section Message below for details how the M&C ID is embedded in the message ID.
ipAddress	String	The IP address of the M&C server as defined in the backend configuration file.
connected	Boolean	The actual state of the connection to the M&C.
primary	Boolean	True marks this M&C as the primary one.
redundancy	String	Describes the state of the redundancy control for this M&C. Possible values are 'NONE' = no redundancy control configured for this M&C 'PRIMARY' = the backend connects to the primary IP address for the M&C 'BACKUP' = the backend connects to the backup IP address for the M&C

1.5.7 ActiveBackend

The ActiveBackend data model carries the name of the backend which actively controls M&C redundancy switching in a redundant backend configuration. It is used with the /api/v1/activebackend API call.

Data Model

Key	Type	Value
activeBackendName	String	The name of the backend which is or shall be the active backend

The name of a backend is defined in the 'backend.properties' file with the 'backend.name' key.

1.5.8 SubscribeList

The SubscribeList is used with the /api/v1/subscribe API call to define a list of message identifiers the front end wants to subscribe or to unsubscribe from.

Data Model

Key	Type	Value
subscribeList	Array	contains an array of strings, each string is on message ID to be subscribed / unsubscribed.

1.5.9 WebsocketId

The WebsocketId carries the ID of a newly created websocket instance. It is returned by the /api/v1/makewebsocket API call in the response body.

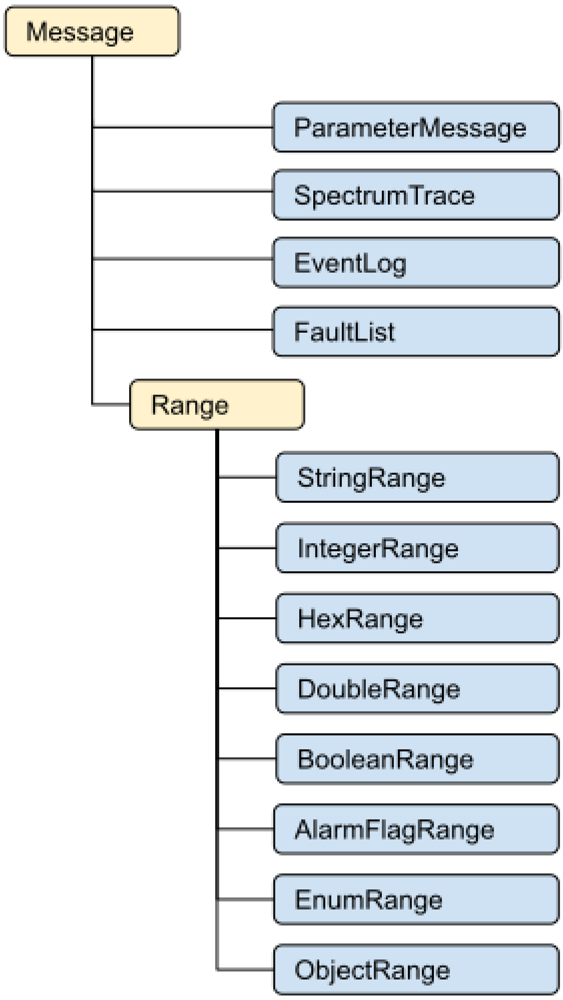
Data Model

Key	Type	Value
id	String	A newly created websocket ID. This is a unique 5-character string identifying the websocket for all further processing.

1.5.10 Message

With the sat-nms software, all volatile data gets exchanged as a Message. A message may carry a parameter like a value read from a device or a setting for this device. Beside this, data type and range information about parameters are exchanged as messages as well.

Because of this there are several flavors of messages defined, based of a hierarchical model derived from object oriented programming. The diagram below shows the known variants of Message as they depend on each other. Data models with yellow background do not exist in reality, they are defined as 'abstract base classes' which define the common parameters of their siblings.



Messages are used either with the /api/v1/peek and /api/v1/poke API calls or with the subscription model over a websocket connection.

The core property of a Message is its message ID. The message ID describes, which parameter of which device connected to which M&C server is addressed with the information of the message. The general format of a message ID is:

```
{M&C-Name}.{Device-Name}.{Parameter-Name}[.R]
```

The message ID is composed of at least three words with dots in between. The first word describes the M&C server to which the device is connected which is addressed by the second word. The third word identifies one parameter of this device, this identifier again may contain dots, if the parameter in the device are organized in a tree-like structure. Finally, the character sequence **.R** may be appended to the ID. This marks the message containing the range information about the parameter instead of the actual parameter value. Example:

```
MNC-1.DEVICE-3.parameter
```

Data Model

Key	Type	Value
id	String	The message ID associated with this message.
type	String	The message type (see below).
time	String	Timestamp in format 'YYYY-MM-DDTHH:MM.SSZ' (UTC)

Defined Message Types

Token	Data Type
VALUE	A Parameter message.
RANGE	A Range message.
SPECTRUM	A SpectrumTrace message. This message may not be sent with a /api/v1/poke call, it is read-only from the front end's point of view.
EVENTLOG	An EventLog message. This message may not be sent with a /api/v1/poke call, it is read-only from the front end's point of view.
FAULTLIST	A FaultList message. This message may not be sent with a /api/v1/poke call, it is read-only from the front end's point of view.
PONG	Echo reply (PONG is only an example, the front end may send any message type other than *VALUE* or *RANGE*, the backend will echo the message as it is received.)

Message IDs of Range messages always end with **.R** - this is the way the underlying ***sat-nms*** M&C software marks these messages.

1.5.11 Parameter

A Parameter message carries a value to or from a device controlled by the M&C server. When sent from the UI to the M&C server, the Parameter message commands a new value to be set at the device. When sent from the M&C server to the UI, the message informs about a change of the parameter. This applies to read-only parameters like state flags or measurement values and to settings-type parameters as well.

Data Model (extends Message)

Key	Type	Value
id	String	The message ID (see Message)

Key	Type	Value
type	String	The message type (always **VALUE**, see Message).
time	String	Timestamp in format 'YYYY-MM-DDTHH:MM.SSZ', see Message).
val	String	The value of the parameter, coded as a String.

The value contained in a Parameter message (*val*) may be of different types. The Range of the parameter defines this type, strictly spoken the *dataType* field in the Range message. For simplicity, all data types are coded as a string value in the Parameter message. The table below specifies, in which way parameter values are coded for all defined *dataType* flavors:

Range *dataType*	**Coding in the *val* field**
TEXT	UTF8 coded String.
INTEGER	Decimal number, consisting of digits 0..9
HEX	Decimal (!) number, consisting of digits 0..9
FLOAT	Floating point number with decimal point
BOOLEAN	Either **true** or **false** (all lower case)
ALARM	Either **true** or **false** (all lower case)
CHOICE	The selected choice as a UTF8 coded string. Please note, case is significant with choices. The values **ON** and **On** are treated as different ones.
OBJECT	ObjectRange parameter are not supported with this version of the API. Writing an ObjectRange parameter to the M&C is ignored, when queries an ObjectRange parameter always reports an empty string in *val*.

1.5.12 Range

A Range message contains the type definition and an optional specification for a range/limit validation for numeric parameters. The data model definition for Range is abstract, which means it never will be used in practice, only the data models derived from this will be used in the API.

Data Model (extends Message)

Key	Type	Value
id	String	The message ID (see Message)

Key	Type	Value
type	String	The message type (always **RANGE**, see Message).
time	String	Timestamp in format 'YYYY-MM-DDTHH:MM.SSZ', see Message).
dataType	String	A token defining the data type this Range object describes. Depending on *dataType*, there are more parameters present in the Range definition.
disabled	Bool	True tells the front end that this parameter is currently not in use. No value shall be displayed for this parameter, even if a corresponding Parameter message has been received. Any decorations like frame or descriptive label shall be displayed grayed/faded in order to show this state to the operator.
readonly	Bool	This parameter is for display only, the operator may not change it.

Defined Data Types

Token	Data Type
TEXT	StringRange
INTEGER	IntegerRange
HEX	HexRange
FLOAT	DoubleRange
BOOLEAN	BooleanRange
ALARM	AlarmFlagRange
CHOICE	EnumRange
OBJECT	ObjectRange

Remark: Range objects are also used in the PresetVars list of device driver variables. In this case *messageId* does not contain a fully qualified message ID, but the name of the variable.

1.5.13 StringRange

A StringRange parameter carries a free text.

Data Model (extends Range)

Key	Type	Value
id	String	The message ID (see Message)

Key	Type	Value
type	String	The message data type (always **RANGE**, see Message).
time	String	Timestamp in format 'YYYY-MM-DDTHH:MM.SSZ', see Message).
dataType	String	Always **TEXT** (see Range)
disabled	Bool	Inherited from Range
readonly	Bool	Inherited from Range

1.5.14 IntegerRange

An IntegerRange parameter carries an integer number. Internally the value is represented by a signed 64-bit number (data type *long*).

Data Model (extends Range)

Key	Type	Value
id	String	The message ID (see Message)
type	String	The message data type (always **RANGE**, see Message).
time	String	Timestamp in format 'YYYY-MM-DDTHH:MM.SSZ', see Message).
dataType	String	Always **INTEGER** (see Range)
disabled	Bool	Inherited from Range
readonly	Bool	Inherited from Range
checkLimits	Boolean	true = the front end must validate any entry for this parameter to be in the range *min ... max*. Invalid entries must be rejected, no values shall be sent to the server if outside the given range.
min	Number	The minimum value to be accepted with*checkLimits*=true
max	Number	The maximum value to be accepted with*checkLimits*=true
unit	String	A unit string to be displayed right of the parameter value (e.g. "MHz" for a frequency)

1.5.15 HexRange

A HexRange parameter carries an integer number. Internally the value is represented by a signed 64-bit number (data type *long*). HexRange is much like IntegerRange with the only difference, that values are printed at the UI in hexadecimal notation (no prefix like '0x' or '#') and user entries are expected in the same format.

Data Model (extends Range)

Key	Type	Value
id	String	The message ID (see Message)
type	String	The message type (always **RANGE**, see Message).
time	String	Timestamp in format 'YYYY-MM-DDTHH:MM.SSZ', see Message).
dataType	String	Always **HEX** (see Range)
disabled	Bool	Inherited from Range
readonly	Bool	Inherited from Range
checkLimits	Boolean	true = the front end must validate any entry for this parameter to be in the range *min ... max*. Invalid entries must be rejected, no values shall be sent to the server if outside the given range.
min	Number	The minimum value to be accepted with *checkLimits*=true
max	Number	The maximum value to be accepted with *checkLimits*=true
unit	String	A unit string to be displayed right of the parameter value (e.g. "MHz" for a frequency)

1.5.16 DoubleRange

A DoubleRange parameter carries a floating point value. Internally floating point values are represented by 64-bit floating point values (data type *double*, that gave the name for the DoubleRange).

Data Model (extends Range)

Key	Type	Value
id	String	The message ID (see Message)
type	String	The message data type (always **RANGE**, see Message).
time	String	Timestamp in format 'YYYY-MM-DDTHH:MM.SSZ', see Message).
dataType	String	Always **FLOAT** (see Range)
disabled	Bool	Inherited from Range
readonly	Bool	Inherited from Range

Key	Type	Value
checkLimits	Boolean	true = the front end must validate any entry for this parameter to be in the range *min ... max*. Invalid entries must be rejected, no values shall be sent to the server if outside the given range.
min	Number	The minimum value to be accepted with*checkLimits*=true
max	Number	The maximum value to be accepted with*checkLimits*=true
prec	Number	The precision (number of digits following the decimal point). Details are given in the paragraph **Number Formatting** below
unit	String	A unit string to be displayed right of the parameter value (e.g. "MHz" for a frequency)

Number Formatting

FLOAT variables are displayed with a fixed precision. The *prec* parameter defines how many digits right of the decimal point shall be displayed. Values of *prec* above 100 define a scientific formatted floating point with *prec - 100* digits precision (e.g. '0.123E-2' for *prec=103*)

1.5.17 BooleanRange

A BooleanRange parameter carries a BOOL value. It may be one of the values*true* or *false*.

Data Model (extends Range)

Key	Type	Value
id	String	The message ID (see Message)
type	String	The message type (always **RANGE**, see Message).
time	String	Timestamp in format 'YYYY-MM-DDTHH:MM.SSZ', see Message).
dataType	String	Always **BOOL** (see Range)
disabled	Bool	Inherited from Range
readonly	Bool	Inherited from Range

1.5.18 AlarmFlagRange

The AlarmFlagRange describes a sat-nms ALARM parameter. Like the BOOL parameter may be *true* (alarm) or *false* (no alarm). The AlarmFlagRange defines a descriptive text for the alarm flag which can be displayed beside the actual state.

Data Model (extends Range)

Key	Type	Value
id	String	The message ID (see Message)
type	String	The message data type (always **RANGE**, see Message).
time	String	Timestamp in format 'YYYY-MM-DDTHH:MM.SSZ', see Message).
dataType	String	Always **ALARM** (see Range)
disabled	Bool	Inherited from Range
readonly	Bool	Inherited from Range (always *true* with alarm flags)
description	String	A short description of the ALARM

1.5.19 EnumRange

The EnumRange describes a parameter which' value can be one of a given set of values. An UI typically implements this parameter type as a drop down box or as a list box which allows a single selection.

Data Model (extends Range)

Key	Type	Value
id	String	The message ID (see Message)
type	String	The message type (always **RANGE**, see Message).
time	String	Timestamp in format 'YYYY-MM-DDTHH:MM.SSZ', see Message).
dataType	String	Always **CHOICE** (see Range)
disabled	Bool	Inherited from Range
readonly	Bool	Inherited from Range
choices	Array	An array of string containing the choices for this parameter

1.5.20 ObjectRange

An ObjectRange describes a parameter carrying binary data, e.g. A spectrum analyzer trace. Special software is required to decode the data contained.

Data Model (extends Range)

Key	Type	Value

Key	Type	Value
id	String	The message ID (see Message)
type	String	The message data type (always **RANGE**, see Message).
time	String	Timestamp in format 'YYYY-MM-DDTHH:MM.SSZ', see Message).
dataType	String	Always **OBJECT** (see Range)
disabled	Bool	Inherited from Range
readonly	Bool	Inherited from Range

1.5.21 SpectrumTrace

A SpectrumTrace message contains the spectrum trace data distributed by a CSM-Spectrum-Analyzer device. The spectrum trace is much like a read-only parameter (it may not be commanded to the M&C), instead of a single value it carries an array of numeric values.

The array elements represent the spectrum trace, all elements are integer numbers describing the point position in pixels from the top of the display downwards. The length of the trace depends on the particular spectrum analyzer model, common trace lengths are 501 or 601 points. The first array element is the leftmost point (at *centerFrequency-frequencySpan/2*), the last array element is the is the rightmost point (at *centerFrequency+frequencySpan/2*).

The point positions in the array (array index, element value) refer to the native size of the spectrum analyzer display as read out by the driver. This spectrum display size in pixels may be read from the spectrum analyzer device as the 'info.screen' variable. The spectrum analyzer widget on the screen may however re-scale the display to any size requested.

Data Model (extends Message)

Key	Type	Value
id	String	The message ID (see Message)
type	String	The message type (always **SPECTRUM**, see Message).
time	String	Timestamp in format 'YYYY-MM-DDTHH:MM.SSZ', see Message).
trace	Array of Number	The spectrum trace, as an array of integer values, describing the point position in pixels from the top of the display downwards.

1.5.22 TrackingPoint

A TrackingPoint message contains pointing data of an antenna tracking controller for one tracking step. An array of TrackingPoint elements makes up the TrackingHistory which is described below

Data Model

Key	Type	Value
time	String	A time stamp telling when this tracking point was recorded. The format follows ISO 8601, example: `2021-03-26T08:02:30Z` , time zone is always UTC.
apos	Number	The azimuth pointing of the antenna after the tracking step.
epos	Number	The elevation pointing of the antenna after the tracking step.
blev	Number	The beacon level after the tracking step.
asucc	Boolean	True = the peaking process of the azimuth axis was successful.
apeak	Number	The azimuth peak position evaluated during the tracking step. Only meaningful if *asucc* is true.
esucc	Boolean	True = the peaking process of the elevation axis was successful.
epeak	Number	The elevation peak position evaluated during the tracking step. Only meaningful if *esucc* is true.

1.5.23 TrackingHistory

A TrackingHistory message contains the tracking memory of an antenna tracking controller. Actually, only the SatService ACU provides this data. The TrackingHistory message is read-only, meaning it may not be commanded to the M&C.

Data Model

Key	Type	Value
trace	Array of TrackingPoint	The tracking memory. The tracking points are ordered by time in the array, the first element refers to the oldest tracking point.

1.5.24 EventLog

An EventLog message carries the up to 24 most recent events that occurred in one of the M&C systems managed by the backend. The backend collects the events from the M&C systems and combines them to this list. The backend updates this message whenever a new message is added to the log, but at a maximum rate of one message per second.

To subscribe for the EventLog message, a/api/v1/subscribe API call must be made for the message ID `{Primary-M&C-Name}.SYSTEM.eventLog` where {Primary-M&C-Name} is the name / ID of the M&C reported with the /api/v1/mnclist API call with the "primary" key set true.

Data Model (extends Message)

Key	Type	Value
id	String	The message ID. This is always the message ID described above. (see Message)
type	String	The message type (always **EVENTLOG**, see Message).
time	String	Timestamp in format 'YYYY-MM-DDTHH:MM.SSZ', see Message).
events	Array	An array of EventInfo objects representing the recent events which occurred. This array has not more than 24 elements, directly after system start the array may be shorter or even empty if no events have been collected yet.

1.5.25 EventReport

An EventReport contains a list of EventInfo objects. The EventReport is the reply of the backend for a /api/v1/eventreport call when sent as a POST.

Data Model

Key	Type	Value
events	Array	An array of EventInfo objects.

1.5.26 EventReportRequest

The EventReportRequest specifies the filters to be applied when generating an EventReport. It has to be sent with an /api/v1/eventreport POST call.

Data Model

Key	Type	Value
from	String	Specifies the start of the time range of the report. Must be formatted following ISO 8601, example: `2021-03-26T08:02:30Z` , time zone is always UTC.
to	String	Specifies the end of the time range of the report. Must be formatted following ISO 8601, example: `2021-03-26T08:02:30Z` , time zone is always UTC. May be *null* if *length* is not zero.
length	Number	The length of the time range of the report in hours. If this value is zero, the report length is determines as the difference *to - from*. If the value is not *null*, the *to* value is ignored and *length* is used as the report length.

Key	Type	Value
mnc	String	The M&C name filter. Only events containing this text in their *source1* (M&C name) field are included in the report. If*mnc* is *null*, all events pass this filter.
device	String	The device name filter. Only events containing this text in their *source2* (device name) field are included in the report. If*device* is *null*, all events pass this filter.
host	String	The host name filter. Only events containing this text in their *origin1* (host name) field are included in the report. If *host* is *null*, all events pass this filter. Please note that only parameter change events have a host name set, other events will never match this filter.
user	String	The user name filter. Only events containing this text in their *origin2* (user name) field are included in the report. If *user* is *null*, all events pass this filter. Please note that only parameter change events have a user name set, other events will never match this filter.
maxlen	Number	The maximum length of the report (number of event messages). If more events than this number match the filter settings, only the oldest *maxlen* events of these will be contained in the report. If maxlen is null, the server side default for the maximum report length is used.
pri	String	The event priority filter. Must contain one of the following: ALL = include all events (no filter) INFO = include the events with priority INFO or higher INFO-ONLY = include the events with exactly priority INFO WARNING = include the events with priority WARNING or higher WARNING-ONLY = include the events with exactly priority WARNING FAULT = include the events with priority FAULT or higher FAULT-ONLY = include the events with exactly priority FAULT
text	String	The message text filter. Only messages which contain the given text are included in the report. If *text* consists of multiple words separated by space characters, the message text must contain all these words in arbitrary sequence. If *text* is *null*, all events pass this filter.

1.5.27 EventAck

The EventAck tells the backend which events in the event log shall be acknowledged. It is sent with a POST call to /api/v1/eventack.

Data Model

Key	Type	Value
ackAll	Boolean	Selects if a single event shall be acknowledged (false) of if all pending events shall be acknowledged (true).
event	EventInfo	The event to be acknowledged. *event* is ignored if *ackAll* is true.

1.5.28 EventInfo

An EventInfo contains the properties of one event within anEventReport or an EventLog message.

Data Model

Key	Type	Value
id	Number	A unique numeric identifier for this record. The event id is required to acknowledge a particular event.
pri	Number	The event priority coded as a one digit number 0 = suppressed, the event shall not be shown 1 = informational 2 = fault 3 = alarm 4 = warning
nak	Boolean	*(needs to be acknowledged)*This reads true for unacknowledged fault events.
msg	String	The event message text.
t1	String	*(time1)* A time stamp defining the time when the event was detected. The format follows ISO 8601, example: `2021-03-26T08:02:30Z` , time zone is always UTC.
t2	String	*(time2)* A time stamp defining the time when the event was stored in the event database. The format follows ISO 8601, example: `2021-03-26T08:02:30Z` , time zone is always UTC.
s1	String	*(source1)* The name / id of the M&C which generated this event.
s2	String	*(source2)* The name of the device which generated this event.
o1	String	*(origin1)* The name of the originator. Used with parameter change events to tell who commanded the change. May be a user name or in case of automatic parameter changes the name of the device which caused the change.
o2	String	*(origin2)* The host name / IP address from where a parameter change was commanded.

1.5.29 FaultList

A FaultList message carries the list of active faults and warnings which are pending at a given M&C. The backend updates this list whenever a fault changes at this M&C but at a maximum rate of one message per second.

To subscribe for the FaultList message, a/api/v1/subscribe API call must be made for the message ID `{M&C-Name}.SYSTEM.faultList` where {M&C-Name} is the name / ID of the M&C of interest.

Data Model (extends Message)

Key	Type	Value
id	String	The message ID. This is always the message ID described above. (see Message)
type	String	The message type (always **FAULTLIST**, see Message).
time	String	Timestamp in format 'YYYY-MM-DDTHH:MM.SSZ', see Message).
faults	Array	An array of FaultInfo objects representing list of actually pending faults and warnings at the given M&C.

1.5.30 FaultInfo

A FaultInfo contains the properties of one fault of warning reported in aFaultList message.

Data Model

Key	Type	Value
time	String	A time stamp telling when this fault changed its state. The format follows ISO 8601, example: `2021-03-26T08:02:30Z` , time zone is always UTC.
src	String	The name of the device which generated this event.
pri	Number	The event priority coded as a one digit number 0 = suppressed or device out-of-service, the fault shall not be shown 1 = informational 2 = fault 3 = alarm 4 = warning
msg	String	The fault message text.

1.5.31 DeviceFrameDefinition

A DeviceFrameDefinition document defines tabs to be shown in a sat-nms device screen. It resembles the content of a file in the *dframes* directory of the ***sat-nms*** M&C software.

DeviceFrameDefinition documents are returned by a GET /api/v1/dframes/{name} call.

Data Model

Key	Type	Value
name	String	The file name of the dframe definition
allowPresets	Boolean	true = the UI shall show the load/save preset buttons for this dframe
presetPrivilege	Number	the privilege level required to launch the preset load/save dialogs. This only is meaningful with *allowPresets=true*
tabs	Array	Contains a list of DeviceFrameTab objects, each describing one tab of the device frame

1.5.32 DeviceFrameTab

A DeviceFrameTab document defines one tab to be shown in a ***sat-nms*** device screen. It appears as part of the DeviceFrameDefinition data object.

Data Model

Key	Type	Value
icon	String	The name of the icon to be shown in the tab header
dscreen	String	The name of the dscreen to be shown in this tab
description	String	A description of what is shown in this tab, the Java UI shows this as a mouse over tooltip

1.5.33 ScreenDefinition

A ScreenDefinition document defines the layout of a screen / page to display. The document contains some global properties of the screen and an array of ScreenElement objects, which each define one UI widget with its properties.

With the sat-nms software there are two general types of screen definitions: 'user defined screens' and 'device screens'. While the first type may contain widgets which show parameters of any device controlled by the sat-nms M&C are 'device screens' bound to one particular device. Device screens are returned by a GET /api/v1/dscreens/{name} call, user screens by a GET /api/v1/uscreens/{name} call.

Data Model

Key	Type	Value
name	String	The file name of the screen definition
deviceRelative	Boolean	true = contains no device IDs, the front end application must add the device ID from the context of the screen
width	Number	Width of the screen (pixels in the Java client)
height	Number	Height of the screen (pixels in the Java client)
bgImage	String	Name of a background image to draw in the screen. An empty value states that there is no background image.
noEditTake	Boolean	true = exclude this screen from global edit/take if this feature has been set for the UI.
elements	Array	Contains ScreenElement objects, the list of the screen elements contained in this screen.

1.5.34 ScreenElement

A ScreenElement object describes the properties of one UI widget contained in a ScreenDefinition. This comprises the type of widget, its location and size and many more parameters which depend on the type of widget this ScreenElement defines. This way ScreenElement is kind of an abstract definition or 'base class' in terms of object oriented programming. ScreenElement defines the parameters which are common to all types of screen elements.

Data Model

Key	Type	Value
type	String	The type of the screen element. See the list of screen element types below for details.
xpos	Number	X-position of the upper left corner of the element in the screen (pixels in the Java client)
ypos	Number	Y-position of the upper left corner of the element in the screen (pixels in the Java client)
width	Number	Width of the element (pixels in the Java client)
height	Number	Height of the element (pixels in the Java client)

As mentioned above, a ScreenDefinition will never contain a ScreenElement with only the parameters shown in the table above. All ScreenElement objects will be of a certain screen element type and will contain all parameters defines for this element type. The parameters described for ScreenElement will be always included.

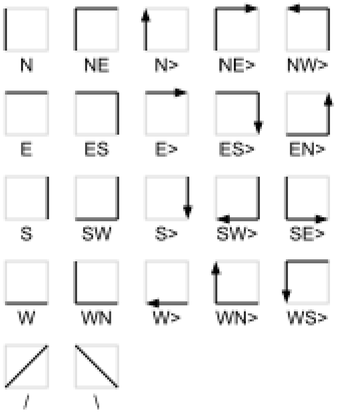
List of screen element types:

Element Type	Description
ArrowElement	Draws a horizontal/vertical line or arrow.
AzElElement	A chart showing the tracking history of an antenna controller as a point cloud in an elevation over azimuth coordinate system.
ButtonElement	Defines a button which launches another screen, e.g. for detail views.
ChartElement	A strip chart element, displaying a numeric parameter as an y/t diagram that automatically advances with 1 pixel / second.
DeviceElement	Places an icon into the screen which represents a device. This icon will display the device's operating/fault state by it's color / shape.
DisplayElement	A parameter display field which does not allow the parameter to be edited.
EditButtonElement	Places an EDIT button into the screen, implicitly forcing the EDIT/TAKE operating method for this screen instead of changing each parameter independently.
FaultElement	A special display element to display a fault flag. It shows the textual description of the fault and it's state in an entry field frame. The right mouse button shows a context menu to change the fault's priority
FrameElement	Draws a sunken 3D frame, may be used to group parameters.
GaugeElement	A gauge element, displaying a numeric parameter as a horizontal bar.
IconElement	Places an icon (PNG, GIF or JPG image) into the screen. Optionally the icon can be programmed to change with a parameter value.
LatchingButtonElement	A button which displays/controls a 2-state parameter using it's pressed state.
LockButtonElement	A button to control the lock state for exclusive operation for a number of devices.
MCPElement	The MCP display element integrates the measurement display of a spectrum analyzer device in multi channel measurement mode in a user screen.

Element Type	Description
ParameterButtonElement	A button which sends a certain parameter value when pressed.
ParameterElement	Either a parameter entry field or a drop down box, depending on the type of parameter.
RadioButtonElement	A parameter entry field specially for CHOICE parameters.
RectElement	Draws a rectangle.
SpectrumElement	The spectrum display element integrates the spectrum display of a spectrum analyzer device in a user screen.
SwitchElement	Like the device icon, but additionally displays the actual position of a switch (Meant to be used for block diagrams showing the true signal path.
TakeButtonElement	Places a TAKE button into the screen.
TargetListElement	The ODM Target List element shows the list of targets of a SatService-ACU-ODM antenna controller device. It permits to recall, save or delete target definitions of this type of antenna controller. It is specialized to this antenna controller, does not support other types.
TextElement	Displays a single line of text.
ThumbnailElement	Displays a thumbnail image fetched directly from a video processing device (encoder, decoder, gateway)
XYChartElement	This element shows the relation of two numeric variables in an X/Y diagram, featuring a trace which shows the recent history of the values with a configurable depth. The update rate, the diagram scaling and much more is configurable with this screen element.

1.5.35 ArrowElement

The ArrowElement draws a horizontal and/or a vertical line and optionally an arrowhead. The line's color is selectable, also may follow a parameter value. Strictly spoken the ArrowElement is a rectangle with only one or two sides drawn. The parameter *arrowCode* defines which sides of the rectangle shall be drawn and if an arrow shall be placed at the end of the line. The figure below shows all arrow codes defined:



Data Model (extends ScreenElement)

Key	Type	Value
type	String	Always "ArrowElement".
xpos	Number	see ScreenElement
ypos	Number	see ScreenElement
width	Number	see ScreenElement
height	Number	see ScreenElement
id	String	The message ID this element listens to. Is only relevant if *variableLineColor* reads *true*.
arrowCode	String	A 1..3 character code defining the line/arrow outline. See the figure above which arrow codes are possible.
color	String	The color used for the line / arrow.
variableLineColor	Boolean	true = set the line color from the value received from the parameter addressed by *id*.
useColorTranslation	Boolean	use the colors[] translation
colors	Array	of ColorDefinition objects. Defines how to translate values of the *id* parameter to color values if *useColorTranslation* reads *true*.

Variable Line Color

With *variableLineColor=true* the ArrowElement will change its line color depending on the value of the parameter addressed by *id*. This way the line can be used as a status indicator, e.g. in a block diagram signalling "the signal goes actually this way". The way the ArrowElement interprets the parameter value depends on the value of *useColorTranslation*.

If *useColorTranslation=false*, the ArrowElement tries to decode the line color directly from the parameter value. First it does the following tests on the parameter value (not case sensitive):

Parameter Value	**Line Color And Thickness**
ends with **BOLD**	The line is drawn thicker than normal
starts with **BLACK**	The line is drawn with this color
starts with **WHITE**	The line is drawn with this color
starts with **RED**	The line is drawn with this color
starts with **BLUE**	The line is drawn with this color
starts with **GREEN**	The line is drawn with this color
starts with **YELLOW**	The line is drawn with this color
starts with **GRAY**	The line is drawn with this color
contains **TRUE**	If the message ID identifies the parameter as an alarm flag (the ID contains 'fault'), the line is drawn **RED-BOLD**, for other parameters it is drawn **GREEN-BOLD**
contains **FAULT**	The line is drawn **RED-BOLD**
starts with **ON**	The line is drawn **GREEN-BOLD**

If none of the above conditions apply, the ArrowElement tries to interpret the parameter value as a hex coded color (#RRGGBB). If this also fails, the standard line color defined in *color* is used.

If *useColorTranslation=true*, the ArrowElement uses the ColorDefinition table in *colors[]* to decode the line's color and thickness.

1.5.36 ButtonElement

The ButtonElement creates a button which launches another user interface screen if pressed.

The screen to be launched may be another user defined screen or a predefined one.

Data Model (extends ScreenElement)

Key	Type	Value
type	String	Always "ParameterButtonElement".
xpos	Number	see ScreenElement
ypos	Number	see ScreenElement
width	Number	see ScreenElement
height	Number	see ScreenElement
privilege	Number	The privilege level required to make a button click send a value
label	String	A label shown on the button
icon	String	An icon shown on the button
enableId	String	The enable ID (may be empty)
enableValue	String	The value which must be matched to enable the field
font	String	The font to be used
color	String	The color used for the label (#RRGGBB)
action	String	The action to perform if the button is clicked (see below)
par1	String	A parameter that applies for the action to perform. The meaning of this field depends on the *action* selected (see below)
par2	String	A second parameter that applies for the action to perform. The meaning of this field depends on the *action* selected (see below)
par3	String	A third parameter that applies for the action to perform. The meaning of this field depends on the *action* selected (see below)
variableBackgroundColor	Boolean	true = use a variable background color
colorId	String	The message ID to listen for the background color
colors	Array	of ColorDefinition objects. Defines how to translate values of the *colorId* parameter to color values.

The *action* field of the ButtonElement defines what the button will do when clicked by the operator. The list below shows all known values for *action*, the meaning of this and a description of the parameters *par1* and *par2* which will have different meanings for particular action.

CHILD-SCREEN

Launches a new browser tab or window showing the user screen referenced with*par1*.

- *par1:* The name of the user screen to show.
- *par2:* null
- *par3:* null

REPLACE-SCREEN

Replaces the actual user screen by the one referenced with*par1*.

- *par1:* The name of the user screen to show.
- *par2:* null
- *par3:* null

LOAD-PRESET

Launches a "Load Preset" dialog which lets the operator select and apply a device preset from a list of presets which are defined for this particular type of device. Device presets are local to each M&C instance, have to be loaded and applied from there.

- *par1:* The name of the device to which the preset shall be applied to. The device name has the name of the M&C prepended where it is controlled. Example: `MYMNC.MYDEVICE` (must be set in the screen editor this way).
- *par2:* The name of the driver of this device.
- *par3:* An option search pattern to filter the list of displayed presets. All presets containing the given string (not case sensitive compare) shall be shown. If *par3* is null, all preset for the given device / driver combination shall be shown.

TRACK-VIEW

Launches a tracking view window. This window shows the tracking history of a given antenna controller device. Not all tracking controller devices are capable to report their tracking history.

- *par1:* The name of the antenna controller device for which the tracking history shall be shown. The device name has the name of the M&C prepended where it is controlled. Example: `MYMNC.MYACU` (must be set in the screen editor this way).
- *par2:* null
- *par3:* null

AZEL-VIEW

Launches a Az/El view window. This window shows the tracking history of a given antenna controller device as a cloud of dots in a azimuth over elevation coordinate system. Not all tracking controller devices are capable to report their tracking history.

- *par1:* The name of the antenna controller device for which the tracking history shall be shown. The device name has the name of the M&C prepended where it is controlled. Example: `MYMNC.MYACU` (must be set in the screen editor this way).
- *par2:* null
- *par3:* null

FREC-VIEW

Launches a File-Recorder view window. This window shows the data recorded by the File-Recorder device referenced with *par1* or the live data provided by this device.

- *par1:* The name of the File-Recorder device. The device name has the name of the M&C prepended where it is controlled. Example: `MYMNC.MY-FREC` (must be set in the screen editor this way). In the device screen of the file recorder, the special device name "." denotes *this* device.
- *par2:* The file recorder view preset number (0..7) to be applied. May be null, in this case no stored settings are applied.
- *par3:* null

SPECTR-VIEW

Launches a Spectrum Display window. This window in fact is a device screen of the CSM-Spectrum-Analyzer device.

- *par1:* The name of the spectrum analyzer device. The device name has the name of the M&C prepended where it is controlled. Example: `MYMNC.MY-FREC` (must be set in the screen editor this way).
- *par2:* A comma separated list of message-id/value pairs. Within each pair the message-id is separated from the value by one space character. Message ids are fully qualified, starting with the M&C name (must be set in the screen editor this way). The Spectrum Display window shall parse this list and send every message defined in the list with a poke call to the backend. This macro-like function is used to initialize the spectrum analyzer or to switch its input when the window is launched.
- *par3:* null

BROWSER-VIEW

Opens a new Browser window and displays a given URL in this window. This function is used to invoke the sat-nms online help and to launch the Web-GUI of certain devices.

- *par1:* The URL to show. Before launching the browser window, some replacements have to be done on the URL string:
    - If *par1* is a string consisting only of 4 decimal digits, this is a sat-nms online help topic number and the string must be expanded to the full URL from where this topic can be loaded by the browser.
    - If the ButtonElement is part of a device screen and the URL contains '@' characters, they have to be replaced with the (IP-) address of the device. A double '@@'

escapes this behavior and gets replaced by a single '@' in the URL string.

- *par2:* null
- *par3:* null

VIDEO-VIEW

Launches a VideoLAN video player to show the video stream from a surveillance camera controlled by a M&C. The VIDEO-VIEW ButtonElement may only occur in device screens, the IP address of the camera has to be taken from the address_ parameter of the device.

- *par1:* null
- *par2:* null
- *par3:* null

ROBOT-IMPORT

Launches a Robot Import screen. This screen lets the operator select a Pointing-Robot file from the local PC, the file gets converted to sat-nms format an then transferred to the M&C.

- *par1:* null
- *par2:* null
- *par3:* null

TREE-NAVIGATE

In a tree view UI this button navigates in the tree to the tree path referenced with*par1*. As the tree view UI with the WebUI is still t.b.d., this description should be considered as preliminary.

- *par1:* The tree view path to navigate to.
- *par2:* null
- *par3:* null

UNKNOWN

All ButtonElement definitions in a screen, which have no WebGUI equivalent for their programmed *action* (like "launching a new Java VM') are translated into an UNKNOWN action. The Frontend may show this e.g. as a disabled button with an "n/a" label.

- *par1:* The complete content of the message-id field in the original button definition.
- *par2:* null
- *par3:* null

1.5.37 ChartElement

The ChartElement shows a strip chart of a numeric parameter. The chart element keeps a local history of the received values, advances with a constant speed of 1 pixel / second. The default y-scale is 1/division but may be changed by clicking to the chart with the right mouse button.

By default, the strip chart element lets the y-scale offset follow the displayed value that the recent measurement samples are shown in the diagram. This behavior is optimized for applications where the strip chart shall indicate a 'trend' for the displayed value, using an element height of only 50 pixels or less.

Data Model (extends ScreenElement)

Key	Type	Value
type	String	Always "ChartElement".
xpos	Number	see ScreenElement
ypos	Number	see ScreenElement
width	Number	see ScreenElement
height	Number	see ScreenElement
id	String	The message ID this element listens to. This should point to a numeric parameter to make sense.
label	String	The label shown above the strip chart
font	String	The font to be used for the label
color	String	The color used for the label and the strip chart line.
mode	String	The strip chart's display mode. This mainly controls the scaling behavior of the chart. See Display Modes below.
scale	double	The scale 1/div
minValue	double	The value corresponding the bottom line of the widget
maxValue	double	The value corresponding the top line of the widget
minThreshold	double	Values below this threshold cause the widget to display a fault
maxThreshold	double	Values above this threshold cause the widget to display a fault

Display Modes

The ChartElement uses different display and scaling modes, depending on the display mode defined in *mode*. The table below lists the defined display modes an their behavior.

mode	**Description**

mode	**Description**
FLOATING-AUTO	Does a full autoscale. The y-offset of the strip chart is set that the newest value displayed appers at the middle of the y axis. The chart y-scale is evaluated in a 1-2-5 raster to the finest scale that allows all points in the history to be displayed in the chart area. All scaling parameters are ignored.
FLOATING-FIXED	Evaluates the y-offset like in **FLOATING-AUTO**, but applies a fixed y-scale as defined in *scale*. The *scale* value is 1/div and chart height is assumed as 2 divisions. So, the top line of the chart corresponds to y-offset + *scale*, the bottom line to y-offset - *scale*.
FIXED	Sets a fixed y-range from the *minValue* / *maxValue* parameters.
FIXED-THRESHOLD	Like **FIXED**, but also checks every new value against the *minThreshold* / *maxThreshold* limits. If outside the limits, the chart background turns to red.
ANTENNA-AZEL	Makes the element to be a AZ/EL x/y diagram instead of the strip chart. The details about this element type will be defined in a later version of the API.

1.5.38 AzElElement

The AzElElement shows the tracking histoy of an antenna controller as a cloud of points in a elevation over azimuth coorsinate system. Actually only SatService antenna conrollers provide the tracking history data for this screen element.

Data Model (extends ScreenElement)

Key	**Type**	**Value**
type	String	Always "AzElElement".
xpos	Number	see ScreenElement
ypos	Number	see ScreenElement
width	Number	see ScreenElement
height	Number	see ScreenElement
id	String	The device name (prepended by the M&C name) of the antenna controller for which the AzElElement shall be shown.
label	String	The label shown above the chart
font	String	The font to be used for the label

1.5.39 DeviceElement

The DeviceElement represents a device in the M&C user interface. It displays the status of the device by it's color/shape and gives access to the device window for this particular device by a double mouse click. The right mouse button launches a context menu with common operations for the device.

Data Model (extends ScreenElement)

Key	Type	Value
type	String	Always "DeviceElement".
xpos	Number	see ScreenElement
ypos	Number	see ScreenElement
width	Number	see ScreenElement
height	Number	see ScreenElement
id	String	The device ID this element represents in the UI. The device ID consists of the M&C name followed by a dot and the device name.
privilege	Number	The privilege level required to show details about the device and to change the device's operating state.
icon	String	The (base-) name of the icon to show for the device. See Icon Selection below.
showName	Boolean	true = show the device name als a label on top of the icon
showIsBackup	Boolean	true = use a different icon variant for this device if it is in 'is backup' state.
signalBackupId	String	A message ID to listen for a parameter from which the 'is backup' state is derived. Ignored unless *showIsBackup* is true.
backupValue	String	The parameter value which must be matched for the 'is backup' state.
font	String	The font to be used for the label.
color	String	The color to be used for the label (#RRGGBB)

Based on the *id* parameter, the DeviceElement subscribes for a number of parameters, the message IDs of these are as shown below:

{id}.fault: The device's summary fault state. May be one of**OK.**, **WARNING** or **Summary**

FAULT

{id}.faults.99: (Boolean) The device's communication fault

{id}.mode: The operation mode of the device. This is one of**OPERATIONAL**, **FAULT-SUPPRESSED**, **OUT-OF-SERVICE** and **MAINTENANCE**

{id}.info.signal.on: If the value reported via this messageID is**true** or **ON**, the device is considered to be actually ON-AIR. In this case, there is an additional emblem displayed at the upper right corner of the icon signalling this state.

signalBackupId: If the value reported via this messageID equals the *backupValue*, the DeviceElement considers the device to be a backup for something else.

Icon Selection

The DeviceElement shows the state of the device by switching between several variants of of the icon. The data model defines the base version of the icon which is shown when the device is in operating state without a fault. Other states are displayed by changing the icon to a variant which has a status code appended to the base name of the icon:

Icon Name	State	Example
basename.gif	Operational and OK	⬜
basename-**C**.gif	There is no communication to the device	COMFLT!
basename-**F**.gif	The device shows a FAULT	FAULT!
basename-**W**.gif	The device shows a WARNING	🟨
basename-**S**.gif	The device id in fault suppressed mode	FAULT SUPPRESSED
basename-**M**.gif	The device is in maintenance mode	MAINTENANCE
basename-**O**.gif	The device is out of service	OUT OF SERVICE
basename-**B**.gif	The device is OK and used as backup	🟦

In the sat-nms software there exist several sets of these icons, the is extensible by the customer. Supported image formats / file extensions are *.gif, *.jpg and *.png.

The pseudo code sniplet below describes how the DeviceElements is expected to combine the individual status variables to the name of icon to be shown.

```
if summary-alarm
   if communication-alarm
      use -C icon
   else
      use -F icon
else if summary-warning
   use -W icon
else if mode == MAINTENANCE
   use -M icon
else if mode == OUT-OF-SERVICE
   use -O icon
else if mode == FAULT-SUPPRESSED
   use -S icon
else if is-backup
   use -B icon
else
   use standard icon
```

1.5.40 DisplayElement

The DisplayElement is used to display M&C parameters read-only. It looks much alike the ParameterElement element, but never allows to change the parameter it displays.

Data Model (extends ScreenElement)

Key	Type	Value
type	String	Always "DisplayElement".
xpos	Number	see ScreenElement
ypos	Number	see ScreenElement
width	Number	see ScreenElement
height	Number	see ScreenElement
id	String	The message ID this element listens to
label	String	A label shown above the field / drop down
enableId	String	The enable ID (may be empty)

Key	Type	Value
enableValue	String	The value which must be matched to enable the field
font	String	The font to be used
color	String	The color used for the label (#RRGGBB)
drawFrameless	Boolean	true = display without a frame and with transparent background
variableBackgroundColor	Boolean	true = use a variable background color.
colorId	String	The message ID to listen for the background color
colors	Array	of ColorDefinition objects. Defines how to translate values of the *colorId* parameter to color values.

Remarks

Conditionally Enable: If the *enableId* property is not empty, the ParameterElement subscribes for this additional parameter and disables (disabled means "dimmed" in this case) the widget unless the received parameter value matches the values stated in *enableValue*.

Font Selection: The sat-nms software defines 6 preset fonts named 'small', 'plain', 'bold', 'title', 'huge' and 'typewriter'. The *font* property contains one of these names. The selected font is used for the text in the widget, the label above the entry field always is shown with the 'plain' font, unless *font* reads 'small', in this case the label is shown with the 'small' font as well. For information about fonts and font sizes see section Fonts in the appendix of this document.

Variable Background Color: With the *variableBackgroundColor* parameter set 'true', the DisplayElement listens to the additional parameter stated in *colorId* and sets the background color of the widget according to the value received. The received value is translated to a color (and optional 'bold' printing) through the *colors* array. If the received value does not match any of the values listed in the array, the first entry in the array is used as a default / fallback.

1.5.41 EditButtonElement

The EditButtonElement places an EDIT button to the screen. It appears as a simple button with the label "EDIT" on it. A screen definition containing an EDIT button switches the screen to Edit/Take mode. This means, parameter changes are not sent immediately to the server, but the screen remembers (and marks) all changes until the operator either clicks TAKE to send all changes to the server or clicks EDIT again to revert all changes made. Also see the definition of TakeButtonElement

Data Model (extends ScreenElement)

Key	Type	Value

Key	Type	Value
type	String	Always "EditButtonElement".
xpos	Number	see ScreenElement
ypos	Number	see ScreenElement
width	Number	see ScreenElement
height	Number	see ScreenElement
privilege	Number	The privilege level required to switch to EDIT mode.

1.5.42 FaultElement

The FaultElement shows a fault flag. The faults page of the predefined device windows makes use of this display element.

The software treats device faults in a special way. In the variable list of a device driver the fault flags appear as variables called "faults.00" .. "faults.99". The software automatically produces a variable "config.faults.XX" for each fault flag "faults.XX" the driver defines. This configuration variable controls the priority of the fault.

Actually, FaultElement is not included in the API. If the original screen definition contains instances of this type, they are skipped and not included in the ScreenDefinition reported to the UI.

1.5.43 FrameElement

The FrameElement draws a sunken 3D frame, which is intended to be used to group other elements. The 3D frame's inside area is assumes to be transparent, it does not conceal the screen elements it encloses.

Data Model (extends ScreenElement)

Key	Type	Value
type	String	Always "FrameElement".
xpos	Number	see ScreenElement
ypos	Number	see ScreenElement
width	Number	see ScreenElement
height	Number	see ScreenElement

1.5.44 GaugeElement

The GaugeElement shows a numeric parameter value as a horizontal bar in an entry field like frame. The gauge element is capable to adjust the scale factor for the gauge automatically from the parameter's range definition. Alternatively the scale parameters may set explicitely.

Data Model (extends ScreenElement)

Key	**Type**	**Value**
type	String	Always "GaugeElement".
xpos	Number	see ScreenElement
ypos	Number	see ScreenElement
width	Number	see ScreenElement
height	Number	see ScreenElement
id	String	The message ID this element listens to. This should point to a numeric parameter to make sense.
label	String	The label shown above the strip chart
font	String	The font to be used for the label
color	String	The bar color. If thresholds are defined, this color is used while the value is inside the thresholds.
useMinMax	Boolean	If true, the min/max values for the gauge are defined in *minValue*, maxValue_. If false, the GaugeElement takes the limits from the watches parameter's range definition.
minValue	double	The value corresponding the left edge of the widget
maxValue	double	The value corresponding the right edge of the widget
useThresholds	Boolean	If true, the GaugeElement checks the value against the thresholds defined in *minThreshold*, *maxThreshold* and changes the color of the bar to *belowColor* or *above_Color* accordingly.
minThreshold	double	Values below this threshold cause the widget to display a fault
maxThreshold	double	Values above this threshold cause the widget to display a fault
belowColor	String	The bar color to be used if *useThresholds* is true and the actual value is below *minThreshold*
aboveColor	String	The bar color to be used if *useThresholds* is true and the actual value is above *maxThreshold*

Key	Type	Value
tickMode	String	One of 'NONE', 'TICKS', 'TICKS-LABELS'. 'TICKS' makes the element show ticks below the bar graph to indicate the bar position for the *minValue*, *minThreshold*, *maxThreshold* and *maxValue* values. The ticks are to be drawn outside the rectangle defining the element bounds. 'TICKS-LABELS' adds labels below the tick marks, showing the numeric values of the four limit parameters. The number of precision digits shown is taken from the subscribed value's range definition.

Restrictions for colors with *useThresholds* ###

Actually the colors to be used with *useThresholds* set are not freely definable. This is a restriction of how the sat-nms M&C software stores the screen definition, it uses a fixed set of color combinations when thresholds are enabled. These are the valid combinations:

belowColor	***color***	***aboveColor***	**sat-nms mode**
#ff0000	#00ff00	#ff0000	red - green - red
#ff0000	#ffff00	#00ff00	red - yellow - green
#00ff00	#ffff00	#ff0000	green - yellow - red

Using other color combinations than those from the table above will cause unpredictable results.

Special Behavior With Level-Adjust/Level-Control Devices

Normally the GaugeElement expects a numeric value to display in the bar graph. When used with the 'gauge' or the 'gauge2' parameter of the Level-Adjust/Level-Control devices, the GaugeElement must implement a special behavior:

The gauge/gauge2 parameters contain a comma separated string with a list of numeric and textual values. The GaugeElement must parse this string and update its display from the values parsed. The content of the gauge/gauge2 comma separated list is defined as follows:

Pos.	Name	Remark
0	val	The gauge value (numeric)
1	minLabel	The label to be shown for the min value below the graph (textual). Overwrites the label derived from the *minValue* field described above.

Pos.	Name	Remark
2	min	The value corresponding the left edge of the widget (numeric). Overwrites the value defined in the *minValue* field described above.
3	maxLabel	The label to be shown for the max value below the graph (textual). Overwrites the label derived from the *maxValue* field described above.
4	max	The value corresponding the right edge of the widget (numeric). Overwrites the value defined in the *maxValue* field described above.
5	minTLabel	The label to be shown for the minThreshold value below the graph (textual). Overwrites the label derived from the *minThreshold* field described above.
6	minThreshold	The value lower threshold value (numeric). Overwrites the value defined in the *minThreshold* field described above.
7	maxTLabel	The label to be shown for the maxThreshold value below the graph (textual). Overwrites the label derived from the *maxThreshold* field described above.
8	maxThreshold	The upper threshold value (numeric). Overwrites the value defined in the *maxThreshold* field described above.

In order to maintain compatibility to future level control devices, the GaugeElement should not use the parameter names or the driver type of the device to decide if this special behavior applies. Instead, the GaugeElement should make this decision from the data message it receives:

Contains the message a 9-element comma separated list?

If yes, the element should behave as with *useMinMax*=true and *useThresholds*=true, but with the settings contained in this list instead of the ones contained in the GaugeElement data.

Contains the message a 5-element comma separated list?

If yes, the element should behave as with *useMinMax*=true and *useThresholds*=false, but with the settings contained in this list instead of the ones contained in the GaugeElement data.

In all other cases the element should try to parse the message value to floating point value and display it with the settings given in the GaugeElement data object.

1.5.45 IconElement

The IconElement shows an arbitrary GIF/JPEG picture. If a message ID is defined with the element properties, the image displayed will change with the parameter value addressed by the message identifier.

Data Model (extends ScreenElement)

Key	Type	Value
type	String	Always "IconElement".
xpos	Number	see ScreenElement
ypos	Number	see ScreenElement
width	Number	see ScreenElement
height	Number	see ScreenElement
id	String	the message ID this element listens to
icon	String	the initial icon file name

When displayed the first time, the Iconelement shows the image specified in*icon*. If the message ID *id* is not empty, the IconElement dubscribes for this ID and changes the displayed icon every time a new value is received. To make use of this, the name of the icon should be structured in the following way:

```
base-name.{value-part}.{extension}
```

The value-part gets replaced by the parameter value received. For a boolean value for example, two icons 'red-lamp.false.png' and 'red-lamp.true.png'. The IconElement is configured to show 'red-lamp.false.png' initially. When the parameter addressed by *id* gets updated, the display changes according to the parameter value.

1.5.46 LatchingButtonElement

The LatchingButtonElement works much like the ParameterButtonElement described later in this document, but is specialized to show and control an enumeration parameter which knows exactly two states (e.g. on/off or true/false).

When the operator changes the state of the button by clicking it once, the latching button sends the 'other' parameter value to the device. On the other hand, if some other instance in the system changes the parameter state, the latching button recognizes this and changes the up/down state of the button accordingly.

Like the parameter button, the latching button may be labeled with text or an image. To reflect the actual state, the latching button always is configred with two text string or two image names which are shown according to the actual parameter value.

Data Model (extends ScreenElement)

Key	Type	Value
type	String	Always "LatchingButtonElement".
xpos	Number	see ScreenElement
ypos	Number	see ScreenElement
width	Number	see ScreenElement
height	Number	see ScreenElement
id	String	The message ID this element listens for
privilege	Number	The privilege level required to change the button state / parameter value
labelUp	String	The label shown on the button when not pressed
labelDown	String	The label shown on the button when pressed
iconUp	String	The icon shown on the button when not pressed
iconDown	String	The icon shown on the button when pressed
enableId	String	The enable ID (may be empty)
enableValue	String	The value which must be matched to enable the field
font	String	The font to be used for the button label
color	String	The color used for the label
mustQuery	Boolean	Show a query popup before changing the value
queryText	String	the text to show in the query
valueUp	String	The parameter value to send when the button is going up
valueDown	String	The parameter value to send when the button is going down
variableBackgroundColor	Boolean	true = use a variable background color
colorId	String	The message ID to listen for the background color
colors	Array	of ColorDefinition objects. Defines how to translate values of the *colorId* parameter to color values.

The LatchingButtonElement never uses a button label and an icon at the same time. Either *iconUp* / *iconDown* or *labelUp* / *labelDown* contain empty strings.

The *valueUp* and *valueDown* values are not only sent when the button is pressed or released. They also are used to set the button state when a parameter value is received though *id*: If the value matches *valueDown* the button goes to 'pressed' state, in all other cases the button gets released. Labels or icons change an this moment according to the new state.

1.5.47 LockButtonElement

The LockButtonElement defines a button to lock the operation of a number of device devices that no other user can change device settings. The button is in pressed state if at least one of the devices it is assigned to is locked by some user. An operator only may release the button if he set the lock by himself of if he has a privilege level of 150 or more.

Actually, LockButtonElement is not included in the API. If the original screen definition contains instances of this type, they are skipped and not included in the ScreenDefinition reported to the UI.

1.5.48 MCPElement

The MCPElement embeds the measurement display of a spectrum analyzer device in multi channel power measurement mode into the screen.

Actually, MCPElement is not included in the API. If the original screen definition contains instances of this type, they are skipped and not included in the ScreenDefinition reported to the UI.

1.5.49 ParameterButtonElement

The ParameterButtonElement is a button which sends a parameter value if pressed. A frequently used application for the parameter button is a RF-OFF button which sends a "tx.on=OFF" to a certain device. Beside this, a parameter button also may be programmed to play a parameter setting macro.

Data Model (extends ScreenElement)

Key	Type	Value
type	String	Always "ParameterButtonElement".
xpos	Number	see ScreenElement
ypos	Number	see ScreenElement
width	Number	see ScreenElement
height	Number	see ScreenElement
id	String	The message ID this element sends

Key	Type	Value
privilege	Number	The privilege level required to make a button click send a value
label	String	A label shown on the button
icon	String	An icon shown on the button
enableId	String	The enable ID (may be empty)
enableValue	String	The value which must be matched to enable the field
font	String	The font to be used
color	String	The color used for the label (#RRGGBB)
mustQuery	Boolean	true = show a query popup before changing the value
queryText	String	The text to show in the query. If empty, a default query text shall be used.
parameterValue	String	The parameter value to send
isMacroButton	Boolean	If true, this button starts a macro instead of sending a parameter
variableBackgroundColor	Boolean	true = use a variable background color
colorId	String	The message ID to listen for the background color
colors	Array	of ColorDefinition objects. Defines how to translate values of the *colorId* parameter to color values.

The ParameterButtonElement never uses a button label and an icon at the same time. Either *icon* or *label* is an empty string.

If no *parameterValue* is set, the ParameterButtonElement sends the button label or the icon name as the parameter value when clicked

Macro Buttons

The MacroButton is a special version of the ParameterButtonElement. A *isMacroButton = true* marks a ParameterButtonElement to be a MacroButton. In this case the 'id' field contains the name of the M&C where to play the macro and the macro name in this case, separated by a period. Example `VLC0001.myMacro`

Instead of calling 'poke' when the button is clicked, a POST /api/v1/playmacro/{mnc-name}/{macro-name} call must be made by the front end (Example

`POST /api/v1/playmacro/VLC0001/myMacro` .

1.5.50 ParameterElement

The ParameterElement is the common component to display and edit most types of M&C parameters. Depending on the data type of the parameter (the data type is detected automatically) the parameter elements appears as textual / numeric entry field, choice box or as display field for read only parameters.

Data Model (extends ScreenElement)

Key	Type	Value
type	String	Always "ParameterElement".
xpos	Number	see ScreenElement
ypos	Number	see ScreenElement
width	Number	see ScreenElement
height	Number	see ScreenElement
id	String	The message ID this element listens to
privilege	Number	The privilege level requires to change this parameter
label	String	A label shown above the field / drop down
enableId	String	The enable ID (may be empty)
enableValue	String	The value which must be matched to enable the field
font	String	The font to be used
color	String	The color used for the label (#RRGGBB)
mustQuery	Boolean	true = show a query popup before changing the value
queryText	String	The text to show in the query. If empty, a default query text shall be used.
useSpinButtons	Boolean	true = use spin buttons for numeric values
spinSmallIncrement	String	Small spin increment, empty = use default
spinLargeIncrement	String	Large spin increment, empty = use default
useComboBox	Boolean	true = use a searchable combobox

Key	Type	Value

editableComboBox	Boolean	true = allow free text edit in the box
variableBackgroundColor	Boolean	true = use a variable background color.
colorId	String	The message ID to listen for the background color
colors	Array	of ColorDefinition objects. Defines how to translate values of the *colorId* parameter to color values.

Remarks

Widget Variants: Depending on the type of parameter and on the*height* parameter there are different widget types associated with the ParameterElement. As the parameter type is not known at the time the screen definition is read, the decision what widget type shall be used must be done after the parameter type description has been received from the server.

Parameter Type	***height < 40***	***height >= 40***
Text	single line entry field	multi line entry field, scrollable if text does not fit into the field
Numeric	single line entry field	single line entry field
Enum/Choice	drop down or combo box	scrollable list selection
r/o Text	single line display	multi line display, scrollable if text does not fit into the field
r/o Numeric	single line display	single line display
r/o Enum/Choice	single line display	single line display

Conditionally Enable: If the *enableId* property is not empty, the ParameterElement subscribes for this additional parameter and disables (disabled means "dimmed, not changeable") the widget unless the received parameter value matches the values stated in *enableValue*.

Font Selection: The sat-nms software defines 6 preset fonts named 'small', 'plain', 'bold', 'title', 'huge' and 'typewriter'. The *font* property contains one of these names. The selected font is used for the text in the widget, the label above the entry field always is shown with the 'plain' font, unless *font* reads 'small', in this case the label is shown with the 'small' font as well. For information about fonts and font sizes see section Fonts in the appendix of this document.

Operator Query: If the property *mustQuery* reads 'true', the ParameterElement pops up a 'Do you really want...' dialog before a parameter change is sent to the server. *queryText* contains the question to be shown in this case. In the question text, placeholders may used for two values: Any occurrences of the pattern **$P** get replaced by the parameter name (message ID). Any occurrences of the pattern **$V** get replaced by the new value to set. If*queryText* is empty, the ParameterElement uses a standard text instead.

Spin Buttons: With *useSpinButtons* set 'true', the ParameterElement shows spin buttons with editable numberic values. The spin buttons allow to increment/decrement the value by clicking on them. By default, a spin button click increments or decrements the lowest significant digit shown in the entry field. With the shift key hold down, the effective increment is x10. The above applies if the *spinSmallIncrement* / *spinLargeIncrement* properties are empty. If set, these values override the defaults.

ComboBox Variants: For CHOICE parameter the ParameterElement normally uses a simple drop down box. With selection parameter with a handful of choices this easy to operate. There are however situations where the ParameterElement must handle hundreds of choices (e.g. in satellite channel lists). For this the ParameterElement provides variants of the simple drop down box.

With the *useComboBox* parameter set 'true', the ParameterElement uses a special combo box, which implements some filtering on the displayed choices. The operator may enter some text into the edit field of the combo box and only choices which (partially) contain the the entered text are offered for selection.

With the *editableComboBox* parameter set 'true', the ParameterElement permits to send any entered text as the new parameter value to the server, not only one of the valid choices for this parameter.

Variable Background Color: With the *variableBackgroundColor* parameter set 'true', the ParameterElement listens to the additional parameter stated in *colorId* and sets the background color of the widget according to the value received. The received value is translated to a color (and optional 'bold' printing) through the *colors* array. If the received value does not match any of the values listed in the array, the first entry in the array is used as a default / fallback.

1.5.51 RadioButtonElement

The RadioButtonElement is a component to display and edit CHOICE type M&C parameters as a number of radio buttons. Depending on the height of the element, radio buttons are positioned in a row or in a column (*height*<=40 means position horizontally). The radio buttons are labeled with the choices of the parameter, there are as many radio buttons as choices in the parameter range.

Data Model (extends ScreenElement)

Key	Type	Value
type	String	Always "RadioButtonElement".

Key	Type	Value
xpos	Number	see ScreenElement
ypos	Number	see ScreenElement
width	Number	see ScreenElement
height	Number	see ScreenElement
id	String	The message ID this element listens to
privilege	Number	The privilege level requires to change this parameter
label	String	A label shown above the field / drop down
enableId	String	The enable ID (may be empty)
enableValue	String	The value which must be matched to enable the field
font	String	The font to be used
color	String	The color used for the label (#RRGGBB)
mustQuery	Boolean	true = show a query popup before changing the value
queryText	String	The text to show in the query. If empty, a default query text shall be used.
drawFrame	Boolean	true = draw a gray rectangle around the radio button group.
variableBackgroundColor	Boolean	true = use a variable background color.
colorId	String	The message ID to listen for the background color
colors	Array	of ColorDefinition objects. Defines how to translate values of the *colorId* parameter to color values.

1.5.52 RectElement

The RectElement draws a rectangle with a selectable color. The element may listen to a parameter value and change the color of the rectangle according to the parameter value.

Data Model (extends ScreenElement)

Key	Type	Value

Key	Type	Value
type	String	Always "RectElement".
xpos	Number	see ScreenElement
ypos	Number	see ScreenElement
width	Number	see ScreenElement
height	Number	see ScreenElement
id	String	The message ID this element listens to. Is only relevant if *variableLineColor* reads *true*.
fill	Boolean	true = draw the rectangle filled, not only the outline.
color	String	The color used for the rectangle.
variableLineColor	Boolean	true = set the line/fill color from the value received from the parameter addressed by *id*.
useColorTranslation	Boolean	use the colors[] translation
colors	Array	of ColorDefinition objects. Defines how to translate values of the *id* parameter to color values if *useColorTranslation* reads *true*.

Variable Line Color

With *variableLineColor=true* the RectElement will change its line/fill color depending on the value of the parameter addressed by *id*. This way the rectangle can be used as a status indicator. The way the RectElement interprets the parameter value depends on the value of *useColorTranslation*.

If *useColorTranslation=false*, the RectElement tries to decode the line color directly from the parameter value. First it does the following tests on the parameter value (not case sensitive):

Parameter Value	Line Color And Thickness
ends with **BOLD**	The line is drawn thicker than normal
starts with **BLACK**	The line is drawn with this color
starts with **WHITE**	The line is drawn with this color

Parameter Value	**Line Color And Thickness**
starts with **RED**	The line is drawn with this color
starts with **BLUE**	The line is drawn with this color
starts with **GREEN**	The line is drawn with this color
starts with **YELLOW**	The line is drawn with this color
starts with **GRAY**	The line is drawn with this color
contains **TRUE**	If the message ID identifies the parameter as an alarm flag (the ID contains 'fault'), the line is drawn **RED-BOLD**, for other parameters it is drawn **GREEN-BOLD**
contains **FAULT**	The line is drawn **RED-BOLD**
starts with **ON**	The line is drawn **GREEN-BOLD**

If none of the above conditions apply, the RectElement tries to interpret the parameter value as a hex coded color (#RRGGBB). If this also fails, the standard line/fill color defined in *color* is used.

If *useColorTranslation=true*, the RectElement uses the ColorDefinition table in *colors[]* to decode the line's color and thickness.

1.5.53 SpectrumElement

The SpectrumElement embeds the spectrum display of a spectrum analyzer device in the screen. Actually, only the CSM-Spectrum-Analyzer device may be used with the SpectrumElement.

The SpectrumElement subscribes for a variety of parameter of the spectrum analyzer device to display its settings in the diagram area. It also permits to command some aspects of the spectrum analyzer device like the marker position. The spectrum data itself is read by subscribing for the 'trace' parameter of the device. This returns the spectrum data as a SpectrumTrace with each sweep of the spectrum analyzer. A complete description of the function of the spectrum element is given in a separate document.

Data Model (extends ScreenElement)

Key	Type	Value
type	String	Always "SpectrumElement".
xpos	Number	see ScreenElement
ypos	Number	see ScreenElement
width	Number	see ScreenElement
height	Number	see ScreenElement
id	String	The device name of the spectrum analyzer which shall be shown.
privilege	Number	The privilege level requires to change settings of the spectrum analyzer.
label	String	A label shown above the element
font	String	The font to be used for the label
color	String	The color used for the label (#RRGGBB)

1.5.54 SwitchElement

The SwitchElement is a special version of the device element which may be used to visualize the position of a switch in a user interface screen designed as a block diagram. The switch icon has all capabilities of a plain device icon display element. The context menu shown with this right mouse button additionally contains an option to toggle the switch position.

Data Model (extends ScreenElement)

Key	Type	Value
type	String	Always "SwitchElement".
xpos	Number	see ScreenElement
ypos	Number	see ScreenElement
width	Number	see ScreenElement
height	Number	see ScreenElement
id	String	The device ID this element represents in the UI. The device ID consists of the M&C name followed by a dot and the device name.
privilege	Number	The privilege level required to show details about the device and to change the device's operating state.

Key	**Type**	**Value**
icon	String	The (base-) name of the icon to show
enableId	String	The enable ID (may be empty)
enableValue	String	The value which must be matched to enable the field
touchScreenMode	Boolean	true = use a different click response, optimized for touch screen
font	String	The font to be used for the label
color	String	The color used for the label (#RRGGBB)
mustQuery	Boolean	true = show a query popup before changing the value
queryText	String	The text to show in the query

Based on the *id* parameter, the SwitchElement subscribes for a number of parameters, the message IDs of these are as shown below:

{id}.fault: The device's summary fault state. May be one of**OK.**, **WARNING** or **Summary FAULT**

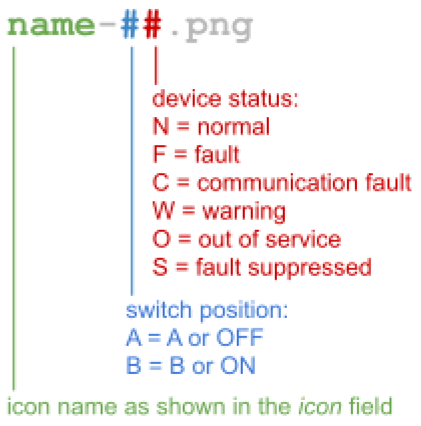
{id}.faults.99: (Boolean) The device's communication fault

{id}.mode: The operation mode of the device. This is one of**OPERATIONAL**, **FAULT-SUPPRESSED**, **OUT-OF-SERVICE** and **MAINTENANCE**

{id}.position: The switch position. May be either 'A' or 'B'. Recognized 'ON'/'OFF' es well and changes the the command sent with a switch toggle accordingly.

Icon Selection

The SwitchElement displays one icon of a given set depending on the switch position and the state of the switch device. For this the SwitchElement appends a '-' character, a 2-character status code and the suffix '.png' to the icon name given in *icon* in order to ge the file name to load / display:



1.5.55 TakeButtonElement

The TakeButtonElement places an TAKE button to the screen. It appears as a simple button with the label "TAKE" on it. The TAKE button complements the EDIT button described above in this document. It sends the changed parameter values to their destination when pressed.

Data Model (extends ScreenElement)

Key	Type	Value
type	String	Always "TakeButtonElement".
xpos	Number	see ScreenElement
ypos	Number	see ScreenElement
width	Number	see ScreenElement
height	Number	see ScreenElement

1.5.56 TargetListElement

The TargetListElement embeds the list of targets of a SatService-ACU-ODM antenna controller in the screen. Targets may be recalled (which moves the antenna to the stored position and sets the tracking parameters associated with this target), saved or deleted.

Data Model (extends ScreenElement)

Key	Type	Value
type	String	Always "TargetListElement".
xpos	Number	see ScreenElement
ypos	Number	see ScreenElement

Key	Type	Value
width	Number	see ScreenElement
height	Number	see ScreenElement
id	String	The device name of the SatService-ACU-ODM. In device screens the *id* has to be interpreted differently, see below.
privilege	Number	The privilege level required to show details about the device and to change the device's operating state.
enableId	String	The enable ID (may be empty)
enableValue	String	The value which must be matched to enable the field

Device Name Indirection

The *id* parameter is interpreted differently depending on the context where the ODM Target List screen element resides:

- When placed in a user defined screen or in the main screen of the application,*id* is the name of the ODM device it shall refer to.
- When placed in the device screen of another device which defines a configuration variable with the antenna controller's device name, *id* is the name of this configuration variable and the name of the ODM device is derived from the content of this variable.
- Finally, when used in the device screen of the ODM device itself,*id* is '@'.

Target List Parsing

The target list element gets its information from a parameter 'target.list' the ODM device provides for this purpose. This variable contains the target list as a one line string with the target definitions appearing at fixed character positions.

Each target definition contains the target name and the azimuth / elevation / polarization angles. The target number is defined implicitly from the position of the target in the complete string. The format of each target definition is as follows:

```
target name (azimuth/elevation/polarization)
```

This string is padded with spaces to 45 characters length. With target numbers starting at 0, you can access a particular target in the string at position *n*45* with a length of 45 characters.

1.5.57 TextElement

The TextElement shows a label (a one line text string) with a chosable font / color.

Data Model (extends ScreenElement)

Key	Type	Value
type	String	Always "TextElement".
xpos	Number	see ScreenElement
ypos	Number	see ScreenElement
width	Number	see ScreenElement
height	Number	see ScreenElement
label	String	The text to be drawn
font	String	The font to be used
color	String	The color used for the label (#RRGGBB)

1.5.58 ThumbnailElement

The ThumbnailElement displays a thumbnail of the video actually processed by a device like an encoder, decoder or gateway. The displayed icon is a static image, fetched directly from the device and updated every couple of seconds.

Data Model (extends ScreenElement)

Key	Type	Value
type	String	Always "ThumbnailElement".
xpos	Number	see ScreenElement
ypos	Number	see ScreenElement
width	Number	see ScreenElement
height	Number	see ScreenElement
id	String	The message ID this element listens to

REMARKS

The Thumbnail Icon element itself must fetch the image to display in regular intervals directly from the device. As a consequence of this the following restrictions apply:

- The device to get the thumbnail images from must support this feature.
- The device to get the thumbnail images from must be accessible in the network from the client's point of view.

With *urlType = PLAIN* , the *imageUrl* is used without modification to fetch the image. The image received from the device must be scaled to fit the screen element size.

With *urlType = ADVANCED* the front end appends some parameters to the URL as shown in the example below. In this case no scaling is done as the software queries the image already in the correct size from the device.

```
http://1.2.3.4/somepath?t=1693050862003&w=320&h=180
```

The example above starts from a *imageUrl = http://1.2.3.4/somepath* definition. With *urlType = ADVANCED* the front end must add the parameters *t*, *w*, and *h* as shown. The value of *t* is the actual time in milliseconds since Jan 1st 1970, 00:00 UTC, *w* and *h* define the dimension of the image in pixels. They usually math the *width* and *height* fields of the data model.

1.5.59 XYChartElement

This element shows the relation of two numeric variables in an X/Y diagram, featuring a 'trace' which shows the recent history of the values with a configurable depth. The update rate, the diagram scaling and much more is configurable with this screen element.

Data Model (extends ScreenElement)

Key	Type	Value
type	String	Always "TextElement".
xpos	Number	see ScreenElement
ypos	Number	see ScreenElement
width	Number	see ScreenElement
height	Number	see ScreenElement
idx	String	The message ID of the parameter to show at the x-axis
idy	String	The message ID of the parameter to show at the y-axis
label	String	The label to be drawn above the diagram
font	String	The font to be used
color	String	The color used for the label (#RRGGBB)
divisions	Number	The number of divisions shown in the diagram for both directions. Typical values are 2 ('hair cross'), 4 or 10.
interval	Number	The update time interval for the display in seconds. 0.1 means to add every 100msec a new value to the display and remove the oldest value from the buffer at the same time.

Key	Type	Value
bufsize	Number	The display maintains a "first in first out" buffer of a size defined with this parameter. The buffer provides a short time memory the display shows as a trace of past values. Typical values are in the range 100-300.
xorigin	Number	The value to be shown at the center of the x axis
yorigin	Number	The value to be shown at the center of the y axis
xscale	Number	The x axis scale in 1/division
yscale	Number	The y axis scale in 1/division
limits	Boolean	If true, the display monitors the actual X/Y values to be within the limits defined below. The limit values are shown as a dark red rectangle. When the actual X/Y values exceed the limits, the diagram background becomes red.
minx	Number	The lower x limit to check. Only meaningful if*limits* is true.
maxx	Number	The upper x limit to check. Only meaningful if*limits* is true.
miny	Number	The lower y limit to check. Only meaningful if*limits* is true.
maxy	Number	The upper y limit to check. Only meaningful if*limits* is true.

1.5.60 ColorDefinition

A ColorDefinition data object defines an entry of a value to color translation table which appears as the *colors* array with several ScreenElement definitions.

Data Model

Key	Type	Value
value	String	The parameter value for which the color shall be used
color	String	The color value (#RRGGBB)
bold	Boolean	true draws the text bold if the value matches

1.5.61 ItemList

The ItemList data object is used by the API at several places to return a list of named items to the front end, e.g. when a list of existing macros or device presets is required.

Data Model

Key	Type	Value
items	Array of String	The item list

1.5.62 Macro

A Macro data object contains a macro of the M&C software, a multi-line text in the sat-nms macro language containing settings / commands to be executed.

Data Model

Key	Type	Value
name	String	The name of this macro
macro	String	The macro itself (multi-line text, lines separated by '\n' character sequences).

1.5.63 PresetValue

Contains one parameter / value definition in a DevicePreset.

Key	Type	Value
name	String	The name of the driver variable this refers to. The driver variable name is the message ID of a parameter with the M&C name and the device name stripped off.
value	String	The value to be assigned to *varName* when the preset get applied to a device.

Values appear in the same way here as in the device preset file. This is formatted along the Range definition for this driver variable:

- FLOAT parameters are formatted with the precision defined in the range, but without a trailing unit string.
- INTEGER parameters appear as simple integer values, also without unit string.
- HEX parameters appear as hexadecimal values without unit string, letters A-F are upper case.
- BOOLEAN parameters are written as TRUE/FALSE.
- CHOICE parameters show the selected choice string.
- TEXT parameters are written without quotes, this means trailing blancs are not stored (this is a general limitation of the sat-nms M&C software)

1.5.64 DevicePreset

A DevicePreset data object contains a device preset of the M&C software.

Data Model

Key	Type	Value
name	String	The name of this preset
driver	String	The name of the device driver this preset is intended to be used with.
comment	String	The comment line appearing as the first line of a device preset file in the sat-nms M&C. Does not include the leading comment character ('#') appearing in the sat-nms preset file.
preset	Array of PresetValue	The value definitions from the device preset and an array of PresetValue definitions.

1.5.65 PresetVars

A PresetVars data object contains a description of an M&C device driver with its preset-storable variable definitions.

Data Model

Key	Type	Value
driver	String	The name of the device driver.
comment	String	The content of the COMMENT statement in the driver (includes the driver version).
vars	Array of Range	The Range definitions of all driver variable which may be stored in a preset. This includes all writable variables which have neither the SETUP nor the NOPRESET property set. The list contains a Range object for each variable with the data type, limits, choices, unit and precision as defined in the device driver file. The *messageId* field of each Range object contains the local name of the variable, not a fully qualified message ID

1.5.66 DeviceVars

A DeviceVars data object contains a description of an M&C device driver with all its variable definitions.

Data Model

Key	Type	Value
driver	String	The name of the device driver.

Key	Type	Value
comment	String	The content of the COMMENT statement in the driver (includes the driver version).
vars	Array of Range	The Range definitions of all driver variables. The list contains a Range object for each variable with the data type, limits, choices, unit and precision as defined in the device driver file. The *messageId* field of each Range object contains the local name of the variable, not a fully qualified message ID

1.5.67 KeyValPair

A KeyValPair defines a key / value pair as used with the Dictionary data model.

Data Model

Key	Type	Value
key	String	The name of the key
val	String	The value assigned to *key*

1.5.68 Dictionary

The Dictionary data model contains a list of key / value pairs.

Data Model

Key	Type	Value
dict	Array	An array of KeyValPair data objects

1.5.69 SatDbSatellite

The SatDbSatellite data record contains the basic properties for one satellite in the *satellites* database table. The fields in the SatDbSatellite data record are 1:1 replicas of the SQL database record. Please refer to the database model description at section Satellite Database how relations between tables are maintained using IDs.

Data Model

Key	Type	Value
satelliteId	Number	unique, primary key, auto generated

Key	**Type**	**Value**
name	String	free text, typically the name as given by satellite operator
operatorId	Number	relation to table *satellite-operators*
noradNumber	Number	unique name (world wide standardized), also relation to table *norad_tle*
intDesignator	String	unique name (world wide standardized)
orbitPosition	Number	satellite orbit position in degrees
inclination	Number	satellite inclination in degrees
alias	String	free text, can be used by customer to add own name
modified	String	timestamp, generated by database at creation / modification time

1.5.70 SatDbBeaconAttenuation

The SatDbBeaconAttenuation data record contains the attenuation value stored in the database for one particular pair of antennaId and beaconId. This is the attenuation to be applied for the given beacon signal when receivbed by this antenn.

Data Model

Key	**Type**	**Value**
attenuationId	Number	unique, primary key, auto generated
beaconId	Number	ID of the beacon this data record belongs to
antennaId	Number	ID of the antenna this beacon belongs to
attenuation	Number	receiver attenuation, floating point, dB
modified	String	timestamp, generated by database at creation / modification time

1.5.71 SatDbBeacon

The SatDbBeacon data record contains the basic properties for one satellite in the *satellite_beacons* database table. The fields in the SatDbBeacon data record are 1:1 replicas of the SQL database record. Please refer to the database model description at section Satellite Database how relations between tables are maintained using IDs.

There may be multiple beacon definitions assigned to one satellite. These beacons share the same *satelliteId*. One of these beacons is the default beacon, this and only this beacon has the *defaultBeacon* flag set *true*.

Data Model

Key	Type	Value
beaconId	Number	unique, primary key, auto generated
satelliteId	Number	ID of the satellite this beacon belongs to
frequency	Number	beacon frequency, floating point, MHz
polarization	String	this is an enumeration, may be one of "H", "V", "RHCP" or "LHCP"
defaultBeacon	Boolean	true means "this is the default beacon for this satellite"
comment	String	free text describing the purpose of this beacon
modified	String	timestamp, generated by database at creation / modification time

1.5.72 SatDbTcAttenuation

The SatDbTcAttenuation data record contains the attenuation values stored in the database for one particular pair of antennaId and tcId. This is the attenuation to be applied for the given TC signal when transmitted by this antenna. The two attenuation values are applied to the same signal at different stages of the transmit chain, e.g. at the upconverter and the HPA.

Data Model

Key	Type	Value
attenuationId	Number	unique, primary key, auto generated
tcId	Number	ID of the TC signal this data record belongs to
antennaId	Number	ID of the antenna this beacon belongs to
attenuation1	Number	transmit attenuation1, floating point, dB
attenuation2	Number	transmit attenuation2, floating point, dB
modified	String	timestamp, generated by database at creation / modification time

1.5.73 SatDbTc

The SatDbTc data record contains the basic properties for one satellite in the*satellite_tc* database table. The fields in the SatDbTc data record are 1:1 replicas of the SQL database record. Please refer to the database model description at section Satellite Database how relations between tables are maintained using IDs.

There may be multiple telecommand channel definitions assigned to one satellite. These channels share the same *satelliteId*. One of these channels is the default TC channel, this and only this channels has the *defaultTc* flag set *true*.

Data Model

Key	Type	Value
tcId	Number	unique, primary key, auto generated
satelliteId	Number	ID of the satellite this TC channel belongs to
frequency	Number	transmit frequency, floating point, MHz
polarization	String	this is an enumeration, may be one of "H", "V", "RHCP" or "LHCP"
defaultTc	Boolean	true means "this is the default TC channel for this satellite"
comment	String	free text describing the purpose of this channel
modified	String	timestamp, generated by database at creation / modification time

1.5.74 SatDbTmAttenuation

The SatDbTmAttenuation data record contains the attenuation values stored in the database for one particular pair of antennaId and tmId. These are the attenuation values to be applied for the given primary and secondary TM signals when received by this antenna.

Data Model

Key	Type	Value
attenuationId	Number	unique, primary key, auto generated
tmId	Number	ID of the TM signal this data record belongs to
antennaId	Number	ID of the antenna this beacon belongs to
attenuation1	Number	transmit attenuation1, floating point, dB
attenuation2	Number	transmit attenuation2, floating point, dB
modified	String	timestamp, generated by database at creation / modification time

1.5.75 SatDbTm

The SatDbTm data record contains the basic properties for one satellite in the*satellite_tm* database table. The fields in the SatDbBeacon data record are 1:1 replicas of the SQL database record. Please refer to the database model description at section Satellite Database how relations between tables are maintained using IDs.

There may be multiple telemetry channel definitions assigned to one satellite. These channels share the same *satelliteId*. One of these TM channels is the default channel, this and only this channel has the *defaultTm* flag set *true*.

Data Model

Key	Type	Value
tmId	Number	unique, primary key, auto generated
satelliteId	Number	ID of the satellite this beacon belongs to
frequency1	Number	primary receive frequency, floating point, MHz
frequency2	Number	secondary receive frequency, floating point, MHz
polarization1	String	this is an enumeration, may be one of "H", "V", "RHCP" or "LHCP"
polarization2	String	this is an enumeration, may be one of "H", "V", "RHCP" or "LHCP"
defaultTm	Boolean	true means "this is the default telemetry channel for this satellite"
comment	String	free text describing the purpose of this beacon
modified	String	timestamp, generated by database at creation / modification time

1.5.76 SatDbSatOperator

The SatDbSatOperator data record contains the basic properties for one satellite in the *satellite_operators* database table. The fields in the SatDbSatOperator data record are 1:1 replicas of the SQL database record. Please refer to the database model description at section Satellite Database how relations between tables are maintained using IDs.

There is a limited list of satellite operators, each being assigned to multiple satellites. The relation is made by the *operatorId* which is stored with each satellite.

Data Model

Key	Type	Value

Key	Type	Value
operatorId	Number	unique, primary key, auto generated
name	String	free text, company name of satellite operator
contact	String	free text, multiline, line breaks are coded as '\n' character sequences.
modified	String	timestamp, generated by database at creation / modification time

1.5.77 SatDbPosition

The SatDbPosition data record contains the antenna pointing information for one satellite and one antenna in the *satellite_positions* database table. The fields in the SatDbPosition data record are 1:1 replicas of the SQL database record. Please refer to the database model description at section Satellite Database how relations between tables are maintained using IDs.

The *Satellite-Positions* table typically contains *number-of-satellites* x *number-of-antennas* records, as each satellite requires different antenna pointing angles for each particular antenna.

Data Model

Key	Type	Value
positionId	Number	unique, primary key, auto generated
satelliteId	Number	the ID of the satellite this SatDbPointing record refers to (relates to the *satellites* table).
azimuth	Number	antenna azimuth pointing angle in degrees (floting point)
elevation	Number	antenna elevation pointing angle in degrees (floting point)
polarization	Number	antenna polarization pointing angle in degrees (floting point)
satelliteColocation	Array of Number	contains the list of satellites (referenced by their satelliteId) which are seen at the same pointing angles by this particular antenna. See remark below
targetNumber	Number	a target number to be set at some antenna controllers (integer)
antennaId	Number	the ID of the antenna this SatDbPointing record refers to (relates to the *antennas* table).

Key	Type	Value
defaultPosition	Boolean	true means "this is the default position for this antenna"
comment	String	a free text describing the purpose of this database entry
modified	String	timestamp, generated by database at creation / modification time

Remark: Depending on the dish size of an antenna, the antenna "sees" not only the satellite it is pointed to but also other satellites which are at positions in the orbit very near to the first satellite. The *satelliteColocation* list contains the satelliteIds for these satellites.

Actually this list is not maintined by the the backend, the *satelliteColocation* will always read as an empty array.

1.5.78 SatDbAntenna

The SatDbAntenna data record describes one antenna in *antennas* database table. The fields in the SatDbPosition data record are 1:1 replicas of the SQL database record. Please refer to the database model description at section Satellite Database how relations between tables are maintained using IDs.

Data Model

Key	Type	Value
antennaId	Number	the ID of the antenna this SatDbAntenna record refers to.
devicename	String	the name of the sat-nms device. controlling this antenna. The name has then name of the M&C system prepended, e.c. "VLC0001.ANTENNA-1"
alias	String	a free text describing the antenna.
modified	String	timestamp, generated by database at creation / modification time

1.5.79 SatDbTLEData

The SatDbTLEData data record contains the TLE ephemeris data for one satellite. The fields in the SatDbTLEData data record are 1:1 replicas of the SQL database record from the *norad_tle* table. Please refer to the database model description at section Satellite Database how relations between tables are maintained using IDs.

Data Model

Key	Type	Value
noradNumber	Number	the norad number of the satellite this SatDbTLEData record refers to. As the norad number is unique, this field is used as the primary key in the database table.
tleName	String	the name of the satellite as stated in the original TLE parameter set.
tleLine1	String	the first line of TLE data
tleLine2	String	the second line of TLE data
modified	String	timestamp, generated by database at creation / modification time

Remark: The TLE (aka Nasa Two Line Elements) format identifies single parameters by their exact position in the lines 1/2. This requires that the *tleLine1* and *tleLine2* fields are processed exactly as they are, there must no whitespace be added or removed from the start or end of the value, multiple blanc characters must not be compressed to one. This is essential to keep the values readable.

1.5.80 SatDbI11Data

The SatDbI11Data data record contains the Intelsat 11 parameter ephemeris data for one satellite. The fields in the SatDbI11Data data record are 1:1 replicas of the SQL database record from the *intelsat_11parameters* table. Please refer to the database model description at section Satellite Database how relations between tables are maintained using IDs.

Data Model

Key	Type	Value
i11Id	Number	a unique ID of this SatDbI11Data record (auto generated).
satelliteId	Number	the ID of the satellite this SatDbI11Data record refers to.
i11Name	String	the name of the satellite as stated in the original I11 parameter set.
i11Data	String	the 11 ephemeris parameters, floating point, separated by semicolon characters
defaultI11	Boolean	true means "this is the default I11 dataset for this satellite"
comment	String	free text describing the purpose of this dataset
modified	String	timestamp, generated by database at creation / modification time

1.5.81 SatDbTableTracking

The SatDbTableTracking data record contains the table tracking information for one satellite. The fields in the SatDbTableTracking data record are 1:1 replicas of the SQL database record from the *table_tracking* table. Please refer to the database model description at section Satellite Database how relations between tables are maintained using IDs.

Data Model

Key	Type	Value
ttId	Number	a unique ID of this SatDbTableTracking record (auto generated).
satelliteId	Number	the ID of the satellite this SatDbTableTracking record refers to.
antennaId	Number	the ID of the antenna this SatDbTableTracking record refers to.
filename	String	the name of the data file containing the position data of the satellite. Should be treated as a free text entry field in the UI.
defaultTt	Boolean	true means "this is the default table tracking dataset for this satellite"
comment	String	free text describing the purpose of this dataset
modified	String	timestamp, generated by database at creation / modification time

1.5.82 SatDbSatDetails

The SatDbSatDetails data record contains detailed information for one satellite. It is returned by a GET performed on the /api/v1/satdetails endpoint, collecting all information known about the satellite in one single document.

Data Model

Key	Type	Value
satellite	SatDbSatellite	the basic properties of the satellite
operator	SatDbSatOperator	the satellite operator assigned to this satellite
beacons	Array of SatDbBeacon	a list of all beacons defined for this satellite, may contain 0 or more elements.
tcChannels	Array of SatDbTc	a list of all telecommand channels defined for this satellite, may contain 0 or more elements.
tmChannels	Array of SatDbTm	a list of all telemetry channels defined for this satellite, may contain 0 or more elements.

Key	Type	Value
tleData	Array of SatDbTLEData	the TLE ephemeris data defined for this satellite, may contain 0 or 1 elements.
i11Data	Array of SatDbI11Data	the I11 ephemeris data defined for this satellite, may contain 0 or more elements.
tableData	Array of SatDbTableTracking	the table tracking information for this satellite, may contain 0 or more elements.

The SatDbSatDetails data record does not contain any antenna pointing information as this also depends on the antenna used to see this satellite.

1.5.83 SatDbTleImport

The SatDbTleUpload data record is used to upload a file containing a list of TLE definitions to the satellite database. It is used with the /api/v1/tleimport API endpoint.

Data Model

Key	Type	Value
fileName	String	the name of the file which is to be imported. This is used in the backend for logging etc.
replaceAll	Boolean	Controls the behavior of the import process, FALSE merges the database and file content together, overwiting data sets with the same norad number. TRUE ensures that after the import only the TLE datasets from imported file are in the database.
fileContent	String	The content of the file to be imported, BASE64 coded.

1.5.84 SatDbState

The SatDbState data record reports the state of the connection of the backend to the database server. The state may be queried by a GET /api/v1/dbstate API call.

Data Model

Key	Type	Value
connected	Boolean	true if the backend is connected to a DB server
primary	Boolean	true if the backend is connected to the primary DB server
serverAddress	String	the IP address of the DB server the backend is actually connected to

1.5.85 ChannelData

The ChannelData data record holds the information about one satellite channel stored in the satellite channel database. It is used by the /api/v1/satellites/##/channels API calls

Data Model

Key	**Type**	**Value**
channelId	Number	A unique ID of this ChannelData record (auto generated).
satelliteId	Number	The ID of the satellite this ChannelData record refers to.
name	String	The user defined channel name.
fullName	String	The full name of this channel as it appears as a selection in the user interface. The full name is built from the satellite's name, a '-' character and the *name* field in this record. The full name must be unique because the operator selects satellite channels by this name. With POST or PATCH calls to the channel database the backend ignores this field, it automatically generates the *fullName* from the satellite name and *name* .
user	String	The name of the user who lastly edited the channel
programTitle	String	Name of the program transmitted or received with this channel
comment	String	A one line comment describing this satellite channel
rxFrequency	Number	The receive frequency in MHz (3 digits precision)
rxPolarization	String	The receive polarization, one of 'X', 'Y', 'L' or 'R'
txFrequency	Number	The transmit frequency in MHz (3 digits precision)
txPolarization	String	The transmit polarization, one of 'X', 'Y', 'L' or 'R'
dvbMode	String	The DVB mode, one of 'AMCD', 'AMCDL', 'AMCDVBS', 'AMCNBC', 'AMDVBS', 'ATSC', 'AUTO', 'DSNG', 'DSS', 'DVBS', 'DVBS2', 'DVBS2X', 'DVBT', 'DVBT2', 'NS3', 'NS4' or 'TURBO'
nlcMode	String	The NLC mode, one of 'OFF' or 'ON'
symbolRate	Number	The symbol rate in Msym/sec (4 digits precision)

Key	Type	Value
fec	String	The FEC rate, one of 'AUTO', '1/1', '1/16', '1/2', '1/3', '1/32', '1/4', '1/5', '1/8', '2/2', '2/3', '2/5', '2/9', '3/16', '3/2', '3/3', '3/4', '3/5', '4/15', '4/2', '4/3', '4/5', '4/9', '5/16', '5/2', '5/6', '5/9', '6/2', '6/5', '6/7', '7/15', '7/2', '7/8', '7/9', '8/10', '8/15', '8/16', '8/2', '8/9', '9/10', '9/2', '9/20', '10/11', '10/2', '11/12', '11/15', '11/2', '11/20', '11/45', '12/2', '12/24', '13/14', '13/15', '13/18', '13/2', '13/24', '13/30', '13/45', '14/24', '14/45', '15/16', '15/25', '16/25', '17/30', '19/20', '19/30', '21/22', '21/44', '22/45', '23/36', '25/36', '26/45', '28/45', '29/45', '31/45', '32/45', '37/45', '41/5', '43/27', '44/27', '45/180', '45/28', '46/28', '47/26', '48/26', '50/15', '55/56', '60/180', '64/57', '70/140', '70/150', '72/180', '77/90', '80/100', '80/180', '90/180', '90/30', '100/180', '108/180', '114/180', '120/180', '126/112', '126/180', '135/180', '144/180', '150/180', '160/180', '162/180', '188/204', '194/178', '204/188', '208/192', '219/201', '225/205', '380/400', '442/485', '485/422', '510/511' or '510/512'
modulation	String	The modulation type, one of '128APSK', '128QAM', '16APSK', '16PSK', '16QAM', '16QUAM', '256APSK', '256QAM', '32APSK', '32QAM', '64APSK', '64QAM', '8APSK', '8PSK', '8QAM', 'AQPSK', 'AUTO', 'BPSK', 'OQPSK', 'PSK8', 'QAM', 'QAMAUTO', 'QPSK', 'SEQ', 'SOQPSK', 'SQPSK', 'UQPSK', 'VIT', 'VSB16' or 'VSB8'
rolloff	String	The roll off factor, one of '0.02', '0.05', '0.10', '0.15', '0.20', '0.25' or '0.35'
pilots	String	The pilots on/off switch, one of 'OFF' or 'ON'
bitRate	Number	The data rate in Mbit/sec (4 digits precision)
useRate	String	Selects the way the data rate is specified, one of 'SYMBOLRATE' or 'BITRATE'
packetSize	String	The packet size, one of '188' or '204'
frameSize	String	The frame size, one of 'SHORT' or 'NORMAL'
videoRate	Number	The video data rate in Mbit/sec (4 digits precision)
vrAuto	String	The video rate auto switch, one of 'FIXED' or 'AUTO'

Key	**Type**	**Value**
profile	String	The video encoding profile, one of 'MPEG2-MP@ML-420', 'MPEG2-MP@LL-420', 'MPEG2-HP@ML-422', 'MPEG2-HP@LL-422', 'MPEG2-SP@LL-420', 'MPEG2-SP@ML-420', 'MPEG2-MP@H14-420', 'MPEG2-MP@HL-420', 'MPEG2-HP@H14-422', 'MPEG2-HP@HL-422', 'MPEG2-HP@HL-420', 'MPEG2-HP@H14-420', 'MPEG2-HP@ML-420', 'MPEG4-BP@L2-420', 'MPEG4-BP@L3-420', 'MPEG4-BP@L4-420', 'MPEG4-MP@L3-420', 'MPEG4-MP@L4-420', 'MPEG4-MP@L4.2-420', 'MPEG4-HP@L3-420', 'MPEG4-HP@L4-420', 'MPEG4-HP@L4.2-420', 'MPEG4-HP@L3-422', 'MPEG4-HP@L4-422' or 'MPEG4-HP@L4.2-422'
eirp	Number	The transmit EIRP in dBW (2 digits precision)
redEirp	Number	The reduced EIRP for line up in dBW (2 digits precision)
autoLineup	String	Configures automatic line up, one of 'OFF' or 'ON'
audioRate1	String	Audio channel 1 data rate, one of 'DISABLED', '64k', '96k', '128k', '160k', '192k', '224k', '256k', '320k' or '384k"
audioRate2	String	Audio channel 2 data rate, one of 'DISABLED', '64k', '96k', '128k', '160k', '192k', '224k', '256k', '320k' or '384k"
audioRate3	String	Audio channel 3 data rate, one of 'DISABLED', '64k', '96k', '128k', '160k', '192k', '224k', '256k', '320k' or '384k"
audioRate4	String	Audio channel 4 data rate, one of 'DISABLED', '64k', '96k', '128k', '160k', '192k', '224k', '256k', '320k' or '384k"
audioChannels	Number	Number of audio channels, (integer, 1-4)
decoderInput	String	Decoder input, one of 'SAT', 'IP' or 'ASI'
ipType	String	IP adressing type, one of 'UNICAST' or 'MULTICAST'
ipProtocol	String	IP protocol, one of 'UDP', 'RTP', 'TCP' or 'ZIXI'
ipSourceAddress	String	Source IP address in dotted quad notation
ipSourcePort	Number	Source port number (integer, 0-65535)
ipDestinationAddress	String	Destination IP address in dotted quad notation

Key	Type	Value
ipDestinationPort	Number	Destination port number (integer, 0-65535)
ipFec	String	IP FEC selection, one of 'OFF' or 'ON'
ipBufferSize	Number	Buffer size (integer)
encoderPhysInterface	Number	Encoder physical output number (integer, 1-4)
decoderPhysInterface	Number	Decoder physical input number (integer, 1-4)
modified	String	timestamp, generated by database at creation / modification time

1.5.86 StreamKeyData

The StreamKeyData data record holds the information about one BISS or SRT key in the *streamkeys* table of the database . It is used by the /api/v1/streamkeys API calls.

Data Model

Key	Type	Value
keyId	Number	A unique ID of this StreamKeyData record (auto generated).
name	String	The user defined name of this stream key. This name also must be unique because the operator selects the key by this name.
keyType	String	The key type, one of 'BISS-1', 'BISS-E' or 'PASSPHRASE'.
key	String	The key value associated with the id and name in this record. For biss keys this is a string entirely consisting of the hexadeximal characters 0123456789ABCDEF, it is either 12 or 16 characters long. For other key types the key is free text.
modified	String	timestamp, generated by database at creation / modification time

1.5.87 TreeViewNode

The TreeViewNode data record holds the information about one node in the sat-nms tree view and the tree below this node. The complete tree is partially autogenerated and user defined, see below about the standard tree structure used in the sat-nms software.

Data Model

Key	Type	Value

Key	**Type**	**Value**
nodeName	String	The (displayed) name of this node. This may be a M&C name, a subsystem name, a device name or a screen name, depending on the type of this node. See below on details about node types and names.
nodeType	String	One of ROOT, MNCNODE, DEVICELIST, SUBSYSTEM or DEVICE. See below on details about node types and names.
children	Array	An array of other TreeViewNode objects being the children of this node. May be null, this denotes a leaf node.

Standard Tree Structure

The general structure of the tree view ist autogenerated by the software. User defined branches with subsystem definitions are integrated at certain points of the tree.

```
ROOT
 +-- MNC001
 +-- MNC002
     +-- Devices
     +-- SUBSYSTEM-1
     +-- SUBSYSTEM-2
         +-- SUBSUBSYSTEM-1
             +-- DEVICE-1
             +-- DEVICE-2
     +-- DEVICE-3
     +-- DEVICE-4
 +-- MNC003
```

A tree always consists of one ROOT node. The children of this node are the M&C systems managed by the backend (one MNCNODE for each M&C system). Each MNCNODE has at least one child, a DEVICELIST node containing the list of devices managed by this M&C. Up to here all nodes are auto-generated and not c hangeable by the user. Additionally, each MNCNODE may have zero or more SUBSYSTEM nodes with user defined groups of devices or nested subsystems.

Node Types and Names

A TreeViewNode may be one of the types ROOT, MNCNODE, DEVICELIST, SUBSYSTEM or DEVICE. The table below lists the properties of these node types in short form:

type	***auto-generated***	***may contain***	***name***
ROOT	yes	MNCNODE	always 'TreeView'

type	***auto-generated***	***may contain***	***name***
MNCNODE	yes	DEVICELIST, SUBSYSTEM	M&C name
DEVICELIST	yes	DEVICE	always 'Devices'
SUBSYSTEM	no	SUBSYSTEM, DEVICE	subsystem name
DEVICE	within a DEVICELIST	-/- (leaf node)	device name

The **ROOT** node is - as the name suggests - the root of the tree view tree. There is exactly one **ROOT** node in the tree, its children are one or more**MNCNODE** nodes, as many as M&C systems are configured to be managed by the backend. The displayed name of each **MNCNODE** node id the ID / name of this M&C, this is the same as used in message IDs. The list of **MNCNODE** nodes is auto-generated by the backend, derived from its configuration data.

The first child of each **MNCNODE** node always is a **DEVICELIST** node containing the complete list of devices configured on this M&C. This node also is automatically created by the backend, the device list is in 'natural' order, meaning that the devices appear in the same order as they do in the M&C configuration file / screen.

DEVICE nodes are the leaf nodes of the tree. The name of a**DEVICE** node is the name of the device without the name of the M&C it is contained in.

SUBSYSTEM nodes define a group of devices or nested subsystems. They are the only user defined node objects in the tree. The name of a **SUBSYSTEM** node is user defined, but as this name appears in the message ID of the summary fault state of a subsystem, the same restrictions applay for subsystem names as for device names. They may consist of upper case letters 'A' .. 'Z', digits and the '-' - character.

Subsystems

Subsystems defined by a **SUBSYSTEM** node in the tree are managed by the M&C like a virtual device which has a summary fault flag which reads as the maximum fault condition that appears at one of the elements the subsystem contains. The definition of all configured subsystems in a M&C is stored by the M&C itself, not by the backend. Due to the M&C-local architecture of subsystems the following rules apply to them:

The name of a subsystem consists of the same limited set of characters like device names do.

A subsystem may contain devices or other subsystems. These children must be located at the same M&C as their parent subsystem. Subsystems cannot be defined across multiple M&Cs.

Each subsystem distributes a summary fault variable which can be used to display the fault state of the subsystem. The full messgeId of this variable is

```
{M&C-name}.SUBSYSTEM.{subsystem-name}.fault
```

or for nested subsystems

`{M&C-name}.SUBSYSTEM.{subsystem-name}.{sub-subsystem-name}....fault`

These variables - like a .fault value of a device - read one of the strings "OK.", "WARNING" or "Summary FAULT".

1.5.88 DatabaseVersion

The DatabaseVersion reports the actual version (migration_step) of the database together with the version required by the backend and a fault flag indicating that the database version is too old for this backend.

Data Model

Key	Type	Value
reported	Number	The version / migration_step reported by the database. a value of -1 means 'undefined'
required	Number	The version / migration_step required by the backend
fault	Boolean	True of the database version is too old

1.5.89 DebugMessage

The DebugMessage data record is used for the communication with a M&C debug terminal. It transports the lines to be displayed in the terminal via the websocket and also the commands to be sent to the debug terminal using the /api/v1/debug API call the other way round.

Data Model

Key	Type	Value
msg	String	The line to be transferred (without any newline character at the end)
error	Boolean	True if this is an error message issued by the backend rather than a message from the M&C

1.5.90 RedundancyState

The RedundancyState data record carries the actual state of a particular M&C redundancy switching logic. It is returned by a GET /api/v1/redundancy API call.

Data Model

Key	Type	Value

Key	Type	Value
mncName	String	The name of the redundant M&C pair for which this data record reports the state
enabled	Boolean	Reports if this redundancy is enabled (true/false)
state	String	Reports the actual state of the redudancy switching logic, one of 'PRIMARY', 'BACKUP', 'NONE'

1.5.91 RedundancySummary

The RedundancySummary data record carries the actual state of the M&C redundancy switching logic for all redundant M&Cs. It is returned by a GET /api/v1/redundancy API call.

Data Model

Key	Type	Value
redundantMnCs	Array	An array of RedundancyState objects, reporting the states to all redundant M&Cs. The array may be empty if no redundant M&Cs are configured in the backend.

1.5.92 RedundancyCmd

The RedundancyCmd data record carries the command to be executed with a /api/v1/redundancy API call.

Data Model

Key	Type	Value
cmd	String	The command to be executed. Valid commands are: 'ENABLE' = enable redundancy switching in case of a communication fault. 'DISABLE' = disable redundancy switching. 'RESET' = switch back to the primary IP address for the M&C 'FORCE' = force a redundancy switch to the backup IP address for the M&C even if the communication at the primary IP address works fine.

1.5.93 RestartCmd

The RestartCmd data record carries the command to be executed with a/api/v1/restart API call.

Data Model

Key	Type	Value

Key	Type	Value
cmd	String	The command to be executed. The only valid command is "RESTART". The restart is only executed if *cmd* has this value.

1.5.94 ScheduleEvent

The ScheduleEvent data record carries the data of one event in the M&C macro scheduler.

Data Model

Key	Type	Value
eventId	Number	An integer value in the range 1 .. 2147483647 uniquely identifying this event in the schedule. A value of 0 means undefined.
groupId	Number	The event's group identifyer. this is the *eventId* if the master event in the group. Even this master event has its own *eventId* set as *groupId*, a *groupId* of 0 means this event is not grouped.
state	String	One of the following: DONE, UPCOMING or MISSED
repeatMode	String	One of the following: NONE, DAILY, WEEKLY or MONTHLY
enabled	Boolean	*true* enables the execution of the event's macro, *false* inhibits this.
description	String	A description for this event given by the operator.
firstExecution	String	The time of the first execution of the event.
nextExecution	String	The time of the next execution of the event.
lastExecution	String	The date of the last execution of the event if a repeat mode is set. For non-repeating events this reads null. The time stamp should contain a time (e.g. 00:00:00) although the scheduler ignores this time.
macro	String	The macro to be executed. newline characters on the macro thave to written as the character sequence "\n".
dict	Array	An optional array of KeyValPair definitions which may be used to append additional information to the event.

Remarks

- New events shall use the lowest *eventId* above 1000 which is not used in the schedule.
- All time stamps are formatted as 'YYYY-MM-DDTHH:MM:SSZ' ans expressed in UTC.

- In the macro text newline characters are expressed as a "\n" character sequence.
- The fields *groupId* and *dict* are not used by the macro scheduler appliaction. The Web GUI shall not show these fields snd shall not present them for editing.

1.5.95 Schedule

The Schedule data record carries the complete shedule of an M&C macro scheduler. It is used with the /api/v1/schedule API call.

Data Model

Key	Type	Value
events	Array	An array of ScheduleEvent data objects defining the complete schedule.

1.5.96 DeviceDefinition

The DeviceDefinition data record carriers the information about one device in an M&C's device setup.

Key	Type	Value
name	String	The name of the device
driver	String	The device driver

1.5.97 DeviceThreadDefinition

The DeviceThreadDefinition data record carriers the information about one device thread in an M&C's device setup.

Key	Type	Value
port	String	The communication port / interface this thread uses
protocol	String	The protocol this thread uses
idle	Number	The idle time of this thread (msecs)
devices	Array	An array of DeviceDefinition objects containing the devices of this thread

1.5.98 DeviceSetup

The DeviceSetup data record carriers the complete information of an M&C's device setup.

Key	Type	Value
comment	String	A short one line description (e.g. time stamp an user name of the last change)
threads	Array	An array of DeviceThreadDefinition objects containing the device thread definitions of this setup

1.5.99 FRViewProperties

The FRViewProperties data record contains the setting stored for a file recorder viewer.

Key	Type	Value
title	String	The diagram title
traces	Array	An array of FRViewTraceDescription objects containing the trace names and unit strings. The File-Recorder device records four traces of data, so this array always contains four elements. When sent to the backend also four elements must be supplied.
presets	Array	An array of FRViewTracePreset objects containing the scaling presets defined for this file recorder view. This array always contains eight elements, when sent to the backend also eight elements must be supplied. In the Java-UI the presets are identified by position in this array, so the front end should not re-order them.

1.5.100 FRViewTraceDescription

The FRViewTraceDescription data record contains a basic description (label, unit) for one trace within the FRViewProperties data record.

Key	Type	Value
unit	String	The unit string to be displayed at the axis for this trace
name	String	The name of the trace

1.5.101 FRViewPreset

The FRViewPreset data record contains the data stored for one user preset within the FRViewProperties data record. This includes the diagram scaling and some additional information like trace colors etc.

Key	Type	Value

Key	Type	Value
absstart	String	The (absolute) starttime of the diagram, expressed as an ISO 8601 string, example: `2021-03-26T08:02:30Z` , time zone is always UTC.
relstart	Number	The starttime of the diagram, in seconds relative to the newest data record recorded.
tscale	String	The time axis scaling, one of ("15s", "30s", "1m", "3m", "5m", "10m", "30m", "1h", "3h", "6h", "12h", "1d", "2d", "6d", "30d"). The values are per division of the time axis
traces	Array	An array of FRViewTraceScaling objects, each describing the scaling properties for one diagram trace. The backend will always report four traces, even if not all four traces are configured in the File-Recorder device. When sent to the backend, each preset **must** provide four elements in this array to be recognized properly.
name	String	The (optional) name of the preset

1.5.102 FRViewTraceScaling

The FRViewTraceScaling data record contains the scaling information for one diagram trace within the FRViewPreset data record.

Key	Type	Value
yref	Number	The reference value (the value shown at the half diagram height)
yscale	Number	The Y-scale value (y-units / division with 10 divisions diagram height). The values must match a 1/2/5 stepping, they will be read back rounded to this if other values are supplied.
enabled	Boolean	TRUE means this trace shall be shown
color	String	The trace color, expressed as a hexadecimal RGB value preceeded by a '#' character

1.5.103 AcuTarget

The AcuTarget data model contains the basic properties (name, az, el, pol) for one target / satellite stored on an ACU device

Key	Type	Value
targetNo	Number	The target number on the ACU

Key	Type	Value
satelliteName	String	The name of the target
azimuth	Number	The stored azimuth angle
elevation	Number	The stored elevation angle
polarization	Number	The stored polarization angle

1.5.104 AcuTargetList

The AcuTargetList data model contains the target list read from an ACU device, showing all used target memories as an array of AcuTarget objects.

Key	Type	Value
acuName	String	Name of the ACU device
acuTargets	Array	Target list as an array ofAcuTarget objects.

1.5.105 InventoryItem

The InventoryItem data record contains the data stored for one particular item in the inventory database. It is used to report, add or modify the properties of an inventory item thru the API. Beside this, the InventoryItem record is used to specify a filter when getting a list of InventoryItem records. in such a case, all fields which are missing or null are interpreted as a wildcard, specified fields must match to include the item in the list.

Key	Type	Value
itemId	Number	A unique ID of this record (auto generated).
serialNo	String	The serial number of the item / device
vendor	String	The vendor name (for serach filters)
model	String	The modem name (for serach filters)
state	String	The administrative state of the device
comment	String	free text
modified	TimeStamp	automatically created timestamp of last change

1.5.106 InventoryDevice

The InventoryDevice data record contains the device mappings for a InventoryItem. There may be multiple devices mapped to one item in the inventory database.

Key	Type	Value
deviceId	Number	A unique ID of this record (auto generated).
itemId	Number	The ID of the InventoryItem this record refers to
mncName	String	The serial number of the item / device
deviceName	String	The vendor name (for serach filters)
state	String	The administrative state of the device
online	Boolean	true means the device is present
modified	TimeStamp	automatically created timestamp of last change

1.5.107 InventoryLogEntry

The InventoryLogEntry data record contains the data of one entry in the inventory life cycle log. It is used to report, add or modify the properties of an inventory log entry the API. Beside this, the InventoryLogEntry record is used to specify a filter when getting a list of InventoryLogEntry records. in such a case, all fields which are missing or null are interpreted as a wildcard, specified fields must match to include the item in the list.

Key	Type	Value
entryId	Number	A unique ID of this record (auto generated).
itemId	Number	Links to the inventory_items entry this log entry refers to.
mncName	String	The M&C which issued this message or null if the message was entered at the UI
devName	String	The satnms device name which issued this message
message	String	the message text
created	TimeStamp	timestamp, generated by database at creation time

1.5.108 DocumentItem

The DocumentItem data record contains the data of one entry in the backend's document list. It is used to report, add or modify the properties of a stored document thru the API. Beside this, the InventoryLogEntry record is used to specify a filter when getting a list of DocumentItem records. in such a case, all fields which are missing or null are interpreted as a wildcard, specified fields must match to include the item in the list.

Key	Type	Value
documentId	Number	A unique ID of this record (auto generated).
fileName	String	Name of the file. The name must be unique, all files are stored in the same directory.
contentType	String	Content type of the file. must be one of application/pdf, image/png or image/jpg.
fileContent	String	The file content as a BASE64 coded string
driver	String	The satnms driver name this document refers to or null if it does not refer to a driver / device type.
mncName	String	The M&C name this document refers to or null if it does not refer to a M&C.
comment	String	free text
modified	TimeStamp	automatically created timestamp of last change

1.5.109 DocumentList

The DocumentList data record contain a list ofDocumentItem records.

Key	Type	Value
documents	Array	An array ofDocumentItem records.

1.5.110 Thumbnail

The Thumbnail data record contains the data fro a thumbnail image when fetched using the /api/v1/thumbnail API call.

Key	Type	Value
messageId	String	The full message ID of the thumbnail image
image	String	The binary thumbnail image (jpf or png), base64 encoded.

1.5.111 BackendInfo

Key	Type	Value

Key	Type	Value
backendName	String	The current backend name
role	String	Role of current backend
isPrimary	Boolean	Whether the backend is main or not
backendNames	String []	Array of backend names (string) to set the activeBackend name

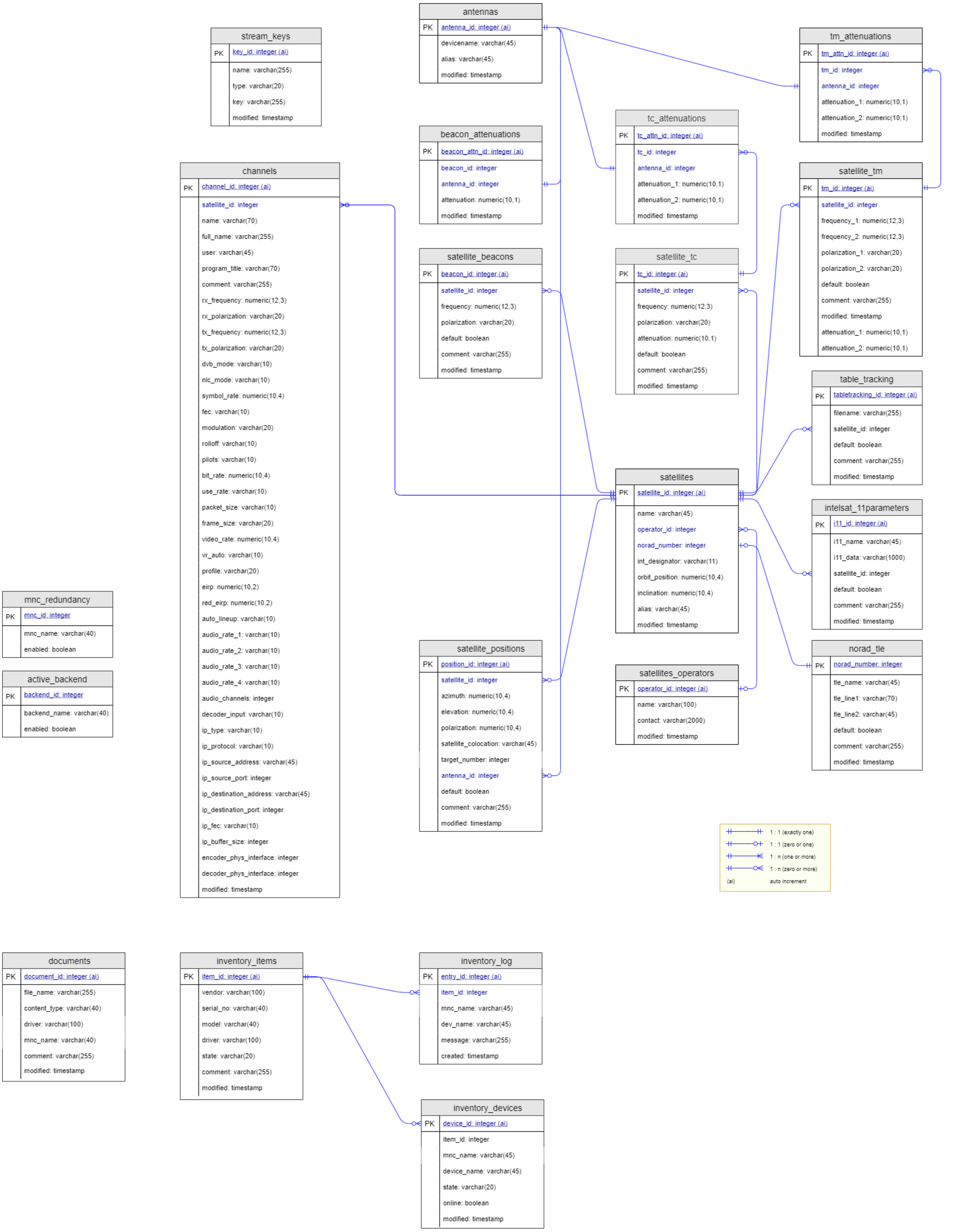
1.5.112 UISettings

The User-Settings data record contains changed settings by the user. Not saved or changed settings will be generated from Backend.

Key	Type	Value
settingsId	Integer	A Unique Id of an setting
key	String	Setting name
value	TEXT	The saved value of the setting
username	String	Username to which the setting belongs
isLocked	Boolean	Whether the setting is locked for user to overwrite it
modified	TimeStamp	automatically created timestamp of last change

1.6 Satellite Database

The satnms4 backend stores all information about satellites, antenna pointing, ephemeris data etc. In a SQL database which is accessible through the backend's API. The following paragraphs describe the table structure of the database and the meaning of each particular table column as an appendix to the API description given above.


stream_keys
PK key_id: integer (ai)
name: varchar(255)
type: varchar(20)
key: varchar(255)
modified: timestamp
antennas
PK antenna_id: integer (ai)
devicename: varchar(45)
alias: varchar(45)
modified: timestamp
tm_attenuations
PK tm_attn_id: integer (ai)
tm_id: integer
antenna_id: integer
attenuation_1: numeric(10,1)
attenuation_2: numeric(10,1)
modified: timestamp
tc_attenuations
PK tc_attn_id: integer (ai)
tc_id: integer
antenna_id: integer
attenuation_1: numeric(10,1)
attenuation_2: numeric(10,1)
modified: timestamp
beacon_attenuations
PK beacon_attn_id: integer (ai)
beacon_id: integer
antenna_id: integer
attenuation: numeric(10,1)
modified: timestamp
channels
PK channel_id: integer (ai)
satellite_id: integer
name: varchar(70)
full_name: varchar(255)
user: varchar(45)
program_title: varchar(70)
comment: varchar(255)
rx_frequency: numeric(12,3)
rx_polarization: varchar(20)
tx_frequency: numeric(12,3)
tx_polarization: varchar(20)
dvb_mode: varchar(10)
nlc_mode: varchar(10)
symbol_rate: numeric(10,4)
fec: varchar(10)
modulation: varchar(20)
rolloff: varchar(10)
pilots: varchar(10)
bit_rate: numeric(10,4)
use_rate: varchar(10)
packet_size: varchar(10)
frame_size: varchar(20)
video_rate: numeric(10,4)
vr_auto: varchar(10)
profile: varchar(20)
eirp: numeric(10,2)
red_eirp: numeric(10,2)
auto_lineup: varchar(10)
audio_rate_1: varchar(10)
audio_rate_2: varchar(10)
audio_rate_3: varchar(10)
audio_rate_4: varchar(10)
audio_channels: integer
decoder_input: varchar(10)
ip_type: varchar(10)
ip_protocol: varchar(10)
ip_source_address: varchar(45)
ip_source_port: integer
ip_destination_address: varchar(45)
ip_destination_port: integer
ip_fec: varchar(10)
ip_buffer_size: integer
encoder_phys_interface: integer
decoder_phys_interface: integer
modified: timestamp
satellite_tm
PK tm_id: integer (ai)
satellite_id: integer
frequency_1: numeric(12,3)
frequency_2: numeric(12,3)
polarization_1: varchar(20)
polarization_2: varchar(20)
default: boolean
comment: varchar(255)
modified: timestamp
attenuation_1: numeric(10,1)
attenuation_2: numeric(10,1)
satellite_beacons
PK beacon_id: integer (ai)
satellite_id: integer
frequency: numeric(12,3)
polarization: varchar(20)
default: boolean
comment: varchar(255)
modified: timestamp
satellite_tc
PK tc_id: integer (ai)
satellite_id: integer
frequency: numeric(12,3)
polarization: varchar(20)
attenuation: numeric(10,1)
default: boolean
comment: varchar(255)
modified: timestamp
table_tracking
PK tabletracking_id: integer (ai)
filename: varchar(255)
satellite_id: integer
default: boolean
comment: varchar(255)
modified: timestamp
satellites
PK satellite_id: integer (ai)
name: varchar(45)
operator_id: integer
norad_number: integer
int_designator: varchar(11)
orbit_position: numeric(10,4)
inclination: numeric(10,4)
alias: varchar(45)
modified: timestamp
intelsat_11parameters
PK i11_id: integer (ai)
i11_name: varchar(45)
i11_data: varchar(1000)
satellite_id: integer
default: boolean
comment: varchar(255)
modified: timestamp
mnc_redundancy
PK mnc_id: integer
mnc_name: varchar(40)
enabled: boolean
active_backend
PK backend_id: integer
backend_name: varchar(40)
enabled: boolean
satellite_positions
PK position_id: integer (ai)
satellite_id: integer
azimuth: numeric(10,4)
elevation: numeric(10,4)
polarization: numeric(10,4)
satellite_colocation: varchar(45)
target_number: integer
antenna_id: integer
default: boolean
comment: varchar(255)
modified: timestamp
satellites_operators
PK operator_id: integer (ai)
name: varchar(100)
contact: varchar(2000)
modified: timestamp
norad_tle
PK norad_number: integer
tle_name: varchar(45)
tle_line1: varchar(70)
tle_line2: varchar(45)
default: boolean
comment: varchar(255)
modified: timestamp
1 : 1 (exactly one)
1 : 1 (zero or one)
1 : n (one or more)
1 : n (zero or more)
(ai) auto increment
documents
PK document_id: integer (ai)
file_name: varchar(255)
content_type: varchar(40)
driver: varchar(100)
mnc_name: varchar(40)
comment: varchar(255)
modified: timestamp
inventory_items
PK item_id: integer (ai)
vendor: varchar(100)
serial_no: varchar(40)
model: varchar(40)
driver: varchar(100)
state: varchar(20)
comment: varchar(255)
modified: timestamp
inventory_log
PK entry_id: integer (ai)
item_id: integer
mnc_name: varchar(45)
dev_name: varchar(45)
message: varchar(255)
created: timestamp
inventory_devices
PK device_id: integer (ai)
item_id: integer
mnc_name: varchar(45)
device_name: varchar(45)
state: varchar(20)
online: boolean
modified: timestamp

1.6.1 Table Satellites (satellites)

All satellites known to this system. Number may vary from a dozen to hundreds.

Field	null	Description
satellite_id	no	unique, primary key, auto increment
name	no	free text, typically the name as given by satellite operator
operator_id	yes	1:1 relation to table *satellite-operators*
norad_number	yes	unique name (world wide standardized), 1:1 relation to table *norad_tle*
int_designator	yes	unique name (world wide standardized)
orbit_position	yes	satellite orbit position in degrees
inclination	yes	satellite inclination in degrees
alias	yes	free text, can be used by customer to add own name
modified	no	automatically created timestamp of last change

1.6.2 Table Satellite Operators (satellites_operators)

All satellites operators known to this system. Usually only a small number (2-10) but not limited to this.

Field	null	Description
operator_id	no	unique, primary key, auto generated
name	no	free text, company name of satellite operator
contact	yes	free text, multiline, hotline phone, e-mail etc.
modified	no	automatically created timestamp of last change

1.6.3 Antennas (antennas)

All antennas known to this system. Usually only a small number (2-10) but not limited to this.

Field	null	Description
antenna_id	no	unique, primary key, auto generated

Field	null	Description
devicename	no	free text, max. 11 characters (at the moment) because it is a device name from the MNC system
alias	yes	free text, can be used by customer to add own name
modified	no	automatically created timestamp of last change

1.6.4 Table Satellite Beacons (satellite_beacons)

All satellites beacons known to this system. Normally every satellite has one or more beacon signals, but it is not necessary to store these information because its only needed if you have to track the satellite. So this table could be completely empty.

Field	null	Description
beacon_id	no	unique, primary key, auto generated
satellite_id	no	ID of the satellite this beacon belongs to
frequency	no	beacon frequency, 3 digits precision (MHz)
polarization	no	integer coded receive polarization
attenuation	yes	an attenuation value to be set at the beacon receiver
default	no	boolean, true meaning this is the default beacon for this satellite
modified	no	automatically created timestamp of last change

1.6.5 Table Satellite TC (satellite_tc)

All satellite telecommand frequencies / channels known to this system. Every satellite may have one or more TC channels, but these are only required if the application deals with satellite TT&C.

Field	null	Description
tc_id	no	unique, primary key, auto generated
satellite_id	no	ID of the satellite this telecommand channel belongs to
frequency	no	TC signal frequency, 3 digits precision (MHz)
polarization	no	integer coded transmit polarization
attenuation	no	an attenuation value to be set at the TC transmitter

Field	null	Description
default	no	boolean, true meaning this is the default beacon for this satellite
modified	no	automatically created timestamp of last change

1.6.6 Table Satellite TM (satellite_tm)

All satellite telemetry frequencies / channels known to this system. Every satellite may have one or more TM channels, but these are only required if the application deals with satellite TT&C.

Field	null	Description
tm_id	no	unique, primary key, auto generated
satellite_id	no	ID of the satellite this telemetry channel belongs to
frequency_1	no	primary receive frequency, 3 digits precision (MHz)
frequency_2	no	secondary receive frequency, 3 digits precision (MHz)
polarization_1	no	integer coded primary receive polarization
polarization_2	no	integer coded secondary receive polarization
attenuation_1	no	attenuation value to be set at the primary telemetry receiver
attenuation_2	no	attenuation value to be set at the secondary telemetry receiver
default	no	boolean, true meaning this is the default beacon for this satellite
modified	no	automatically created timestamp of last change

1.6.7 Table Satellite Positions (satellite_positions)

This tables stores an antenna position (Azimuth/Elevation/Polarization) for a given Satellite and a specific Antenna.

Field	null	Description
position_id	no	unique, primary key, auto generated
satellite_id	no	ID of the satellite this position belongs to
azimuth	yes	antenna azimuth position in degrees
elevation	yes	antenna elevation position in degrees

Field	null	Description
polarization	yes	antenna polarization position in degrees
satellite-colocation	yes	TODO we need to recap how this field will be used, for now its just free text
target_number	yes	a target number for this satellite to command at the antenna controller (device specific)
antenna_id	no	id of antenna, 1:1 relation to *antenna* table
default	no	boolean, true meaning this is the default satellite position
modified	no	automatically created timestamp of last change

Should later provide a dropdown list with the valid antenna (device) names.

1.6.8 Table Table Tracking (table_tracking)

This tables stores "table tracking" files names for satellites. It could have none or one filename per satellite.

Field	null	Description
tabletracking_id	no	unique, primary key, auto generated
filename	no	free text
satellite_id	no	ID of the satellite this tracking table belongs to
default	no	boolean, true meaning this is the default table track
comment	yes	free text
modified	no	automatically created timestamp of last change

1.6.9 Table Intelsat 11 Parameters (intelsat_11parameters)

This tables stores "Intelsat 11 Parameter Sets" for satellites. It could have none or one entry per satellite.

Field	null	Description
i11_id	no	unique, primary key, auto generated
satellite_id	no	ID of the satellite these I11 parameters belong to
i11_name	no	free text, name of I11

Field	null	Description
i11_data	no	free text, (raw) data of I11 (not split into 11 single parameters)
default	no	boolean, true meaning this is the default table track
comment	yes	free text
modified	no	timestamp, generated by database at creation time

1.6.10 Table Norad TLE (norad_tle)

This tables stores "Norad Two Line Elements (TLE) " for satellites. It could have none or one entry per satellite.

Field	null	Description
norad_number	no	unique Norad Number, primary key, defined by the satellite
tle_name	no	free text, name of TLE
tle_line1	no	free text, first line of TLE
tle_line2	no	free text, second line of TLE
default	no	boolean, true meaning this is the default table track
comment	yes	free text
modified	no	timestamp, generated by database at creation time

1.6.11 Table Channels (channels)

This tables stores satellite channel records as described below.

Field	null	Description
channel_id	no	A unique ID of this ChannelData record (auto generated).
satellite_id	no	The ID of the satellite this ChannelData record refers to.
name	no	The user defined channel name.

Field	null	Description
full_name	no	The full name of this channel as it appears as a selection in the user interface. The full name is built from the satellite's name, a '-' character and the *name* field in this record. The full name must be unique because the operator selects satellite channels by this name. With POST or PATCH calls to the channel database the backend ignores this field, it automatically generates the *fullName* from the satellite name and *name* .
user	yes	The name of the user who lastly edited the channel
program_title	yes	Name of the program transmitted or received with this channel
comment	yes	A one line comment describing this satellite channel
rx_frequency	yes	The receive frequency in MHz (3 digits precision)
rx_polarization	yes	The receive polarization, one of 'X', 'Y', 'L' or 'R'
tx_frequency	yes	The transmit frequency in MHz (3 digits precision)
tx_polarization	yes	The transmit polarization, one of 'X', 'Y', 'L' or 'R'
dvb_mode	yes	The DVB mode, one of 'DVBS', 'DVBS2', 'NS3' or 'NS4'
nlc_mode	yes	The NLC mode, one of 'OFF' or 'ON'
symbol_rate	yes	The symbol rate in Msym/sec (4 digits precision)
fec	yes	The FEC rate, one of 'AUTO', '1/2', '1/3', '1/4', '2/3', '3/4', '2/5', '3/5', '4/5', '5/6', '6/7', '7/8', '8/9' or '9/10'
modulation	yes	The modulation type, one of 'BPSK', 'QPSK', 'OQPSK', '8PSK', '16PSK', '16QAM', '64QAM', '256QAM', '16APSK' or '32APSK'
rolloff	yes	The roll off factor, one of '0.02', '0.05', '0.10', '0.15', '0.20', '0.25' or '0.35'
pilots	yes	The pilots on/off switch, one of 'OFF' or 'ON'
bit_rate	yes	The data rate in Mbit/sec (4 digits precision)
use_rate	yes	Selects the way the data rate is specified, one of 'SYMBOLRATE' or 'BITRATE'
packet_size	yes	The packet size, one of '188' or '204'

Field	null	Description
frame_size	yes	The frame size, one of 'SHORT' or 'NORMAL'
video_rate	yes	The video data rate in Mbit/sec (4 digits precision)
vr_auto	yes	The video rate auto switch, one of 'FIXED' or 'AUTO'
profile	yes	The video encoding profile, one of 'MPEG2-MP@ML-420', 'MPEG2-MP@LL-420', 'MPEG2-HP@ML-422', 'MPEG2-HP@LL-422', 'MPEG2-SP@LL-420', 'MPEG2-SP@ML-420', 'MPEG2-MP@H14-420', 'MPEG2-MP@HL-420', 'MPEG2-HP@H14-422', 'MPEG2-HP@HL-422', 'MPEG2-HP@HL-420', 'MPEG2-HP@H14-420', 'MPEG2-HP@ML-420', 'MPEG4-BP@L2-420', 'MPEG4-BP@L3-420', 'MPEG4-BP@L4-420', 'MPEG4-MP@L3-420', 'MPEG4-MP@L4-420', 'MPEG4-MP@L4.2-420', 'MPEG4-HP@L3-420', 'MPEG4-HP@L4-420', 'MPEG4-HP@L4.2-420', 'MPEG4-HP@L3-422', 'MPEG4-HP@L4-422' or 'MPEG4-HP@L4.2-422'
eirp	yes	The transmit EIRP in dBW (2 digits precision)
red_eirp	yes	The reduced EIRP for line up in dBW (2 digits precision)
auto_lineup	yes	Configures automatic line up, one of 'OFF' or 'ON'
audio_rate_1	yes	Audio channel 1 data rate, one of 'DISABLED', '64k', '96k', '128k', '160k', '192k', '224k', '256k', '320k' or '384k"
audio_rate_2	yes	Audio channel 2 data rate, one of 'DISABLED', '64k', '96k', '128k', '160k', '192k', '224k', '256k', '320k' or '384k"
audio_rate_3	yes	Audio channel 3 data rate, one of 'DISABLED', '64k', '96k', '128k', '160k', '192k', '224k', '256k', '320k' or '384k"
audio_rate_4	yes	Audio channel 4 data rate, one of 'DISABLED', '64k', '96k', '128k', '160k', '192k', '224k', '256k', '320k' or '384k"
audio_channels	yes	Number of audio channels, (integer, 1-4)
decoder_input	yes	Decoder input, one of 'SAT', 'IP' or 'ASI'
ip_type	yes	IP addressing type, one of 'UNICAST' or 'MULTICAST'
ip_protocol	yes	IP protocol, one of 'UDP', 'RTP', 'TCP' or 'ZIXI'

Field	null	Description
ip_source_address	yes	Source IP address in dotted quad notation
ip_source_port	yes	Source port number (integer, 0-65535)
ip_destination_address	yes	Destination IP address in dotted quad notation
ip_destination_port	yes	Destination port number (integer, 0-65535)
ip_fec	yes	IP FEC selection, one of 'OFF' or 'ON'
ip_buffer_size	yes	Buffer size (integer)
encoder_phys_interface	yes	Encoder physical output number (integer, 1-4)
decoder_phys_interface	yes	Decoder physical input number (integer, 1-4)
modified	no	automatically created timestamp of last change

1.6.12 Table StreamKeys (stream_keys)

This table stores stream keys (BISS keys, SRT passwords) as described below.

Field	null	Description
key_id	no	A unique ID of this StreamKeyData record (auto generated).
name	no	The user defined name of this stream key. This name also must be unique because the operator selects the key by this name.
type	no	The key type, one of 'BISS-1', 'BISS-E' or 'PASSPHRASE'.
key	no	The key value associated with the id and name in this record. For biss keys this is a string entirely consisting of the hexadeximal characters 0123456789ABCDEF, it is either 12 or 16 characters long. For other key types the key is free text.
modified	no	automatically created timestamp of last change

1.6.13 Table BeaconAttenuations (beacon_attenuations)

This table stores the attenuation values for a particular beacon / antenna combination

Field	null	Description
beacon_attn_id	no	A unique ID of this record (auto generated).
beacon_id	no	The ID of the beacon this attenuation belongs to.

Field	null	Description
antenna_id	no	The ID of the antenna this attenuation belongs to,
attenuation	no	The attenuation value (dB)
modified	no	automatically created timestamp of last change

1.6.14 Table TcAttenuations (tc_attenuations)

This table stores the attenuation values for a particular TC channel / antenna combination

Field	null	Description
tc_attn_id	no	A unique ID of this record (auto generated).
beacon_id	no	The ID of the beacon this attenuation belongs to.
antenna_id	no	The ID of the antenna this attenuation belongs to,
attenuation_1	no	The upconverter attenuation value (dB)
attenuation_2	no	The HPA attenuation value (dB)
modified	no	automatically created timestamp of last change

1.6.15 Table TmAttenuations (tm_attenuations)

This table stores the attenuation values for a particular TM channel / antenna combination

Field	null	Description
tm_attn_id	no	A unique ID of this record (auto generated).
beacon_id	no	The ID of the beacon this attenuation belongs to.
antenna_id	no	The ID of the antenna this attenuation belongs to,
attenuation_1	no	The primary TM signal attenuation value (dB)
attenuation_2	no	The secondary TM signal value (dB)
modified	no	automatically created timestamp of last change

1.6.16 Table InventoryItems (inventory_items)

The table inventory_items stores the properties of all items known to the inventory database.

Field	null	Description
item_id	no	A unique ID of this record (auto generated).
serial_no	yes	The serial number of the item / device
vendor	yes	The vendor name (for search filters)
model	yes	The model name (for search filters)
state	no	The administrative state of the device
comment	yes	free text
modified	no	automatically created timestamp of last change

1.6.17 Table InventoryDevices (inventory_devices)

The table inventory_devices stores the device mappings of all items known to the inventory database.

Field	null	Description
device_id	no	A unique ID of this record (auto generated).
item_no	yes	The ID of the item this device is mapped to
mnc_name	yes	The M&C name
device_name	yes	The device name
state	no	The administrative state of the device
online	yes	Boolean flag indicating if the device is online
modified	no	automatically created timestamp of last change

1.6.18 Table InventoryLog (inventory_log)

The table inventory_log stores the live cycle logs of all items which are registered in the inventory_items table.

Field	null	Description
entry_id	no	A unique ID of this record (auto generated).
item_id	no	Links to the inventory_items entry this log entry refers to.

Field	null	Description
mnc_name	yes	The M&C which issued this message or null if the message was entered at the UI
dev_name	yes	The satnms device name which issued this message
message	no	the message text
created	no	timestamp, generated by database at creation time

1.6.19 Table Documents (documents)

The table documents stores information about the documents (pdf, png, jpg) stored in the backend.

Field	null	Description
document_id	no	A unique ID of this record (auto generated).
file_name	no	Name of the file.
content_type	no	Content type of the file. must be one of application/pdf, image/png or image/jpg.
driver	yes	The satnms driver name this document refers to or null if it does not refer to a driver / device type.
mnc_name	yes	The M&C name this document refers to or null if it does not refer to a M&C.
comment	yes	free text
modified	no	automatically created timestamp of last change

The backend handles three types of documents:

- 'M&C related' documents refer to a particular M&C. They have the 'mnc_name' field set.
- 'Driver related' documents refer to a particular device type / driver. They have the 'driver' field set.
- 'global' documents do not refer to anything

Documents are stored on the backend machine in a common directory tree. Global documents are stored at the root of this tree. For each M&C and for each device driver the backend creates a subdirectory. M&C or driver related documents are stored in one of these subdirectories. File names must be unique within the subdirectory where they are are stored.

1.6.20 Table GlobalSettings (globalSettings)

The global_settings stores the properties of changed settings from default by admins.

Field	null	Description
settings_id	Integer	A Unique Id of an setting
key	no	Setting name as key
value	no	Value of the setting
username	yes	Username to which the setting belongs
isLocked	no	Whether the setting is locked for user to overwrite it
modified	no	timestamp, generated by database at creation time

1.6.21 Table User Settings (userSettings)

The user_settings stores the properties of changed settings from default by user.

Field	null	Description
settings_id	Integer	A Unique Id of an setting
key	no	Setting name as key
value	no	Value of the setting
username	yes	Username to which the setting belongs
isLocked	no	Whether the setting is locked for user to overwrite it
modified	no	timestamp, generated by database at creation time

1.7 Fonts

The satnms software uses a number of predefined fonts to be used in the user interface. The table below shows which font types / sizes match the pixel by pixel position and size values of existing screen definitions.

Name	Font	Size	Style
small	Arial	10px	PLAIN
plain	Arial	12px	PLAIN
bold	Arial	12px	BOLD

Name	Font	Size	Style
title	Arial	16px	BOLD
huge	Arial	28px	PLAIN
typewriter	Courier New	10px	PLAIN