



## DAPNET 2.0 CONCEPT AND INTERFACE DEFINITION

*Thomas Gatzweiler, DL2IC*

*Philipp Thiel, DL6PT*

*Marvin Menzerath*

*Ralf Wilke, DH3WR*

LAST CHANGE: SEPTEMBER 4, 2018

## **Abstract**

This is the concept and interface description of the version 2 of the DAPNET. It's purpose in comparison to the first version released is a more robust clustering and network interaction solution to cope with the special requirements of IP connections over HAMNET which means that all network connections have to be considered with a WAN character resulting in unreliable network connectivity. In terms of consistence of the database, "eventually consistence" is considered to be the most reachable. There are "always right" database nodes inside the so called HAMCLOUD. In case of database conflicts, the version inside the HAMCLOUD cluster is always to be considered right.

# Contents

<b>1</b>	<b>Introduction</b>	<b>7</b>
1.1	Key Features	7
1.2	Historic Background	7
1.3	Concept presentation	7
1.4	Transmitter Software	8
1.4.1	Unipager	8
1.4.2	Backward compatibility to XOS slave protocol	8
1.4.3	DAPNET-Proxy for AX.25 transmitters	8
1.5	DAPNET Network	9
1.5.1	Overview and Concept	9
1.5.2	Used third-party Software	9
1.5.3	HAMCLOUD Description	9
1.5.4	Transmitter Group Handling Concept	9
1.5.5	Rubric Handling Concept	9
1.5.6	Queuing Priority Concept	9
<b>2</b>	<b>DAPNET Network Definition</b>	<b>12</b>
2.1	Cluster Description	12
2.1.1	Real-time Message delivery with RabbitMQ	12
2.1.2	Distributed Database with CouchDB	12
2.1.3	Authentication Concept	12
2.1.4	Integration of new Nodes	12
2.2	Interface Overview and Purpose	12
2.2.1	RabbitMQ Exchange	12
2.2.1.1	dapnet.calls	12
2.2.1.2	dapnet.local_calls	13
2.2.1.3	dapnet.telemetry	13
2.2.2	CouchDB Interface	13
2.2.3	Core REST API	13
2.2.4	Statistic, Status and Telemetry REST API	13
2.2.5	Websocket for real-time updates on configuration, Statistics and Telemetry API	13
2.2.6	MQTT Fanout for third-party consumers	13
2.3	Other Definitions	14
2.3.1	Scheduler	14
2.3.2	User Roles and Permissions	14

<b>3</b>	<b>Internal Programming Workflows</b>	<b>15</b>
3.1	Sent calls	15
3.2	Add, edit, delete User	15
3.3	Add, edit, delete Subscriber	16
3.4	Add, edit, delete Node (tbd)	16
3.5	Add, edit, delete Transmitter	16
3.6	Implementation of Transmitter Groups	16
3.7	Add, edit, delete Rubrics	16
3.8	Add, edit, delete Rubrics content	16
3.9	Add, edit, delete, assign Rubrics to Transmitter/-Groups	16
3.10	Docker integration	16
3.11	Microservices	17
3.11.1	Database Service	17
3.11.2	Call Service	17
3.11.3	Rubric Service	17
3.11.4	Transmitter Service	17
3.11.5	Cluster Service	18
3.11.6	Telemetry Service	18
3.11.7	Database Changes Service	18
3.11.8	Status Service	18
3.11.9	RabbitMQ Auth Service	18
3.11.10	Time and Identification Service	18
3.12	Ports and Loadbalancing Concept	18
3.13	Periodic Tasks (Scheduler)	18
3.14	Plugin Interface	18
3.15	Transmitter Connection	18
3.16	Transmitter connections	19
3.16.1	Authentication of all HTTP-Requests in this context	19
3.17	DAPNET-Proxy	19
<b>4</b>	<b>External Usage Workflows</b>	<b>20</b>
4.1	General Concept of REST and Websocket-Updates	20
4.2	Website and App	20
4.2.1	Authentication	20
4.2.2	Calls	20
4.2.3	Rubrics	20
4.2.4	Rubrics content	20
4.2.5	Transmitters and Telemetry	20
4.2.6	Nodes	20
4.2.7	Users	20
4.2.8	MQTT consumers	20
4.2.9	Scripts and automated Software for DAPNET-Input	20

<b>5</b>	<b>Setup and Installation</b>	<b>21</b>
5.1	Accessible ports from HAMNET	21
5.2	Unipager	21
5.3	DAPNET-Proxy	21
5.4	DAPNET Core	21
5.5	Special issues for Core running in Hamcloud	21
5.5.1	Accessible ports from internet	21
5.5.2	Load balancing and high availability	21
<b>6</b>	<b>Protocol Definitions</b>	<b>22</b>
6.1	Input Data Validation	22
6.1.1	Name	22
6.1.2	Bcrypt Hash	22
6.1.3	Email Address	22
6.1.4	Roles	22
6.1.5	Auth Key	22
6.1.6	Owners	22
6.1.7	Rubric Number	22
6.1.8	Rubric Label	23
6.1.9	Transmitter Groups	23
6.1.10	Rubric Cyclic Transmit Interval	23
6.1.11	Description	23
6.1.12	RIC	23
6.1.13	Function Bits	23
6.2	Microservices API	23
6.2.1	Preamble	23
6.2.2	Filtering and pagination in detailed lists	23
6.2.3	Database Service	24
6.2.3.1	GET /users	24
6.2.3.2	GET /users/<username>	25
6.2.3.3	GET /users/_usernames	25
6.2.3.4	PUT /users - Add new user	26
6.2.3.5	PUT /users - Edit existing user	26
6.2.3.6	DELETE /users/<username>?rev=<rev>	27
6.2.3.7	GET /nodes	28
6.2.3.8	GET /nodes/<nodename>	30
6.2.3.9	GET /nodes/_names	31
6.2.3.10	GET /nodes/_descriptions	31
6.2.3.11	PUT /nodes - Add new node	32
6.2.3.12	PUT /nodes - Edit existing node	33
6.2.3.13	DELETE /nodes/<nodename>?rev=<rev>	33
6.2.3.14	GET /rubrics	34
6.2.3.15	GET /rubrics/_view/byNumber?startkey=<n>&endkey=<m>	35
6.2.3.16	GET /rubrics/_view/byTransmitter?key=<transmittername>	35

6.2.3.17	GET /rubrics/_view/byTransmitterGroup?key=<groupname>	35
6.2.3.18	GET /rubrics/_view/withCyclicTransmit	35
6.2.3.19	GET /rubrics/<rubricname>	36
6.2.3.20	GET /rubrics/_names	36
6.2.3.21	GET /rubrics/_descriptions	37
6.2.3.22	PUT /rubrics - Add new rubric	37
6.2.3.23	PUT /rubrics - Edit existing rubric	38
6.2.3.24	DELETE /rubrics/<rubricname>?rev=<rev>	39
6.2.3.25	GET /subscribers	40
6.2.3.26	GET /subscribers/<subscribername>	41
6.2.3.27	GET /subscribers/_names	42
6.2.3.28	GET /subscribers/_descriptions	42
6.2.3.29	PUT /subscribers - Add new subscriber	43
6.2.3.30	PUT /subscribers - Edit existing subscriber	45
6.2.3.31	DELETE /subscribers/<subscribername>?rev=<rev>	46
6.2.3.32	GET /subscriber_groups	46
6.2.3.33	GET /transmitters/_names	46
6.2.3.34	GET /transmitters/_view/groups	46
6.2.3.35	DELETE /transmitters/<transmittername>?rev=	46
6.2.3.36	PUT /transmitters - Add new transmitter	46
6.2.3.37	PUT /transmitters - Edit existing transmitter	47
6.2.4	Call Service	48
6.2.4.1	GET /calls	48
6.2.4.2	GET /calls?limit=<number>	48
6.2.4.3	GET /calls/_view/byDate	48
6.2.4.4	GET /calls/_view/byIssuer	48
6.2.4.5	GET /calls/_view/byRecipient	48
6.2.4.6	GET /calls/_view/pending	48
6.2.4.7	GET /calls/_view/pending_all	48
6.2.4.8	POST /call	48
6.2.5	Rubric Service	49
6.2.5.1	GET /news	49
6.2.5.2	GET /news/_view/byRubric	49
6.2.5.3	GET /news/_view/byRubric/message_no>	49
6.2.5.4	PUT /news/<rubricname>	49
6.2.5.5	PUT /news/<rubricname>/<message_no>	49
6.2.5.6	DELETE /news/<rubricname>?rev=	49
6.2.5.7	DELETE /news/<rubricname>/<message_no>?rev=	49
6.2.6	Transmitter Service	49
6.2.6.1	GET /transmitters	49
6.2.6.2	GET /transmitter/<transmittername>	49
6.2.6.3	POST /transmitters/_bootstrap	50
6.2.6.4	POST /transmitters/_heartbeat	50

6.2.7	Cluster Service	51
6.2.7.1	POST /cluster/discovery	51
6.2.8	Telemetry Service	51
6.2.8.1	GET /telemetry/transmitters	51
6.2.8.2	GET /telemetry/transmitters/<transmittername>	51
6.2.8.3	GET /telemetry/nodes	51
6.2.8.4	GET /telemetry/nodes/<nodesname>	51
6.2.8.5	WS /telemetry	51
6.2.9	Database Changes Service	51
6.2.9.1	WS /changes	51
6.2.10	Status Service	51
6.2.10.1	GET /status/nodes	51
6.2.10.2	GET /status/node/<nodename>	52
6.2.10.3	GET /status	52
6.2.10.4	GET /status/<service_name>	52
6.2.11	Statistics Service	53
6.2.11.1	GET /statistics	53
6.2.12	Auth Service	54
6.2.12.1	Password hashing	54
6.2.12.2	Role definition	54
6.2.12.3	Permission definition	54
6.2.12.4	Permission naming definition	54
6.2.12.5	Permission matrix	55
6.2.12.6	Auth API definition	57
6.2.12.7	Auth call reference list	59
6.2.13	RabbitMQ Service	61
6.2.13.1	GET /rabbitmq/*	61
6.3	RabbitMQ	61
6.3.1	Transmitters	61
6.3.1.1	dapnet.calls	61
6.3.1.2	dapnet.local_calls	61
6.3.2	Telemetry	61
6.3.3	MQTT API for third-party consumers	61
6.4	Telemetry from Transmitters	63
6.5	Telemetry from Nodes	64
6.6	Statistic, Status and Telemetry REST API	65
6.6.1	Telemetry from Transmitters	65
6.6.2	Telemetry from Nodes	65
6.7	Websocket API	66
6.7.1	Telemetry from Transmitters - Summary of all TX	66
6.7.2	Telemetry from Transmitters - Details of Transmitter	67
6.7.3	Telemetry from Nodes - Summary of all Nodes	68
6.7.4	Telemetry from Transmitters - Details of Node	69

6.7.5	Database Changes . . . . .	70
6.7.5.1	Transmitter related . . . . .	70
6.7.5.2	User related . . . . .	70
6.7.5.3	Rubric related . . . . .	71
6.7.5.4	Rubric content related . . . . .	71
6.7.5.5	Node related . . . . .	72
6.8	CouchDB Documents and Structure . . . . .	72
6.8.1	Users . . . . .	72
6.8.2	Nodes . . . . .	73
6.8.3	Transmitters . . . . .	73
6.8.4	Subscribers . . . . .	74
6.8.5	Rubrics . . . . .	74
6.8.6	Rubric's content . . . . .	75
6.8.7	MQTT services and subscribers . . . . .	75



# Chapter 1

## Introduction

more  
text

### 1.1 Key Features

The version 2 will please the user/operator with the following key features:

- Reliable clustering of Node instances over unreliable WAN connections like HAMNET
- Transmitter telemetry realtime display on Website and App with Websockets
- Use of microservices instead of one big application. Easier to develop, maintain and update
- Load-Balancing and fail-over for user and transmitter interfaces
- Real-time on-air display of transmitters on map
- Third-Party API for Brandmeister, APRS, etc.
- Priority Queuing for transmitters
- Send calls to individual transmitters and/or transmitter groups
- Inflexible concept of transmitter groups is replaced by group tags on transmitters
- Improved Cluster status monitoring

### 1.2 Historic Background

write  
some his-  
tory

### 1.3 Concept presentation

An overview of the DAPNET 2.0 concept is given in Fig. [1.1](#).

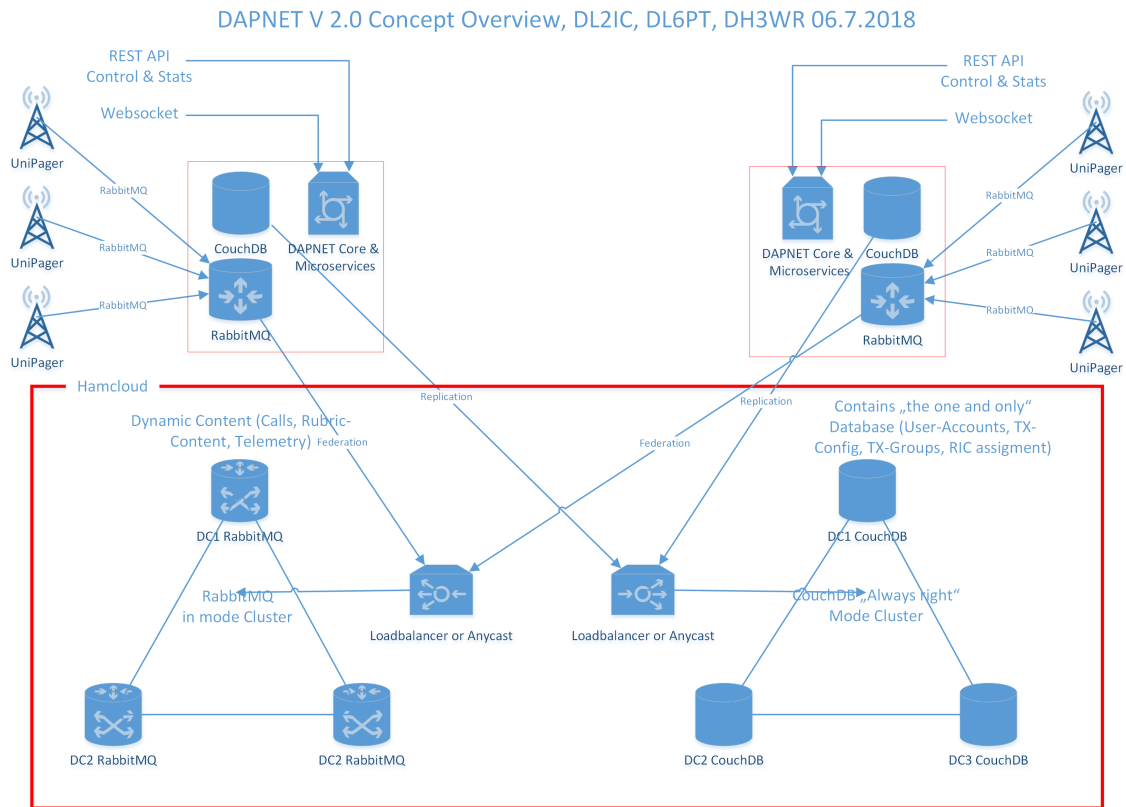


Figure 1.1: Overview of DAPNET Clustering and Network Structure

The details of a single node implementation are shown in Fig. 1.2.

## 1.4 Transmitter Software

### 1.4.1 Unipager

The default software for new transmitters is [Unipager](#). It is developed and maintained by Thomas Gatzweiler, DL2IC. There is a debian based repository available. CI technology is used to assure automated compiling of new versions. Transmitters can be updated all at once if they subscribe to the [SaltStack](#) remote management program.

### 1.4.2 Backward compatibility to XOS slave protocol

The former amateur radio paging transmitters use the XOS slave protocol. It is defined [here](#). There is a NTP like time sync sequence at the beginning of each connection establishment to assure the synchronicity of the transmitters for TDMA. In times of packet-radio, this approach was necessary. Nowadays NTP is used to sync the transmitter clocks; anyway it's still supported.

As DAPNET V2 introduces RabbitMQ and REST interfaces towards transmitters, there is a need for a backward compatibility module, which is also part of the DAPNET V2 package. We hope that after some month, all IP based transmitters have switched to the new interface implementation.

### 1.4.3 DAPNET-Proxy for AX.25 transmitters

For AX.25 only transmitters like [PR430](#), there is still a demand to support the XOS slave protocol over plain AX.25. There is a already working solution to pipe the TCP Data through a lot of intermediate programs towards a AX.25 device. The general data flow is shown in the [DAPNET DokuWiki](#). Figure 1.3 is shown for reference only.

## 1.5 DAPNET Network

### 1.5.1 Overview and Concept

### 1.5.2 Used third-party Software

Used third-party Software is:

- RabbitMQ for Message delivery to transmitters and between nodes
- CouchDB as distributed database backend working on unreliable WAN connections
- NGINX as low resource high performance load balancing server for default Interface endpoints
- Docker for easy deployment and update purposes
- SaltStack for easy distributed updates and maintenance

### 1.5.3 HAMCLOUD Description

The HAMCLOUD is a virtual server combination of server central services on the HAMNET and provide short hop connectivity to deployed service on HAMNET towards the Internet. There are three data centers at Essen, Nürnberg and Aachen, which have high bandwidth interlinks over the DFN. There are address spaces for uni- and anycast services. How this concept is deployed is still tbd.

More information is [here](#) and [here](#).

### 1.5.4 Transmitter Group Handling Concept

In the first Version of DAPNET, transmitters had to be member of one or more logical transmitter groups. Personal calls and rubric content could only be send to a transmitter group, which afterwards sent the data to be member transmitters. Changes in membership required the assigned owner of the group to do so.

In DAPNET V2, there will be just *virtual* transmitter groups by assigning one of more *tags* to a transmitter by its owner himself. Messages can be sent to either a single or group of individual transmitters and/or a single or group of tags. Each transmitter containing the tag will send out the message.

### 1.5.5 Rubric Handling Concept

### 1.5.6 Queuing Priority Concept

A main drawback of the original DAPNET implementation was the lack of priorities in the message queuing on a transmitter. With increasing popularity the load on the transmitters increased and the FIFO working principle led to personal calls being sent out several 10 minutes later than submitted.

To overcome this, a 5 class priority scheme is implemented in DAPNET. Messages to send out are queued

Define if uni- or anycast entry points will exist

Define if all 3 ham-cloud sites will have the same internet incoming ports, and what is the Internet DNS concept

DAPNET V 2.0 Node Details, DL2IC, DL6PT, DH3WR 19.7.2018

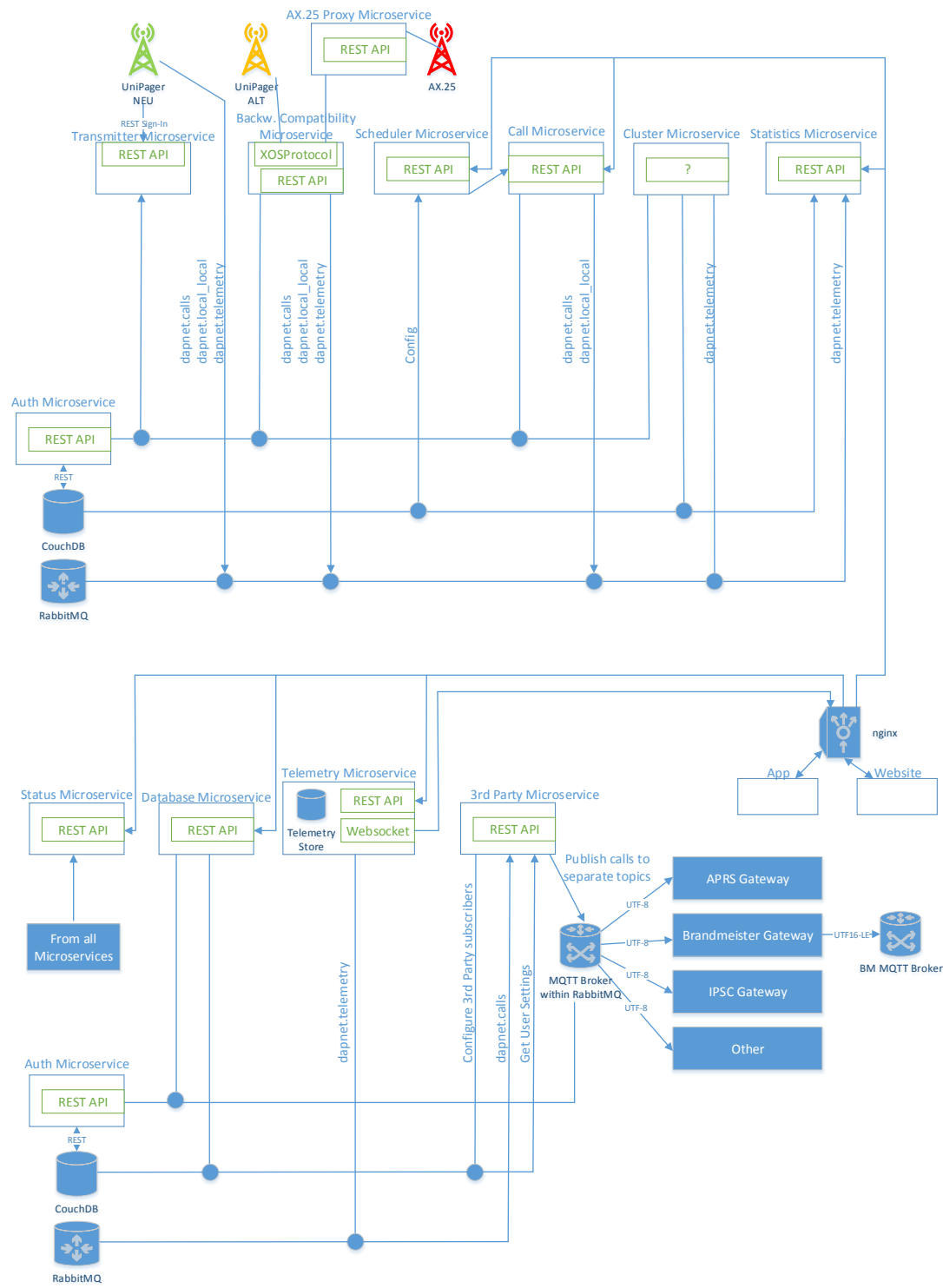


Figure 1.2: Node Details

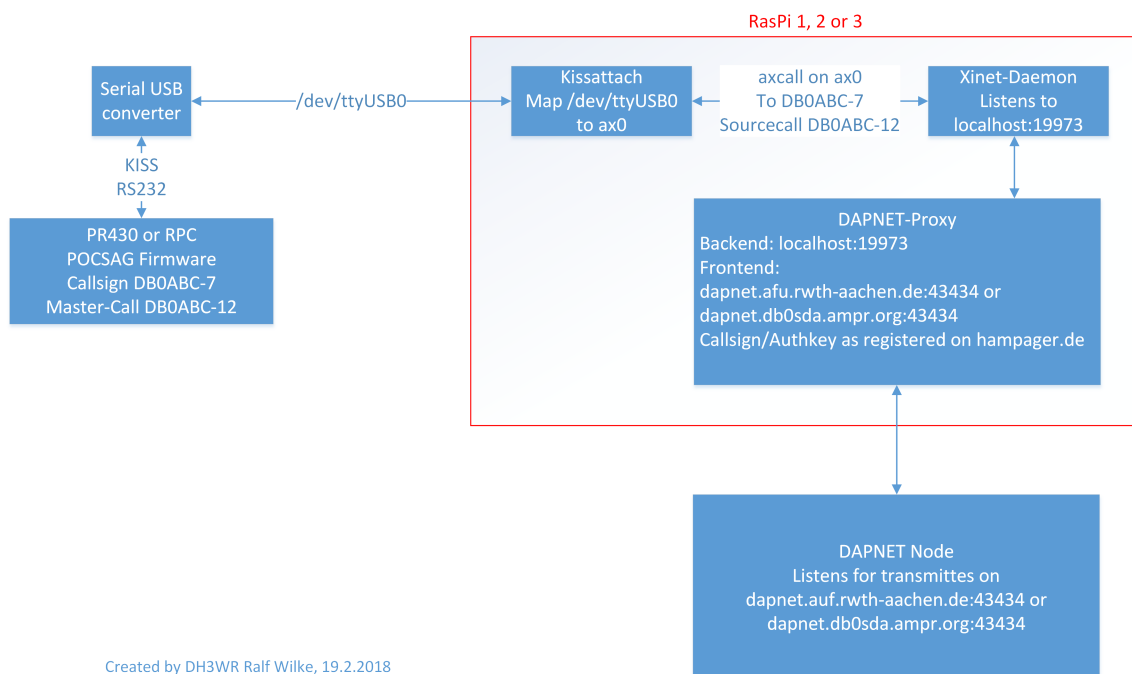


Figure 1.3: Data flow for AX.25 connections from DAPNET

## Chapter 2

# DAPNET Network Definition

### 2.1 Cluster Description

#### 2.1.1 Real-time Message delivery with RabbitMQ

#### 2.1.2 Distributed Database with CouchDB

#### 2.1.3 Authentication Concept

#### 2.1.4 Integration of new Nodes

### 2.2 Interface Overview and Purpose

#### 2.2.1 RabbitMQ Exchange

There are 3 exchanges on each RabbitMQ instance available:

**dapnet.calls** Messages that are distributed to all nodes

**dapnet.local\_calls** Messages coming only from the local node instance

**dapnet.telemetry** Messages containing telemetry from transmitters

Transmitters publish their telemetry data to the **dapnet.telemetry** exchange, while consume the data to be transmitted from a queue that is bound to the **dapnet.calls** and **dapnet.local\_calls** exchanges.

The idea is to distinguish between *local* data coming from the local Core instance and data coming from the DAPNET network. This is necessary, as for example the calls to set the time on the pagers are generated by the local Core and not shall not distributed to other Cores and their connected transmitter to avoid duplicates.

##### 2.2.1.1 dapnet.calls

This federated exchange receives calls from all Core instances. Personal calls are always published to this exchange, as they are unique and only published by the Core that receives the call via the [Core REST API](#). Rubric content is also emitted here. The transmitter to receive the call is defined via the routing key.

### 2.2.1.2 dapnet.local\_calls

The the local Core publishes special calls to this exchange, like the time set calls, the rubric names and repetitions of rubric content for the local connected transmitters.

In short, all calls that are generated by the [Scheduler](#) on a Core instance are published to this exchange. As the scheduler runs on every node, otherwise the calls would be transmitted several times by the same transmitter. This exchange is not federated with other RabbitMQ instances on other Cores.

### 2.2.1.3 dapnet.telemetry

On each Core instance, the [Statistic, Status and Telemetry](#) microservice described in section 2.2.4 is consuming the telemetry of all transmitters. The received data is stored and delivered via the [Core REST API](#) and the [Websocket API](#) in section 2.2.5 to connected websites or apps.

## 2.2.2 CouchDB Interface

The CouchDB interface is a REST interface defined in the CouchDB documentation. All communication with the CouchDB database are done by means of the interface. No user should be able to connect to the CouchDB REST interface, only the Core software components should be able to do so. The local node can access CouchDB with randomly created credentials which are automatically generated on the first startup of the node. For database replication, the other nodes are authenticated by their authentication key in the nodes database.

### 2.2.3 Core REST API

The Core REST API is the main interface for user interactions with the DAPNET network.

### 2.2.4 Statistic, Status and Telemetry REST API

### 2.2.5 Websocket for real-time updates on configuration, Statistics and Telemetry API

### 2.2.6 MQTT Fanout for third-party consumers

In order to allow third-party application to consume the data sent out by DAPNET transmitters in an easy and most generic way, there is an MQTT broker on each Core. As the [RabbitMQ](#) instance already has a plugin to act as an MQTT broker, this solution is chosen.

To dynamically manage the third-party applications attached to DAPNET, there is a [CouchDB](#)-Database containing the existing third-party descriptive names, corresponding MQTT topic names and authentication credentials to be allowed to subscribe to the that specific MQTT topic.

It is a intention to not fan out every content on DAPNET to every third-party application but let the user decide if personal calls directed to her/him will be available on other third-party applications or not. The website will display opt-in checkboxes for each subscriber to enable or disable the message delivery for each third-party application. As we have had some issues in this topic in the past, this seems the best but still generic and dynamic solution.

The fan out consists of the source and destination callsign, the destination RIC and SubRIC and an array of callsign and geographic location of the transmitters, where this specific call is supposed to be sent out by DAPNET transmitters. The type of transmitter is also given. The reason to output also the transmitter and their location is to enable third-party applications to estimate the content's distribution geographic area and take adequate action for their own delivery or further processing. (Example: Regional Rubric content to Regional DMR Group SMS.)

The third-party applications can (if access is granted) only read from the topic. All Core instaces have read/write access to publish the data.

The MQTT topics are kept local on the Core instance and are never distributed between DAPNET-Cores.

## 2.3 Other Definitions

### 2.3.1 Scheduler

### 2.3.2 User Roles and Permissions

There are two types of users: Admins and Non-Admins. Admins are allowed to do everything. Non-Admins are just allowed to edit the entities that they own and send calls.

Make  
overview  
of data  
displayed  
to Non-  
Admin  
users  
from  
CouchDB  
in  
REST-  
Calls (see  
[6.2.](#)



## Chapter 3

# Internal Programming Workflows

### 3.1 Sent calls

### 3.2 Add, edit, delete User

#### Show current users

1. Get current status via `GET /users` on Core URL
2. Handle updates via Websocket

#### Add and Edit User

1. If edit: Get current status via `GET /users/<username>` on Core URL
2. Show edit form and place data
3. On save button event, send `POST /users/<username>` on Core URL

The core will update the CouchDB and generate a RabbitMQ administration message to inform all other nodes. This information is transmitted by the Stats and Websocket Micro-Service to all connected websocket clients to get them updated. This will also happen for the website instance emitting the edit request, so its content is also updated.

#### Delete User

1. Ask "Are you sure?"
2. If yes, send `DELETE /users/<username>` on Core URL

The core will update the CouchDB and generate a RabbitMQ administration message to inform all other nodes. This information is transmitted by the Stats and Websocket Micro-Service to all connected websocket clients to get them updated. This will also happen for the website instance emitting the edit request, so its content is also updated.

### 3.3 Add, edit, delete Subscriber

### 3.4 Add, edit, delete Node (tbd)

### 3.5 Add, edit, delete Transmitter

### 3.6 Implementation of Transmitter Groups

### 3.7 Add, edit, delete Rubrics

#### Show current configuration

1. Get current status via GET /rubrics on Core URL
2. Handle updates via websocket

#### Add and Edit rubrics

1. If edit: Get current status via GET /rubrics/<rubricname> on Core URL
2. Show edit form and place data
3. On save button event, send POST /users/<rubricname> on Core URL

The core will update the CouchDB and generate a RabbitMQ administration message to inform all other nodes. This information is transmitted by the Stats and Websocket Micro-Service to all connected websocket clients to get them updated. This will also happen for the website instance emitting the edit request, so its content is also updated.

#### Delete rubric

1. Ask "Are you sure?"
2. If yes, send DELETE /users/<rubricname> on Core URL

The core will update the CouchDB and generate a RabbitMQ administration message to inform all other nodes. This information is transmitted by the Stats and Websocket Micro-Service to all connected websocket clients to get them updated. This will also happen for the website instance emitting the edit request, so its content is also updated.

### 3.8 Add, edit, delete Rubrics content

### 3.9 Add, edit, delete, assign Rubrics to Transmitter/-Groups

### 3.10 Docker integration

---

DL2IC:  
Docker  
Inte-  
gration  
beschreiben

## 3.11 Microservices

A DAPNET node consists of several isolated microservices with different responsibilities. Each microservice runs in a container and is automatically restarted if it should crash. Some microservices can be started in multiple instances to fully utilize multiple cores. The access to the microservices is proxied by a NGINX webserver which can also provide load balancing and caching.

REST endpoint	Microservice
* /users/* * /nodes/* * /rubrics/* * /subscribers/* * /subscriber_group(s)/* GET /transmitter/grouptags DELETE /transmitter/<transmittername> PUT /transmitter/<transmittername>	Database Service
* /calls/*	Call Service
* /rubrics/content/*	Rubric Service
GET /transmitters GET /transmitters/:id POST /transmitters/bootstrap POST /transmitters/heartbeat	Transmitter Service
POST /cluster/discovery	Cluster Service
GET /telemetry/*	Telemetry Service
WS /telemetry/transmitters WS /telemetry/transmitter/<TxName> WS /telemetry/nodes WS /telemetry/node/<NodeName>	Summary data of all TX Details for TX <TxName> Summary data of all nodes Details for Node <NodeName>
WS /changes	Database Changes Service
GET /status/*	Status Service
GET /statistics	Statistics Service
GET /rabbitmq/*	RabbitMQ Auth Service

### 3.11.1 Database Service

- Proxies calls to the CouchDB database
- Controls access to different database actions
- Removes private/admin only fields from documents

### 3.11.2 Call Service

- Generates and publishes calls to RabbitMQ
- Receives all calls from RabbitMQ
- Maintains a database of all calls

### 3.11.3 Rubric Service

- Publishes rubric content as calls to RabbitMQ
- Periodically publishes rubric names as calls to RabbitMQ

### 3.11.4 Transmitter Service

- Maintains a list of all transmitters and their current status

### 3.11.5 Cluster Service

- Maintains a list of known nodes and their current status
- Manages federation between RabbitMQ queues
- Manages replication between CouchDB databases

### 3.11.6 Telemetry Service

- Maintains the telemetry state of all transmitters
- Forwards telemetry updates via websocket

### 3.11.7 Database Changes Service

- Forwards database changes via websocket

### 3.11.8 Status Service

- Periodically checks all other services and connections

### 3.11.9 RabbitMQ Auth Service

- Provides authentication for RabbitMQ against the CouchDB users database

### 3.11.10 Time and Identification Service

- Sends periodic time and identification messages to RabbitMQ

## 3.12 Ports and Loadbalancing Concept

## 3.13 Periodic Tasks (Scheduler)

## 3.14 Plugin Interface

## 3.15 Transmitter Connection

Transmitter connections consist of two connections to a Node. A REST connection for initial announcement of a new transmitter, heartbeat messages and transmitter configuration and a RabbitMQ connection to receive the data to be transmitted.

The workflow for a transmitter connection is the following:

1. Announce new connecting transmitter via Core REST Interface ([6.2.6.3](#)).
2. Get as response the transmitter configuration or an error message ([6.2.6.3](#)).
3. Initiate RabbitMQ connection to get the data to be transmitted ([6.3.1](#)).

The authentication of the transmitter's REST calls consist of the transmitter name and its AuthKey, which is checked against the value in the CouchDB for this transmitter.

## 3.16 Transmitter connections

If a transmitter wants to connect to DAPNET, the first step is to sign-in and show its presence via the Core REST interface. This interface is also used for transmitter configuration like enabled timeslots and keep-alive polling.

### 3.16.1 Authentication of all HTTP-Requests in this context

All HTTP-requests issued from a transmitter have to send a valid HTTP authentication, which is checked against the CouchDB. It consists of the transmitter name and its AuthKey.

## 3.17 DAPNET-Proxy

Da es sich bei den Anfragen um POST-Requests mit JSON Body handelt, wäre es einfacher da den AuthKey mit dazu zu packen, so wie es auch schon in der Protokoll-Definition umgesetzt ist.

## Chapter 4

# External Usage Workflows

### 4.1 General Concept of REST and Websocket-Updates

### 4.2 Website and App

#### 4.2.1 Authentication

#### 4.2.2 Calls

#### 4.2.3 Rubrics

#### 4.2.4 Rubrics content

#### 4.2.5 Transmitters and Telemetry

#### 4.2.6 Nodes

#### 4.2.7 Users

#### 4.2.8 MQTT consumers

#### 4.2.9 Scripts and automated Software for DAPNET-Input

## Chapter 5

# Setup and Installation

### 5.1 Accessible ports from HAMNET

### 5.2 Unipager

### 5.3 DAPNET-Proxy

### 5.4 DAPNET Core

### 5.5 Special issues for Core running in Hamcloud

#### 5.5.1 Accessible ports from internet

To offer the endpoints to internet-based transmitters and users, the following port have to be accessible:

Type	Port	Application
TCP	80	HTTP Webinterface and Websocket
TCP	443	HTTPS Webinterface and Websocket
TCP	4369	RabbitMQ peer discovery
TCP	5672	RabbitMQ Client connection
TCP	5671	RabbitMQ-TLS Client connection
TCP	25672	RabbitMQ Federation Internode Connection
TCP	1883	MQTT Third Party clients
TCP	8883	MQTT-TLS Third Party clients

#### 5.5.2 Load balancing and high availability

##### Internet-based

To offer load balancing and high availability, the internet-based DNS record *hampager.de* would use DNS round-robin with the static internet IPs of the Hamcloud instances.

##### HAMNET/Hamcloud-based

The Hamcloud instances would offer an anycast IP to for transmitter and user connections. There is a special subnet of 44.0.0.0/8 IPs designated for this anycast approach. Besides, the Hamcloud DAPNET instances will have unicast IPs for administration and their inter-node-synchronization. To connect other nodes besides from the three hamcloud instances, the endpoint to be attached will also be distributed via anycast for maximal fail-over capability.

check if  
TCP/25672  
is correct  
for feder-  
ation

rework  
with con-  
tent of  
discus-  
sion from  
2.8.2018  
on net-  
work

## Chapter 6

# Protocol Definitions

### 6.1 Input Data Validation

#### 6.1.1 Name

First transform to lowercase and remove whitespaces everywhere in the String. Then check against `^[a-z0-9._-]{3,40}$`

#### 6.1.2 Bcrypt Hash

`^\$2[aby]?\$d{1,2}\$[.\A-Za-z0-9]{53}$`

#### 6.1.3 Email Address

`/.+@.+\./i`

The user MMI should do a precheck on the email address. Example for NodeJs: <https://github.com/mailcheck/mailcheck>.

#### 6.1.4 Roles

Get the actual set of allowed roles from the Auth Service at startup and check that the given value is in the returned array. See [6.2.12.6](#) for the API call to the Auth Service. The JSON Value must be an Array with at least one entry. All entries must be a valid role. All entries have to be different, no repetitions of given roles are allow. They have to be filtered out if they exist.

#### 6.1.5 Auth Key

`^[a-zA-Z0-9]{3,40}$`

#### 6.1.6 Owners

Each entry of the array must validate successfully against [6.1.1](#). No checking, if username is existing. The array must contain at least one element.

#### 6.1.7 Rubric Number

The rubric number must be an integer  $\geq 1$  and  $\leq 95$ .



### 6.1.8 Rubric Label

String with only a-zA-Z0-9\_- and max length of TBD.

### 6.1.9 Transmitter Groups

String with only a-zA-Z0-9\_- and max length of TBD.

### 6.1.10 Rubric Cyclic Transmit Interval

Integer value between 0 and 60\*60\*24\*30.

### 6.1.11 Description

The description has to be a string with arbitrary content. Max length 45 characters.

### 6.1.12 RIC

Integer value between 1 and

$2^{21}$

### 6.1.13 Function Bits

Integer value  $\geq 0$  and  $\leq 3$ .

## 6.2 Microservices API

### 6.2.1 Preamble

All HTTP(s) communication should be compress with gzip to reduce network load. That's especially important for the answers to GET-calls of all entity's details.

See [Microservices definition](#).

### 6.2.2 Filtering and pagination in detailed lists

For the endpoints GET /users, GET /nodes, GET /transmitters and GET /rubrics, there are the following GET parameters available for pagination and filtering.

Parameter	Description
?skip=<n>&limit=<m>	Used for pagination, just output <m> entries and skip the first <n> ones. Mapping to CouchDB: transparent
?startkey="dh3wr"\&endkey="d12ic"	Get all entries with an _id between dh3wr and d12ic. Mapping to CouchDB: transparent
?startswith="dh3"	Get all entries where the _id starts with dh3. Used for AJAX based search in tables on website. Mapping to CouchDB: ?startkey="dh3"&endkey="dh3\ufff0"

Table 6.1: Pagination and filtering syntax

**Attention:** The parameter ?include\_docs=true has to be included by the Microservice where it is stated in the detailed description below.

## 6.2.3 Database Service

### 6.2.3.1 GET /users

Item	Description
Description	Returns all users (depending on the filtering in 6.2.2 in JSON format.
Permission	<code>user.read</code>
Mapping to CouchDB	<code>GET /users/_design/users/_view/byId?include_docs=true</code>

Special treatment for user database: The password field has always to be filtered out.

**Auth response: true:**

```
{
  "total_rows": 2,
  "offset": 0,
  "rows": [
    {
      "_id": "dh3wr",
      "_rev": "1-09352254509c9ddf86e80fd83868d557",
      "email": "ralf@secret.com",
      "roles": "user",
      "enabled": true,
      "created_on": "2018-07-08T11:50:02.168325Z",
      "created_by": "dl2ic",
      "changed_on": "2018-07-08T11:50:02.168325Z",
      "changed_by": "dl2ic"
    }, ...
  ]
}
```

*Whitelist:* Transparent, no filtering needed

**Auth response: limited:**

If `_id == <username>` output the corresponding document transparent. All other document's

*Whitelist:* `_id`, `roles`, `enabled`

```
{
  "total_rows": 2,
  "offset": 0,
  "rows": [
    {
      "_id": "dh3wr",
      "_rev": "1-09352254509c9ddf86e80fd83868d557",
      "email": "ralf@secret.com",
      "roles": ["user", "admin"],
      "enabled": true,
      "created_on": "2018-07-08T11:50:02.168325Z",
      "created_by": "dl2ic",
      "changed_on": "2018-07-08T11:50:02.168325Z",
      "changed_by": "dl2ic"
    },
    {
      "_id": "dl2abc",
      "roles": ["user"],
      "enabled": true
    }
  ]
}
```

**Auth response: false:**

Send 403 Forbidden.

### 6.2.3.2 GET /users/<username>

Item	Description
Description	Return details of <username> in JSON format.
Permission	<code>user.read/&lt;username&gt;</code>
Mapping to CouchDB	GET /users/<username>

Special treatment for user database: The password field has always to be filtered out.

**Auth response: true:**

```
{
  "_id": "dh3wr",
  "_rev": "1-09352254509c9ddf86e80fd83868d557",
  "email": "ralf@secret.com",
  "roles": ["user"],
  "enabled": true,
  "created_on": "2018-07-08T11:50:02.168325Z",
  "created_by": "dl2ic",
  "changed_on": "2018-07-08T11:50:02.168325Z",
  "changed_by": "dl2ic"
}
```

**Auth response: limited:**

If `_id == <username>` output the corresponding document transparent.

If not:

*Whitelist:* `_id`, `roles`, `enabled`

**Auth response: false:**

Send 403 Forbidden.

### 6.2.3.3 GET /users/\_usernames

Item	Description
Description	Return just an JSON array of all enabled usernames. Used where selections have to be done on the website.
Permission	<code>user.list</code>
Mapping to CouchDB	GET /users/_design/users/_list/usernames/byId

**Auth response: true:**

```
["dh3wr", "dl2ic"]
```

**Auth response: false or limited:**

Send 403 Forbidden.

#### 6.2.3.4 PUT /users - Add new user

Item	Description
Description	Add a non-existing new user with <code>_id</code> <username>
Permission	<code>user.create</code>
Mapping to CouchDB	PUT /users/<username>
Mandatory fields	<code>_id</code> , <code>password</code> , <code>email</code> , <code>roles</code> , <code>enabled</code>
Field validation	<code>_id</code> : 6.1.1 <code>password</code> : 6.1.2 <code>email</code> : 6.1.3 <code>roles</code> : 6.1.4 <code>enabled</code> : boolean

**Auth response: true:** User to Database-Service: Example content to add user dl6pt

```
{
  "_id" : "dl6pt",
  "password": "$2y$12$lqUeRV094f439Tt7zqrZ0HPfm6YoBzNawWLLIykF3nMip3L6mxLK",
  "email": "ralf@secret.com",
  "roles": ["admin"],
  "enabled": true
}
```

Mapping to CouchDB with adding `created_*` information by database microservice:

```
{
  "_id" : "dl6pt",
  "password": "$2y$12$lqUeRV094f439Tt7zqrZ0HPfm6YoBzNawWLLIykF3nMip3L6mxLK",
  "email": "ralf@secret.com",
  "roles": ["admin"],
  "enabled": true,
  "created_on": "2018-07-08T11:50:02.168325Z",
  "created_by": <username from basic-auth>
}
```

**Auth response: false or limited:**

Send 403 Forbidden.

#### 6.2.3.5 PUT /users - Edit existing user

Edit is differentiated from create by the presence of the `_rev` field.

Item	Description
Description	Edit the existing user <username>.
Permission	<code>user.update/&lt;username&gt;</code>
Mapping to CouchDB	PUT /users/<username>
Mandatory fields	<code>_id</code> , <code>_rev</code> (\rev has to be obtained as in 6.2.3.2)
Optional fields	<code>password</code> , <code>email</code> , <code>roles</code> , <code>enabled</code> (minimum one)
Field validation	<code>_id</code> : Let CouchDB decide, as the entry is already present. <code>_rev</code> : Let CouchDB decide, as the entry is already present. <code>password</code> : 6.1.2 <code>email</code> : 6.1.3 <code>roles</code> : 6.1.4 <code>enabled</code> : boolean

**Auth response: true:** User to Database-Service: Example content to edit user dl6pt setting new password

```
{
  "_id" : "dl6pt",
  "_rev": "1-09352254509c9ddf86e80fd83868d557",
  "password": "$2y$12$lqUeRV094f439Tt7zqrZ0HPfm6YoBzNawWLLIykF3nMip3L6mxLK"
}
```

Mapping to CouchDB with adding `updated_*` information by database microservice:

PUT /users/<username>

```
{
  "_id" : "dl6pt",
  "_rev" : "1-09352254509c9ddf86e80fd83868d557",
  "password" : "$2y$12$1qUeRV094f439Tt7zqrZ0HPfm6YoBzNawWLLIykF3nMip3L6mxLK",
  "updated_on" : "2018-07-08T11:50:02.168325Z",
  "updated_by" : <username from basic-auth>
}
```

**Auth response: false or limited:**

Send 403 Forbidden.

Special treatment if field `roles` is present:

Item	Description
Description	Edit the existing user <username>
Permission	<code>user.change_role/&lt;username&gt;</code>
Mapping to CouchDB	PUT /users/<username>
Mandatory fields	<code>_id</code> , <code>_rev</code> , <code>roles</code> (\rev has to be obtained as in <a href="#">6.2.3.2</a> )
Optional fields	<code>password</code> , <code>email</code> , <code>roles</code> , <code>enabled</code> (minimum one)
Field validation	<code>_id</code> : Let CouchDB decide, as the entry is already present. <code>_rev</code> : Let CouchDB decide, as the entry is already present. <code>password</code> : <a href="#">6.1.2</a> <code>email</code> : <a href="#">6.1.3</a> <code>roles</code> : <a href="#">6.1.4</a> <code>enabled</code> : boolean

**Auth response: true:**

Same behavior as before.

**Auth response: false or limited:**

Send 403 Forbidden.

#### 6.2.3.6 DELETE /users/<username>?rev=<rev>

Item	Description
Description	Delete user <username>
Permission	<code>user.delete/&lt;username&gt;</code>
Mapping to CouchDB	DELETE /users/<username>?rev=1-093...
Mandatory fields	<code>_rev</code> as parameter (\rev has to be obtained as in <a href="#">6.2.3.2</a> )

**Auth response: true:**

Transparent forwarding.

**Auth response: false or limited:**

Send 403 Forbidden.

### 6.2.3.7 GET /nodes

Item	Description
Description	Returns all users (depending on the filtering in 6.2.2 in JSON format.
Permission	<code>node.read</code>
Mapping to CouchDB	<code>GET /users/_design/nodes/_view/byId?include_docs=true</code>

**Auth response: true:**

```
{
  "total_rows": 2,
  "offset": 0,
  "rows": [
    {
      "_id": "db0sda-dc1",
      "_rev": "1-cf7d2abfe193f476888be7108a0f548f",
      "auth_key": "8PL9eJXccQ6X9Yq",
      "coordinates": [34.123456, -23.123456],
      "description": "some words about that node",
      "hamcloud": true,
      "created_on": "2018-07-03T08:00:52.786458Z",
      "created_by": "dh3wr",
      "changed_on": "2018-07-03T08:00:52.786458Z",
      "changed_by": "dh3wr",
      "owners": ["d11abc", "dh3wr", "d12ic"]
    },
    {
      "_id": "db0sda-dc2",
      "_rev": "1-ee070b17db9c3c58658d10fdedad2f48",
      "auth_key": "73mxX4JLttzmVZ2",
      "coordinates": [34.123456, -23.123456],
      "description": "some words about that node",
      "hamcloud": true,
      "created_on": "2018-07-03T08:00:52.786458Z",
      "created_by": "dh3wr",
      "changed_on": "2018-07-03T08:00:52.786458Z",
      "changed_by": "dh3wr",
      "owners": ["d11abc", "dh3wr", "d12ic"]
    }
  ]
}
```

**Auth response: limited:**

The output rows have to be checked, if the asking username is in the `owners` array.

If yes: return the complete row.

If not: return just the whitelisted fields:

*Whitelist:* `_id`, `coordinates`, `description`, `hamcloud`, `owners`

Example (here `dh3wr` is asking with limited permission):

```
{
  "total_rows": 2,
  "offset": 0,
  "rows": [
    {
      "_id": "db0sda-dc1",
      "coordinates": [34.123456, -23.123456],
      "description": "some words about that node",
      "hamcloud": true,
      "owners": ["d11abc", "d12ic"]
    },
    {
      "_id": "db0sda-dc2",
      "_rev": "1-ee070b17db9c3c58658d10fdedad2f48",

```

```
    "auth_key": "73mxX4JLttzmVZ2"  
    "coordinates": [34.123456, -23.123456],  
    "description": "some words about that node",  
    "hamcloud": true,  
    "created_on": "2018-07-03T08:00:52.786458Z",  
    "created_by": "dh3wr",  
    "changed_on": "2018-07-03T08:00:52.786458Z",  
    "changed_by": "dh3wr",  
    "owners": ["dl1abc", "dh3wr", "dl2ic"]  
  }  
]  
}
```

**Auth response:** false:

Send 403 Forbidden.

### 6.2.3.8 GET /nodes/<nodename>

Item	Description
Description	Return details of <nodename> in JSON format.
Permission	<code>node.read/&lt;nodename&gt;</code>
Mapping to CouchDB	<code>GET /users/&lt;nodename&gt;</code>

**Auth response:** `true`:

```
{
  "_id": "db0sda-dc2",
  "_rev": "1-ee070b17db9c3c58658d10fdedad2f48",
  "auth_key": "73mxX4JLttzmVZ2",
  "coordinates": [34.123456, -23.123456],
  "description": "some words about that node",
  "hamcloud": true,
  "created_on": "2018-07-03T08:00:52.786458Z",
  "created_by": "dh3wr",
  "changed_on": "2018-07-03T08:00:52.786458Z",
  "changed_by": "dh3wr",
  "owners": ["dl1abc", "dh3wr", "dl2ic"]
}
```

**Auth response:** `limited`:

The output has been checked, if the asking username is in the `owners` array.

If yes: return the complete output from CouchDB.

If not: return just the whitelisted fields:

*Whitelist:* `_id`, `coordinates`, `description`, `hamcloud`, `owners`

Example (here `dh3wr` is asking with limited permission):

```
{
  "_id": "db0sda-dc1",
  "coordinates": [34.123456, -23.123456],
  "description": "some words about that node",
  "hamcloud": true,
  "owners": ["dl1abc", "dl2ic"]
}
```



### 6.2.3.9 GET /nodes/\_names

Item	Description
Description	Return just an JSON array of all nodes. Used where selections have to be done on the website.
Permission	<code>nodes.list</code>
Mapping to CouchDB	GET /nodes/_design/nodes/_list/names/byId

**Auth response: true:**

```
["db0sda-dc2","db0sda-dc3"]
```

**Auth response: false or limited:**

Send 403 Forbidden.

### 6.2.3.10 GET /nodes/\_descriptions

Item	Description
Description	Return just an JSON array of all nodenames and their description, sorted as a list. Used where descriptive selections have to be done on the website.
Permission	<code>nodes.list</code>
Mapping to CouchDB	GET /nodes/_design/nodes/_list/descriptions/descriptions

**Auth response: true:**

```
[
  {
    "_id": "db0sda-dc1",
    "description": "some words about that node"
  },
  {
    "_id": "db0sda-dc2",
    "description": "some words about that node"
  }
]
```

**Auth response: false or limited:**

Send 403 Forbidden.

### 6.2.3.11 PUT /nodes - Add new node

Item	Description
Description	Add a non-existing new node with <code>_id</code> <nodename>
Permission	<code>node.create</code>
Mapping to CouchDB	<code>PUT /nodes/&lt;nodesname&gt;</code>
Mandatory fields	<code>_id</code> , <code>auth_key</code> , <code>hamcloud</code> , <code>owners</code> , <code>description</code>
Field validation	<code>_id</code> : 6.1.1  <code>auth_key</code> : 6.1.5 <code>hamcloud</code> : boolean <code>owners</code> : 6.1.6 <code>description</code> : 6.1.11

**Auth response: true:** User to Database-Service: Example content to add node db0sda-dc2

```
{
  "_id": "db0sda-dc2",
  "auth_key": "73mxX4JLttzmVZ2",
  "coordinates": [34.123456, -23.123456],
  "description": "some words about that node",
  "hamcloud": true,
  "owners": ["d11abc", "dh3wr", "d12ic"]
}
```

Mapping to CouchDB with adding `created_*` information by database microservice:

```
{
  "_id": "db0sda-dc2",
  "auth_key": "73mxX4JLttzmVZ2",
  "coordinates": [34.123456, -23.123456],
  "description": "some words about that node",
  "hamcloud": true,
  "owners": ["d11abc", "dh3wr", "d12ic"],
  "created_on": "2018-07-08T11:50:02.168325Z",
  "created_by": <username from basic-auth>
}
```

**Auth response: false or limited:**

Send 403 Forbidden.

### 6.2.3.12 PUT /nodes - Edit existing node

Edit is differentiated from create by the presence of the `_rev` field.

Item	Description
Description	Edit the existing node <nodename>.
Permission	<code>node.update/&lt;nodename&gt;</code>
Mapping to CouchDB	<code>PUT /nodes/&lt;nodename&gt;</code>
Mandatory fields	<code>_id</code> , <code>_rev</code> (\rev has to be obtained as in <a href="#">6.2.3.8</a> )
Optional fields	<code>auth_key</code> , <code>hamcloud</code> , <code>owners</code> , <code>description</code> (minimum one)
Field validation	<code>_id</code> : Let CouchDB decide, as the entry is already present. <code>_rev</code> : Let CouchDB decide, as the entry is already present. <code>auth_key</code> : <a href="#">6.1.5</a> <code>hamcloud</code> : boolean <code>owners</code> : <a href="#">6.1.6</a> <code>description</code> : <a href="#">6.1.11</a>

**Auth response: true:**

User to Database-Service: Example content to edit user dl6pt setting new password

```
{
  "_id": "db0sda-dc2",
  "_rev": "1-ee070b17db9c3c58658d10fdedad2f48",
  "auth_key": "73mxX4JLttzmVZ2"
}
```

Mapping to CouchDB with adding `updated_*` information by database microservice:

`PUT /users/<username>`

```
{
  "_id": "db0sda-dc2",
  "_rev": "1-ee070b17db9c3c58658d10fdedad2f48",
  "auth_key": "73mxX4JLttzmVZ2",
  "updated_on": "2018-07-08T11:50:02.168325Z",
  "updated_by": <username from basic-auth>
}
```

**Auth response: false or limited:**

Send 403 Forbidden.

### 6.2.3.13 DELETE /nodes/<nodename>?rev=<rev>

Item	Description
Description	Delete node <nodename>
Permission	<code>node.delete/&lt;nodename&gt;</code>
Mapping to CouchDB	<code>DELETE /nodes/&lt;nodename&gt;?rev=1-093...</code>
Mandatory fields	<code>_rev</code> as parameter (\rev has to be obtained as in <a href="#">6.2.3.8</a> )

**Auth response: true:**

Transparent forwarding.

**Auth response: false or limited:**

Send 403 Forbidden.

Maybe the transmitter service should inform the connected transmitters of the deleted

#### 6.2.3.14 GET /rubrics

Item	Description
Description	Returns all rubrics (depending on the filtering in <a href="#">6.2.2</a> in JSON format.
Permission	rubric.read
Mapping to CouchDB	GET /rubrics/_design/rubrics/_view/byId?include_docs=true

**Auth response: true:**

```
{
  "total_rows": 1,
  "offset": 0,
  "rows": [
    {
      "_id": "dx-kw",
      "_rev": "1-166c3257894d0aea8ee68c1861ca508a",
      "number": 4,
      "description": "DX Cluster Spots KW",
      "label": "DX KW",
      "transmitter_groups": [
        "dl-hh"
      ],
      "transmitters": [
        "db0abc"
      ],
      "cyclic_transmit": false,
      "cyclic_transmit_interval": 0,
      "owner": [
        "dh3wr",
        "dliabc"
      ]
    }
  ]
}
```

**Auth response: false or limited:**

Send 403 Forbidden.

#### 6.2.3.15 GET /rubrics/\_view/byNumber?startkey=<n>&endkey=<m>

Item	Description
Description	Get the output as in section ??, but just for rubric numbers between <n> and <m>.
Permission	rubric.read
Mapping to CouchDB	GET /rubrics/_view/byNumber?startkey=<n>\&endkey=<m>
Field validation	startkey : 6.1.7  endkey : 6.1.7

Auth handling is the same as in section ??.

#### 6.2.3.16 GET /rubrics/\_view/byTransmitter?key=<transmittername>

Item	Description
Description	Get the output as in section ??, but just for rubrics that contain <transmittername> in the "transmitters" array.
Permission	rubric.read
Mapping to CouchDB	GET /rubrics/_view/byTransmitter?key=<transmittername>

Auth handling is the same as in section ??.

#### 6.2.3.17 GET /rubrics/\_view/byTransmitterGroup?key=<groupname>

Item	Description
Description	Get the output as in section ??, but just for rubrics that contains <groupname> in the "transmitter_groups" array.
Permission	rubric.read
Mapping to CouchDB	GET /rubrics/_view/byTransmitterGroup?key=<groupname>

Auth handling is the same as in section ??.

#### 6.2.3.18 GET /rubrics/\_view/withCyclicTransmit

Auth handling is the same as in section ??.

Item	Description
Description	Get the output as in section ??, but just for rubric that have the cyclic transmit flag enabled.
Permission	rubric.read
Mapping to CouchDB	GET /rubrics/\_view/withCyclicTransmit

#### 6.2.3.19 GET /rubrics/<rubricname>

Item	Description
Description	Return details of <rubricname> in JSON format. This does <b>not</b> include the content of the 10 rubric message slots.
Permission	rubric.read/<username>
Mapping to CouchDB	GET /rubric/<username>

**Auth response: true:**

```
{
  "_id": "dx-kw",
  "_rev": "1-166c3257894d0aea8ee68c1861ca508a",
  "number": 4,
  "description": "DX Cluster Spots KW",
  "label": "DX KW",
  "transmitter_groups": [
    "dl-hh"
  ],
  "transmitters": [
    "db0abc"
  ],
  "cyclic_transmit": false,
  "cyclic_transmit_interval": 0,
  "owner": [
    "dh3wr",
    "dl1abc"
  ]
}
```

**Auth response: false or limited:**  
Send 403 Forbidden.

#### 6.2.3.20 GET /rubrics/\\_names

Item	Description
Description	Return just an JSON array of all rubricnames. Used where selections have to be done on the website.
Permission	rubrics.list
Mapping to CouchDB	GET /rubrics/\_design/rubrics/\_list/names/byId

**Auth response: true:**

```
["dl-hh", "dl-nw"]
```

**Auth response: false or limited:**  
Send 403 Forbidden.

### 6.2.3.21 GET /rubrics/\_descriptions

Item	Description
Description	Return just an JSON array of all rubricnames their description in a JSON Array. Used where selections have to be done on the website.
Permission	rubrics.list
Mapping to CouchDB	GET /rubrics/_design/rubrics/_list/descriptions/descriptions

**Auth response: true:**

```
[
  {
    "_id": "dl-hh",
    "description": "Germany, Hamburg"
  },
  {
    "_id": "dl-by",
    "description": "Bavaria"
  }
]
```

**Auth response: false or limited:**

Send 403 Forbidden.

### 6.2.3.22 PUT /rubrics - Add new rubric

Item	Description
Description	Add a non-existing new rubric with _id <rubricname>
Permission	rubric.create
Mapping to CouchDB	PUT /rubrics/<rubricsname>
Mandatory fields	_id, number, description, label, transmitter_groups, cyclic_transmit, cyclic_transmit_interval, owners
Field validation	<div> <div>_id</div> <div>: 6.1.1 </div> </div> <div> <div>number</div> <div>: 6.1.7</div> </div> <div> <div>description</div> <div>: 6.1.11</div> </div> <div> <div>label</div> <div>: 6.1.8</div> </div> <div> <div>transmitter_groups</div> <div>: 6.1.9</div> </div> <div> <div>cyclic_transmit</div> <div>: boolean</div> </div> <div> <div>cyclic_transmit_interval</div> <div>: ??</div> </div> <div> <div>owners</div> <div>: 6.1.6</div> </div>

**Auth response: true:** User to Database-Service: Example content to add rubric dx-kw

```
{
  "_id": "dx-kw",
  "number": 4,
  "description": "DX Cluster Spots KW",
  "label": "DX KW",
  "transmitter_groups": [
    "dl-hh"
  ],
  "transmitters": [
    "db0abc"
  ],
  "cyclic_transmit": false,
  "cyclic_transmit_interval": 0,
  "owners": [
    "dh3wr",
    "dl1abc"
  ]
}
```

Mapping to CouchDB with adding created\_\* information by database microservice:

```
{
  "_id": "dx-kw",
  "number": 4,
  "description": "DX Cluster Spots KW",
  "label": "DX KW",
  "transmitter_groups": [
    "dl-hh"
  ],
  "transmitters": [
    "db0abc"
  ],
  "cyclic_transmit": false,
  "cyclic_transmit_interval": 0,
  "owners": [
    "dh3wr",
    "dl1abc"
  ],
  "created_on": "2018-07-08T11:50:02.168325Z",
  "created_by": <username from basic-auth>
}
```

**Auth response:** false or limited:  
Send 403 Forbidden.

### 6.2.3.23 PUT /rubrics - Edit existing rubric

Edit is differentiated from create by the presence of the **\_rev** field.

Item	Description
Description	Edit the existing node <nodename>.
Permission	rubric.update/<rubricname>
Mapping to CouchDB	PUT /rubrics/<rubricsname>
Mandatory fields	_id, _rev (\rev has to be obtained as in <a href="#">6.2.3.19</a> )
Optional fields	number, description, label, transmitter_groups, cyclic_transmit, cyclic_transmit_interval, owners (minimum one)
Field validation	_id : Let CouchDB decide, as the entry is already present. _rev : Let CouchDB decide, as the entry is already present. number : <a href="#">6.1.7</a> description : <a href="#">6.1.11</a> label : <a href="#">6.1.8</a> transmitter_groups : <a href="#">6.1.9</a> cyclic_transmit : boolean cyclic_transmit_interval : ?? owners : <a href="#">6.1.6</a>

**Auth response:** true:

User to Database-Service: Example content to edit to add rubric dx-kw changing the description

```
{
  "_id": "db0sda-dc2",
  "_rev": "1-ee070b17db9c3c58658d10fdedad2f48",
  "description": "New DX Cluster Spots KW",
}
```

Mapping to CouchDB with adding updated\_\* information by database microservice:

```
{
  "_id": "db0sda-dc2",
  "_rev": "1-ee070b17db9c3c58658d10fdedad2f48",
  "description": "New DX Cluster Spots KW",
  "updated_on": "2018-07-08T11:50:02.168325Z",
  "updated_by": <username from basic-auth>
}
```



**Auth response: false or limited:**  
Send 403 Forbidden.

#### 6.2.3.24 DELETE /rubrics/<rubricname>?rev=<rev>

Item	Description
Description	Delete node <rubricname>
Permission	rubric.delete/<rubricname>
Mapping to CouchDB	DELETE /rubrics/<rubricname>?rev=<rev>
Mandatory fields	_rev as parameter (\rev has to be obtained as in <a href="#">6.2.3.19</a> )

**Auth response: true:**  
Transparent forwarding.

**Auth response: false or limited:**  
Send 403 Forbidden.

### 6.2.3.25 GET /subscribers

Item	Description
Description	Returns all subscribers (depending on the filtering in <a href="#">6.2.2</a> in JSON format.
Permission	subscriber.read
Mapping to CouchDB	GET /subscribers/_design/subscribers/_view/byId?include_docs=true

**Auth response: true or limited:**

```
\begin{lstlisting}
{
  "total_rows": 1,
  "offset": 0,
  "rows": [
    {
      "_id": "dl1abc",
      "_rev": "1-44182aeb25815b19babe1c0a6bb95e68",
      "description": "Peter",
      "paggers": [
        {
          "ric": 123456,
          "function": 3,
          "name": "Peters Alphapoc",
          "type": "alphaPpoc",
          "enabled": true
        }
      ],
      "third_party_services": [
        "APRS",
        "BM"
      ],
      "owners": [
        "dh3wr",
        "dl1abc"
      ],
      "groups": [
        "rwth-afu"
      ]
    }
  ]
}
```

**Auth response: false:**

Send 403 Forbidden.

### 6.2.3.26 GET /subscribers/<subscribername>

Item	Description
Description	Return details of <subscribername> in JSON format.
Permission	subscriber.read/<subscribername>
Mapping to CouchDB	GET /subscriber/<subscribername>

**Auth response: true or limited:**

```
{
  "_id": "dl1abc",
  "_rev": "1-44182aeb25815b19babe1c0a6bb95e68",
  "description": "Peter",
  "pagers": [
    {
      "ric": 123456,
      "function": 3,
      "name": "Peters Alphapoc",
      "type": "alphaPpoc",
      "enabled": true
    }
  ],
  "third_party_services": [
    "APRS",
    "BM"
  ],
  "owners": [
    "dh3wr",
    "dl1abc"
  ],
  "groups": [
    "rwth-afu"
  ]
}
```

**Auth response: false:**

Send 403 Forbidden.

### 6.2.3.27 GET /subscribers/\_names

Item	Description
Description	Return just an JSON array of all subscribers. Used where selections have to be done on the website.
Permission	rubric.list
Mapping to CouchDB	GET /subscribers/_design/subscribers/_list/names/byId

**Auth response:** true or limited:

```
["dh3wr","dl2ic"]
```

**Auth response:** false:

Send 403 Forbidden.

### 6.2.3.28 GET /subscribers/\_descriptions

Item	Description
Description	Return just an JSON array of all subscribers and their descriptions. Used where selections have to be done on the website.
Permission	rubric.list
Mapping to CouchDB	GET /subscribers/_design/subscribers/_list/descriptions/descriptions

**Auth response:** true or limited:

```
[
  {
    "_id": "dh3wr",
    "description": "Ralf"
  },
  {
    "_id": "dl2ic",
    "description": "Thomas"
  }
]
```

**Auth response:** false:

Send 403 Forbidden.

### 6.2.3.29 PUT /subscribers - Add new subscriber

Item	Description
Description	Add a non-existing new subscriber with <code>_id</code> <subscribername>
Permission	<code>subscriber.create</code>
Mapping to CouchDB	PUT /subscribers/<subscribername>
Mandatory fields	<code>_id</code> , <code>description</code> , <code>paggers</code> , <code>ric</code> , <code>function</code> , <code>type</code> , <code>enabled</code> , <code>third_party_services</code> , <code>owners</code> , <code>groups</code>
Field validation	<code>_id</code> : 6.1.1 <code>description</code> : 6.1.11 <code>paggers</code> : The key for the array, must be present. <code>ric</code> : 6.1.12 <code>function</code> : 6.1.13 <code>type</code> : <code>enabled</code> : boolean <code>third_party_services</code> : The key for the array, must be present <code>owners</code> : 6.1.6 <code>groups</code> : The key for the array, must be present

**Auth response:** true: User to Database-Service: Example content to add subscriber dl1abc

```
{
  "_id": "dl1abc",
  "description": "Peter",
  "paggers": [
    {
      "ric": 123456,
      "function": 3,
      "name": "Peters Alphapoc",
      "type": "alphaPpoc",
      "enabled": true
    }
  ],
  "third_party_services": [
    "APRS",
    "BM"
  ],
  "owners": [
    "dh3wr",
    "dl1abc"
  ],
  "groups": []
}
```

Mapping to CouchDB with adding `created_*` information by database microservice:

```
\begin{lstlisting}
{
  "_id": "dl1abc",
  "description": "Peter",
  "paggers": [
    {
      "ric": 123456,
      "function": 3,
      "name": "Peters Alphapoc",
      "type": "alphaPpoc",
      "enabled": true
    }
  ],
  "third_party_services": [
    "APRS",
    "BM"
  ],
  "owner": [
    "dh3wr",
    "dl1abc"
  ],
}
```

```
"groups": [],  
"created_on": "2018-07-08T11:50:02.168325Z",  
"created_by": <username from basic-auth>  
}
```

**Auth response:** false or limited:  
Send 403 Forbidden.

### 6.2.3.30 PUT /subscribers - Edit existing subscriber

Edit is differentiated from create by the presence of the **\_rev** field.

Item	Description																						
Description	Add a edit a existing subscriber with _id <subscribername>																						
Permission	subscriber.update																						
Mapping to CouchDB	PUT /subscribers/<subscribername>																						
Mandatory fields	_id, _rev																						
Optional fields	description, pagers, ric, function, type, enabled, third_party_services, owners, groups (minimum one, if it's pagers, the complete pagers array has to be sent).																						
Field validation	<table><tr><td>_id</td><td>: Let CouchDB decide, as the entry is already present.</td></tr><tr><td>_rev</td><td>: Let CouchDB decide, as the entry is already present.</td></tr><tr><td>description</td><td>: <a href="#">6.1.11</a></td></tr><tr><td>pagers</td><td>: The key for the array, must be present.</td></tr><tr><td>ric</td><td>: <a href="#">6.1.12</a></td></tr><tr><td>function</td><td>: <a href="#">6.1.13</a></td></tr><tr><td>type</td><td>:</td></tr><tr><td>enabled</td><td>: boolean</td></tr><tr><td>third_party_services</td><td>: The key for the array, must be present</td></tr><tr><td>owners</td><td>: <a href="#">6.1.6</a></td></tr><tr><td>groups</td><td>: The key for the array, must be present</td></tr></table>	_id	: Let CouchDB decide, as the entry is already present.	_rev	: Let CouchDB decide, as the entry is already present.	description	: <a href="#">6.1.11</a>	pagers	: The key for the array, must be present.	ric	: <a href="#">6.1.12</a>	function	: <a href="#">6.1.13</a>	type	:	enabled	: boolean	third_party_services	: The key for the array, must be present	owners	: <a href="#">6.1.6</a>	groups	: The key for the array, must be present
_id	: Let CouchDB decide, as the entry is already present.																						
_rev	: Let CouchDB decide, as the entry is already present.																						
description	: <a href="#">6.1.11</a>																						
pagers	: The key for the array, must be present.																						
ric	: <a href="#">6.1.12</a>																						
function	: <a href="#">6.1.13</a>																						
type	:																						
enabled	: boolean																						
third_party_services	: The key for the array, must be present																						
owners	: <a href="#">6.1.6</a>																						
groups	: The key for the array, must be present																						

**Auth response: true:** User to Database-Service: Example content to edit the description of subscriber dllabc

```
{
  "_id": "dllabc",
  "_rev": "1-44182aeb25815b19babe1c0a6bb95e68",
  "description": "Peter",
}
```

Mapping to CouchDB with adding updated\_\* information by database microservice:

```
{
  "_id": "dllabc",
  "_rev": "1-44182aeb25815b19babe1c0a6bb95e68",
  "description": "Peter",
  "updated_on": "2018-07-08T11:50:02.168325Z",
  "updated_by": <username from basic-auth>
}
```

**Auth response: false or limited:**  
Send 403 Forbidden.

#### 6.2.3.31 DELETE /subscribers/<subscribername>?rev=<rev>

Item	Description
Description	Delete subscriber <subscribername>
Permission	subscriber.delete/<subscribername>
Mapping to CouchDB	DELETE /subscribers/<subscribername>?rev=<rev>
Mandatory fields	_rev as parameter (\rev has to be obtained as in ??)

**Auth response: true:**

Transparent forwarding.

**Auth response: false or limited:**

Send 403 Forbidden.

#### 6.2.3.32 GET /subscriber\_groups

Returns an array of existing subscribers\_groups tags in JSON format. This is allowed for **all** roles.

```
{
  ["dl.OV-G01", "dl.rwth-afu"]
}
```

#### 6.2.3.33 GET /transmitters/\_names

**limited:** \_id, \_rev, usage, timeslots, power, owners, groups, emergency\_power, coordinates, aprs\_bro and sub-JSON-nodes of these keys.

Return an JSON array of all transmitter names. Used where selections have to be done on the website. For all roles example result:

Mapping to CouchDB: GET /transmitters/\_design/transmitters/\_list/names/byId

```
{
  ["db0sda", "db0wa"]
}
```

#### 6.2.3.34 GET /transmitters/\_view/groups

Returns a JSON array of used transmitter groups tags from all known transmitters. Used for a suggestion of already existing transmitter group tags on the website.

#### 6.2.3.35 DELETE /transmitters/<transmittername>?rev=

Delete the transmitter <transmittername>. Also delete the transmitter from explicit entries on rubrics.

First get transmitter revision as defined in section 6.2.6.2. Then send the request with the revision.

#### 6.2.3.36 PUT /transmitters - Add new transmitter

Add a new the transmitter.

Allowed roles are **admin** and **support**. Role **user** will get 403 Forbidden Example data to send:

```
{
  "_id": "db0wa",
  "usage": "widerange",
  "timeslots": [
    true,
    true,
    false,
    true,
  ]
}
```



```

        true,
        false,
        true,
        true,
        false,
        false,
        true,
        true,
        false,
        true,
        true,
        false
    ],
    "power": 20,
    "owners": [
        "dh3wr"
    ],
    "groups": [
        "dl.nw.koeln.aachen"
    ],
    "emergency_power": {
        "available": true,
        "infinite": false,
        "duration": 7200
    },
    "coordinates": [
        50.71613,
        6.165481
    ],

    "aprs_broadcast": false,
    "enabled": true,
    "auth_key": "Arj39135jAKS",
    "antenna": {
        "type": "omni",
        "gain": 0,
        "direction": 0,
        "agl": 1
    }
}

```

The MS has to add

```

    "created_on": "2018-07-03T08:00:52.786458Z",
    "created_by": "dh3wr",

```

and add it to the REST PUT call.

### 6.2.3.37 PUT /transmitters - Edit existing transmitter

Edit an existing transmitter.

Allowed roles are **admin** and **support**. Role **user** will get 403 Forbidden `_id` and `_rev` have to be send always. So first get the current revision of the transmitter with a GET `/transmitters/<transmittername>`.

Example data to send:

```

{
    "_id": "db0wa",
    "_rev": "3-212820d0a75061289c8fbe39192fde22",
    "usage": "widerange"
}

```

The MS has to add or change the keys

```

    "changed_on": "2018-07-03T08:00:52.786458Z",
    "changed_by": "dh3wr",

```

and add them to the REST PUT call.

Is an active connection reset to that transmitter necessary? If the Authkey changes, an already established connection will keep working? And what about

## 6.2.4 Call Service

### 6.2.4.1 GET /calls

Returns the last 100 calls with all details.

### 6.2.4.2 GET /calls?limit=<number>

Returns the last <number> of calls with all details. If <number> is higher than the available calls, just return all available calls.

### 6.2.4.3 GET /calls/\_view/byDate

With GET parameters:

```
GET /calls/_view/byDate?startkey="<startddate">&endkey="<enddate">
```

Returns the calls made within the specified time span with all details. If there are no calls stored in the specified time span, return empty JSON.

### 6.2.4.4 GET /calls/\_view/byIssuer

```
GET /calls/_view/byIssuer?key="dh3wr"
```

Returns all the calls issued from callsign dh3wr with all details. If there are no calls stored with in the specified time span, return empty JSON. (The microservice has to transform the request into *startkey="dh3wr"&endkey="dh3wr"* to the CouchDB GET request by itself.)

### 6.2.4.5 GET /calls/\_view/byRecipient

```
GET /calls/_view/byRecipient?key="dh3wr"
```

Returns all the calls with recipient callsign dh3wr with all details. If there are no calls stored in the specified time span, return empty JSON. (The microservice has to transform the request into *startkey="dh3wr"&endkey="dh3wr"* to the CouchDB GET request by itself.)

### 6.2.4.6 GET /calls/\_view/pending

Return all details of pending calls, that are not transmitted by at least one transmitter.

### 6.2.4.7 GET /calls/\_view/pending\_all

Return all details of pending calls, that are not transmitted by all designated transmitters.

### 6.2.4.8 POST /call

Insert call to the system. Send in POST content:

```
{
  "subscriber": ["dh3wr",...],
  "subscriber\_groups": ["d1.ov-g01",...]
  "priority" : 1 to 5,
  "message": "This is an example call",
  "transmitter_groups": ["d1-all","on-all"]
}
```

Any combination of the given filter method shall be possible. Is this possible?

## 6.2.5 Rubric Service

### 6.2.5.1 GET /news

Returns an array of all rubrics and their content in JSON format.

zu be-  
denken

### 6.2.5.2 GET /news/\_view/byRubric

GET /news/\_view/byRubric?startkey="metar-dl"&endkey="metar-dl"

Returns just the content of <rubricname> content in JSON format.

### 6.2.5.3 GET /news/\_view/byRubric/message\_no>

Returns just the content of <rubricname> content and message number <message\_no> in JSON format.

### 6.2.5.4 PUT /news/<rubricname>

Add content to rubric <rubricname> on the first message slot and move the existing message one to the end. The 10th. entry will be lost. An automated resend of all rubric content slots will be necessary.

### 6.2.5.5 PUT /news/<rubricname>/<message\_no>

Add or override the content of rubric <rubricname> on the message slot <message\_no>. An automated resend of just this message slot will be necessary.

### 6.2.5.6 DELETE /news/<rubricname>?rev=

Delete all content in rubric <rubricname>. The content will be still on Skypers that have received it before, but it will not be transmitted periodically any more. No dependency check necessary.

### 6.2.5.7 DELETE /news/<rubricname>/<message\_no>?rev=

Delete the content in rubric <rubricname> with message slot <message\_no>. The content will be still on Skypers that have received it before, but it will not be transmitted periodically any more. No dependency check necessary.

## 6.2.6 Transmitter Service

### 6.2.6.1 GET /transmitters

Return all transmitters with all details in JSON format.

### 6.2.6.2 GET /transmitter/<transmittername>

Return all details just of transmitter <transmittername>.

### 6.2.6.3 POST /transmitters/\_bootstrap

POST /transmitter/bootstrap

```
{
  "callsign": "db0avr",
  "auth_key": "<secretInCleartext>",
  "software": {
    "name": "UniPager",
    "version": "1.0.2"
  }
}
```

Answers from the bootstrap REST call The application type shall be application/json.

200 OK

```
{
  "timeslots": [true, true, false, true, ...],
  "nodes": [
    {
      "host": "node1.ampr.org",
      "port": 4000,
      "reachable": true,
      "last_seen": "2018-07-03T07:43:52.783611Z",
      "response_time": 42
    }
  ]
}
```

Was mag  
denn hier  
der Port  
sein?

423 Locked

```
{
  "error": "Transmitter temporarily disabled by configuration."
}
```

423 Locked

```
{
  "error": "Transmitter software type not allowed due to serious bug."
}
```

### 6.2.6.4 POST /transmitters/\_heartbeat

POST /transmitter/heartbeat

```
{
  "callsign": "db0avr",
  "auth_key": "<secretInCleartext>",
  "ntp_synced": true
}
```

Answers from the heartbeat REST call The application type shall be application/json.

200 OK

```
{
  "status": "ok"
}
```

If network wants to assign new timeslots without disconnecting (for dynamic timeslots)

200 OK

```
{
  "status": "ok",
  "timeslots": [true, true, false, ...],
  "valid_from": "2018-07-03T08:00:52.786458Z"
}
```

If network wants to initiate handover to other node

503 Service unavailable

```
{
  "error": "Node not available, switch to other node."
}
```

## 6.2.7 Cluster Service

### 6.2.7.1 POST /cluster/discovery

## 6.2.8 Telemetry Service

### 6.2.8.1 GET /telemetry/transmitters

Return the stored telemetry **summary** values for all transmitters.

### 6.2.8.2 GET /telemetry/transmitters/<transmittername>

Return all the stored telemetry values for transmitter <transmittername>.

### 6.2.8.3 GET /telemetry/nodes

Return the stored telemetry **summary** values for all nodes.

### 6.2.8.4 GET /telemetry/nodes/<nodename>

Return all the stored telemetry values for node <nodename>.

### 6.2.8.5 WS /telemetry

See [the section for Websocket API](#).

## 6.2.9 Database Changes Service

### 6.2.9.1 WS /changes

See [the section for Websocket API on database changes](#).

## 6.2.10 Status Service

The purpose of the status service is to provide a short overview of the DAPNET network and the microservices.

### 6.2.10.1 GET /status/nodes

No authentication required.

Answer: 200 OK

```
{
  "nodes": [
    {
      "host": "node1.ampr.org",
      "port": 4000,
      "reachable": true,
      "last_seen": "2018-07-03T07:43:52.783611Z",
      "response_time": 42
    }
  ],
  "connections": {
    "rabbitmq": true,
    "couchdb": true,
    "hamcloud": true,
  },
  "hamcloud_node": false,
  "general_health": true
}
```

What is  
"port"?

#### 6.2.10.2 GET /status/node/<nodename>

No authentication required.

Answer: 200 OK

```
{
  "host": "node1.ampr.org",
  "port": 4000,
  "reachable": true,
  "last_seen": "2018-07-03T07:43:52.783611Z",
  "response_time": 42,
  "connections": {
    "rabbitmq": true,
    "couchdb": true,
    "hamcloud": true,
  },
  "hamcloud_node": false,
  "general_health": true
}
```

What is  
"port"?

#### 6.2.10.3 GET /status

Get status of this node. 200 OK

```
{
  "good_health" : true,
  "version" : "1.2.3"
  "microservices\_running" : {
    "database" : true,
    "call" : true,
    "rubric" : true,
    "transmitter" : true,
    "cluster" : true,
    "telemetry" : true,
    "database-changes" : true,
    "statistics" : true,
    "rabbitmq" : true,
    "thirdparty" : true
  }
}
```

#### 6.2.10.4 GET /status/<service\_name>

List of valid values for *service\_name*:

database-service  
call-service  
rubric-service  
transmitter-service  
cluster-service  
telemetry-service  
database-changes-service  
statistics-service  
rabbitmq-service

200 OK

<Status output from service itself>

## 6.2.11 Statistics Service

### 6.2.11.1 GET /statistics

No authentication required.

Answer: 200 OK

```
{
  "users" : 1234,
  "transmitters": {
    "personal": {
      "online": 13
      "total": 34
    },
    "widerage": {
      "online": 53,
      "total": 97
    }
  }
  "nodes": {
    "online": 10,
    "total": 19
  },
  "processed_calls": 1234,
  "processed_rubric_content_changes": 234
}
```

On the calls and rubric content changes: Always increasing counter link traffic on network device or reset at 00:00 am?

## 6.2.12 Auth Service

The auth service provides authentication information to all other services. It works on the CouchDB and reads information from there.

### 6.2.12.1 Password hashing

In version 1, the used hashing algorithm was PBKDF2WithHmacSHA1. In DAPNET 2, the preferred algorithm is BCrypt. In order not to send emails to every already registered user to update her/his password, both hashing algorithms are supported by the Auth Service. Anyway, as soon as a user sends its credentials and in the database, still the PBKDF\ hash is stored, it is updated with the corresponding format.

The format of the hash can be distinguished by the \$2a\$, \$2b\$ or \$2y\$-Prefix. With this method, the transition happens transparently regarding the user.

### 6.2.12.2 Role definition

The following roles are available:

Name	Description
<b>user</b>	This is a normal user with the possibility of being owner of a subscriber association and transmitters.
<b>support</b>	Trustworthy volunteer that is working on the user support, e.g. the ticket system. Can manage user, transmitter, subscriber and rubric settings.
<b>admin</b>	Can do everything, especially create new nodes.
<b>thirdparty.&lt;X&gt;</b>	A machine interface user being able to subscribe to the third party MQTT topics for service <X>. Examples are APRS or Brandmeister.
<b>guest</b>	Any other request making entity.

Table 6.2: Role definition

A user can be part of multiple roles. The Auth service as to check for each permission, if at least one of the roles allows it. Otherwise it is denied.

### 6.2.12.3 Permission definition

The following permissions are available:

Name	Description
<b>all</b>	All data access and modification allowed.
<b>none</b>	Data access and modification forbidden.
<b>if_owner</b>	Data modification allowed, if asking username is in <b>owners</b> list of the entity.
<b>limited</b>	Data access is restricted. The consuming service has to define what that means in detail.

Table 6.3: Permission definition

### 6.2.12.4 Permission naming definition

The name of the permission has the following significance:



Name	Description
<b>*.list</b>	Get an array of all <code>_ids</code> or further summary data. Used for selections by users.
<b>*.read</b>	Get one or many documents in complete or limited version. A white list defined in the corresponding section of this document is given for the limited case.
<b>*.create</b>	Create a new document.
<b>*.update</b>	Modify an existing document.
<b>*.delete</b>	Delete an existing document.
<b>user.change_role</b>	Change the <b>roles</b> array of a user's entry.

Table 6.4: Permission naming definition

#### 6.2.12.5 Permission matrix

The following table defines for each role which actions are permitted.

action	admin	support	user	guest	thirdparty.<X>
<b>user.list</b>	all	all	all	none	all
<b>user.read</b>	all	all	limited	none	none
<b>user.create</b>	all	all	none	none	none
<b>user.update</b>	all	all	if_owner	none	if_owner
<b>user.delete</b>	all	all	if_owner	none	if_owner
<b>user.change_role</b>	all	none	none	none	none

Table 6.5: Role's permissions for users database

action	admin	support	user	guest	thirdparty.<X>
<b>node.list</b>	all	all	limited	limited	all
<b>node.read</b>	all	all	limited	limited	all
<b>node.create</b>	all	all	none	none	none
<b>node.update</b>	all	all	none	none	none
<b>node.delete</b>	all	all	none	none	none

Table 6.6: Role's permissions for nodes database

action	admin	support	user	guest	thirdparty.<X>
rubric.list	all	all	all	none	all
rubric.read	all	all	all	none	all
rubric.create	all	all	none	none	none
rubric.update	all	all	none	none	none
rubric.delete	all	all	none	none	none

Table 6.7: Role's permissions for rubrics database

action	admin	support	user	guest	thirdparty.<X>
news.read	all	all	all	none	all
news.create	all	all	if_owner	none	if_owner
news.update	all	all	if_owner	none	if_owner
news.delete	all	all	if_owner	none	if_owner

Table 6.8: Role's permissions for news database

action	admin	support	user	guest	thirdparty.<X>
subscriber.list	all	all	all	none	all
subscriber.read	all	all	limited	none	limited
subscriber.create	all	all	none	none	none
subscriber.update	all	all	if_owner	none	if_owner
subscriber.delete	all	all	if_owner	none	if_owner
subscriber_groups.list	all	all	all	none	all

Table 6.9: Role's permissions for subscribers database

action	admin	support	user	guest	thirdparty.<X>
transmitter.list	all	all	all	limited	all
transmitter.read	all	all	limited	none	limited
transmitter.create	all	all	none	none	none
transmitter.update	all	all	if_owner	none	if_owner
transmitter.delete	all	all	if_owner	none	if_owner
transmitter_groups.list	all	all	all	none	all

Table 6.10: Role's permissions for transmitters database

action	comment
transmitter.new_conn_post	If transmitter credentials are ok
transmitter.rabbitmq.subscribe	If transmitter credentials are ok, for both RX of messages and TX of telemetry

Table 6.11: Role's permissions for transmitters

action	admin	support	user	guest	thirdparty.<X>
ws.telemetry.subscribe	all	all	all	all	all
ws.database_change.subscribe	all	all	limited	none	limited

Table 6.12: Role's permissions for websocket

**limited:** Just same content as permitted over http. Websocket authentication like in unipager.

action	admin	support	user	guest	thirdparty.<X>
status.read	all	all	all	all	all
statistics.read	all	all	all	all	all

Table 6.13: Role's permissions for status and statistics

action	admin	support	user	guest	thirdparty.<X>
thirdparty.subscribe.aprs	all	none	none	none	if _<X>=aprs
thirdparty.subscribe.brandmeister	all	none	none	none	if _<X>=brandmeister

Table 6.14: Role's permissions for MQTT subscription

#### 6.2.12.6 Auth API definition

The following table states all Auth Service API calls and their description. The POST request has to come always with a POST content of:

```
{"username": "<asking_user>", "password": "<asking_user's_password>"}
```

There are two endpoints on the Auth service. One for overview of permissions and one explicitly answering, if a permission is granted.

#### POST /auth/users/login

This returns an detailed overview of all permissions of the user referred in the POST content. If a permission is not listed, it's implicitly none. Example output:

```
{
  "user": {
    "roles": [
      "admin",
      "support",
      "user"
    ],
    "enabled": true,
    "email": "mailmenot@dl2ic.de",
    "created_on": "2018-07-08T11:50:02.168325Z",
    "created_by": "dh3wr",
    "_rev": "4-3ebbe52b3da83a2a6c1f8093efebdc07",
    "_id": "dl2ic"
  },
  "permissions": {
    "user.update": "all",
    "user.read": "all",
    "user.list": "all",
    "user.delete": "all",
    "user.create": "all",
    "user.change\_role": "all",
    "transmitter\_groups.list": "all",
    "transmitter.update": "all",
    "transmitter.read": "all",
    "transmitter.list": "all",
    "transmitter.delete": "all",
    "transmitter.create": "all",
    "subscriber\_groups.list": "all",
    "subscriber.update": "all",
    "subscriber.read": "all",
    "subscriber.list": "all",
    "subscriber.delete": "all",
    "subscriber.create": "all",
    "rubric.update": "all",
    "rubric.read": "all",
    "rubric.list": "all",
    "rubric.delete": "all",
    "rubric.create": "all",
    "node.update": "all",
    "node.read": "all",
    "node.list": "all",
    "node.delete": "all",
    "node.create": "all",
    "news.update": "all",
    "news.read": "all",
    "news.delete": "all",
    "news.create": "all"
  }
}
```

**POST /auth/users/permissions/<permission>/<entity>**

This is a explicit query endpoint for a specific <permission> as listed in section [6.2.12.5](#). The entity against the permission is applied is <entity>. Ownership bf is considered in this case by the Auth Service.

**POST /auth/users/permissions/<service>.read**

This is a explicit query endpoint for a <service>.read permission grant to get one or many documents. Ownership is **NOT** considered in this case. The consuming service has to take care of adequate data handling.

There are three possible answers:

**Access is granted completely:**

```
{ "access": true }
```

**Access is forbidden completely:**

```
{ "access": false }
```

**Access granted, but limited to a subset of data:**

```
{
  "access": false,
  "limited": true
}
```

If the limited key is not present, the access is still forbidden.

**GET /auth/users/roles**

A JSON array of available roles can be obtained without authentication by.

Example output:

```
["user", "support", "admin", "thirdparty.brandmeister", "thirdparty.aprs"]
```

#### 6.2.12.7 Auth call reference list

In order to shorten the table, the abbreviation **/a/u/p** is used to represent **/auth/users/permission**.

The value in < > is always the resource to be accessed.

Auth REST endpoint	Referring to	Section
POST /a/u/p/user.list	GET /users/_usernames	<a href="#">6.2.3.3</a>
POST /a/u/p/user.read/<username>	GET /users/<username>	<a href="#">6.2.3.2</a>
POST /a/u/p/user.read	GET /users	<a href="#">6.2.3.1</a>
POST /a/u/p/user.create	PUT /users	<a href="#">6.2.3.4</a>
POST /a/u/p/user.update/<username>	PUT /users	<a href="#">6.2.3.5</a>
POST /a/u/p/user.delete/<username>	DELETE /users/<username>?rev=	<a href="#">6.2.3.6</a>
POST /a/u/p/user.change_role/<username>	PUT /users	<a href="#">6.2.3.5</a>
POST /a/u/p/transmitter.list	GET /transmitters/_transmitternames	??
POST /a/u/p/transmitter.read/<txname>	GET /transmitters/<txname>	??
POST /a/u/p/transmitter.read	GET /transmitters[?*	??
POST /a/u/p/transmitter.create	PUT /transmitters	??
POST /a/u/p/transmitter.update/<txname>	PUT /transmitters	??
POST /a/u/p/transmitter.delete/<txname>	DELETE /transmitters/<txname>?rev=	??
POST /a/u/p/transmitter_groups.list	GET /transmitters/_groups	??
POST /a/u/p/transmitter.new_conn_post	POST /transmitters/bootstrap	??
POST /a/u/p/transmitter.rabbitmq.subscribe	<i>RabbitMQ Auth</i>	??
POST /a/u/p/subscriber.list	GET /subscribers/_subscribernames	??
POST /a/u/p/subscriber.read/<subcname>	GET /subscribers/<subcname>	??
POST /a/u/p/subscriber.read	GET /subscribers[?*	??
POST /a/u/p/subscriber.create	PUT /subscribers	??
POST /a/u/p/subscriber.update/<subcname>	PUT /subscribers	??
POST /a/u/p/subscriber.delete/<subcname>	DELETE /subscribers/<subcname>?rev=	??
POST /a/u/p/subscriber_groups.list	GET /subscribers/_groups	??
POST /a/u/p/node.list	GET /nodes/_nodenames GET /nodes/_nodenamedescription	?? ??
POST /a/u/p/node.read/<nodename>	GET /nodes/<nodename>	<a href="#">6.2.3.8</a>
POST /a/u/p/node.read	GET /nodes[?*	<a href="#">6.2.3.7</a>
POST /a/u/p/node.create	PUT /nodes	<a href="#">6.2.3.11</a>
POST /a/u/p/node.update/<nodename>	PUT /nodes	<a href="#">6.2.3.12</a>
POST /a/u/p/node.delete/<nodename>	DELETE /nodes/<nodename>?rev=	<a href="#">6.2.3.13</a>
POST /a/u/p/rubric.list	GET /rubrics/_rubricnames	??
POST /a/u/p/rubric.read/<rubricname>	GET /rubrics/<rubricsame>	<a href="#">6.2.3.19</a>
POST /a/u/p/rubric.read	GET /rubrics[?*	<a href="#">6.2.3.14</a>
POST /a/u/p/rubric.create	PUT /rubrics	<a href="#">6.2.3.22</a>
POST /a/u/p/rubric.update/<rubricname>	PUT /rubrics	<a href="#">6.2.3.23</a>
POST /a/u/p/rubric.delete/<rubricname>	DELETE /rubrics/<rubricsname>?rev=	<a href="#">6.2.3.24</a>
POST /a/u/p/news.read	GET /news[?*	??
POST /a/u/p/news.create	PUT /news	??
POST /a/u/p/news.update/<rubricname>	PUT /news/<rubricname>	??
POST /a/u/p/news.delete/<rubricname>	DELETE /news/<rubricname>?rev=	??
POST /a/u/p/news.delete/<rubricname>	DELETE /news/<rubricname>/<msg_no>?rev=	??
POST /a/u/p/ws.telemetry.subscribe	WS /telemetry/*	??
POST /a/u/p/ws.database_change.subscribe	WS /changes/*	??
POST /a/u/p/status.read	GET /status/*	??
POST /a/u/p/statistics.read	GET /statistics/*	??
POST /a/u/p/thirdparty.subscribe.<service>	<i>MQTT Auth</i> to subscribe to topic <service>	??

Table 6.15: Auth REST API endpoint and references

### 6.2.13 RabbitMQ Service

#### 6.2.13.1 GET /rabbitmq/\*

## 6.3 RabbitMQ

There are 3 exchanges available on each RabbitMQ instance:

**dapnet.calls** Messages shared between all nodes

**dapnet.local\_calls** Messages coming from the local node instance

**dapnet.telemetry** Messages containing telemetry from transmitters

### 6.3.1 Transmitters

Valid Messages are:

#### 6.3.1.1 dapnet.calls

The messages to transfer data to be transmitted by the transmitter have the following format.

For each transmission, there is a separate RabbitMQ message, as different receivers might need different text encoding. All encoding is already done, when this message is created. The transmitter does no character encoding at all. Both personal pagings and rubric related messages are transmitted with this protocol.

```
{
  "id": "016c25fd-70e0-56fe-9d1a-56e80fa20b82",
  "protocol": "pocsag",
  "priority": 3,
  "expires": "2018-07-03T08:00:52.786458Z",
  "message": {
    "ric": 12342, (max 21 Bits)
    "type": "alphanum", | "numeric"
    "speed": 1200,
    "function": 0 to 3,
    "data": "Lorem ipsum dolor sit amet"
  }
}
```

The selection of the transmitter is done by means of the routing key. Besides, the priority is also used in the RabbitMQ queuing to deliver higher priority messages first.

#### 6.3.1.2 dapnet.local\_calls

Same as for the the network originated calls in section 6.3.1.1.

### 6.3.2 Telemetry

On the telemetry exchange, all transmitters and nodes publish their telemetry messages. The format the same as in section 6.4 and 6.5.

### 6.3.3 MQTT API for third-party consumers

In order to allow third-party instances like , or others to get the emitted calls and rubric contents in a real time event driven way, there is an MQTT API. It is not implemented via a dedicated MQTT broker, but uses the existing RabbitMQ instance (<https://www.rabbitmq.com/mqtt.html>). There is no distribution of the messages via this MQTT broker; it is local only. So every node publishes the messages locally on its own. Each subscriber has an array of enabled third-party applications.

check  
with  
DL2IC

This allow to define the user, if call directed to her/his subscriber shall be also sent to third-party services (see 6.8.4.

The currently existing MQTT topics are defined in the CouchDB (see section 6.8.7). This makes it possible to add more third-party services and authorized users during runtime without the need to update the software. The valid users to subscribe to the topic are also listed in the same CouchDB database.

The only permitted access for third-party consumers is read. So the subscribe request from a third-party MQTT-Client must use authentication which is checks against the CouchDB data. If correct, read access is granted. Core software has always write access to publish the calls group messages.

The transmitters who are supposed to send out the personal call or the rubric content are published with callsign, geographic location and type of transmitter (widerange or personal). With this generic concept, every third-party application can decide what to do with the content received.

The encoding of the data is UTF-8.

The format of the data published for **personal paging calls** is

```
{
  "pagingcall" : {
    "srccallsign" : "d12ic",
    "dstcallsign" : "dh3wr",
    "dstric" : 12354,
    "dstfunction" : 0 .. 3,
    "priority" : 3,
    "message" : "DAPNET 2.0 rocks dear YL/OM"
    "transmitted_by" : [
      {
        "callsign" : "db0abc",
        "lat" : 12.123456,
        "long" : 32.123456,
        "type" : "personal" | "widerange"
      },
      {
        "callsign" : "db0def",
        "lat" : 12.123456,
        "long" : 32.123456,
        "type" : "personal" | "widerange"
      }
    ],
    "timestamp" : "2018-07-03T08:00:52.786458Z"
  }
}
```

The format of the data published for **rubric\_content paging calls** is

```
{
  "rubricmessage" : {
    "message" : ""
    "transmitted_by" : [
      {
        "callsign" : "db0abc",
        "lat" : 12.123456,
        "long" : 32.123456,
        "type" : "PERONAL" | "WIDERANGE"
      },
      {
        "callsign" : "db0def",
        "lat" : 12.123456,
        "long" : 32.123456,
        "type" : "PERONAL" | "WIDERANGE"
      }
    ],
    "timestamp" : "2018-07-03T08:00:52.786458Z"
  }
}
```



## 6.4 Telemetry from Transmitters

Telemetry is sent from transmitters to the RabbitMQ exchange **dapnet.telemetry** as defined in section 6.3. It is also used in the same way on the websocket API to inform the website and the app about the telemetry in real-time in section 6.7.1 and 6.7.2.

This is sent every minute in complete. If there are changes, just a subset is sent. The name of the transmitter is used as routing key for the message.

```
{
  "onair": true,
  "node": {
    "name": "db0xyz",
    "ip": "44.42.23.8",
    "port": 1234,
    "connected": true,
    "connected_since": "2018-07-03T08:00:52.786458Z"
  },
  "ntp": {
    "synced": true,
    "offset": 124,
    "server": ["134.130.4.1", "12.2.3.2"],
  },
  "messages": {
    "queued": [123, 123, 123, 123, 123, 123],
    "sent": [123, 123, 123, 123, 123, 123]
  },
  "temperatures": {
    "unit": "C" | "F" | "K",
    "air_inlet": 12.2,
    "air_outlet": 14.2,
    "transmitter": 42.2,
    "power_amplifier": 45.2,
    "cpu": 93.2,
    "power_supply": 32.4,
    "custom": [
      {"value": 12.2, "description": "Aircon Inlet"},
      {"value": 16.2, "description": "Aircon Outlet"},
      {"value": 12.3, "description": "Fridge Next to Programmer"}
    ]
  },
  "power_supply": {
    "on_battery": false,
    "on_emergency_power": false,
    "dc_input_voltage": 12.4,
    "dc_input_current": 3.23
  },
  "rf_output": {
    "fwd": 12.2,
    "refl": 12.2,
    "vswr": 1.2
  },
  "config": {
    "ip": "123.4.3.2",
    "timeslots": [true, false, ..., false],
    "software": {
      name: "Unipager" | "MMDVM" | "DAPNET-Proxy",
      version: "v1.2.3", | "20180504" | "v2.3.4",
    },
  },
  "hardware": {
    "platform": "Raspberry Pi 3B+"
  },
  "rf_hardware": {
    "c9000": {
      "name": "C9000 Compact",
      "<pa_dummy>": {
        "output_power": 123,
        "port": "/dev/ttyUSB0"
      }
    },
    "<rpc>": {
      "version": "XOS/2.23pre"
    }
  }
}
```

```

    },
    "raspager": {
      "name": "Raspager",
      "modulation": 13,
      "power": 63,
      "external_pa": false,
      "version": "V2"
    },
    "audio": {
      "name" = "Audio",
      "transmitter": "GM1200" | "T7F" | "GM340" | "FREITEXT",
      "audio_level": 83,
      "tx_delay": 3
    },
    "rfm69": {
      "name" : "RFM69",
      "port": "/dev/ttyUSB0"
    },
    "mmdvm": {
      "name" : "MMDVM",
      "dapnet_exclusive": true
    }
  },
  "proxy" : {
    "status": "connected" | "connecting" | "disconnected"
  }
}

```

## 6.5 Telemetry from Nodes

Telemetry is sent from nodes to the RabbitMQ exchange **dapnet.telemetry** as defined in section 6.3. It is also used in the same way on the websocket API to inform the website and the app about the telemetry in real-time in section 6.7.3 and 6.7.4.

This is sent every minute in complete. If there are changes, just a subset is sent. The name of the nodes is used as routing key for the message.

```

{
  "good_health" : true,
  "microservices" : {
    "database" : {
      "ok" : true,
      "version" : "1.2.3"
    },
    "call" : {
      "ok" : true,
      "version" : "1.2.3"
    },
    "rubric" : {
      "ok" : true,
      "version" : "1.2.3"
    },
    "transmitter" : {
      "ok" : true,
      "version" : "1.2.3"
    },
    "cluster" : {
      "ok" : true,
      "version" : "1.2.3"
    },
    "telemetry" : {
      "ok" : true,
      "version" : "1.2.3"
    },
    "database-changes" : {
      "ok" : true,
      "version" : "1.2.3"
    },
    "statistics" : {
      "ok" : true,

```

```

    "version" : "1.2.3"
  },
  "rabbitmq" : {
    "ok" : true,
    "version" : "1.2.3"
  },
  "thirdparty" : {
    "ok" : true,
    "version" : "1.2.3"
  },
  },
  "connections" : {
    "transmitters" : 123,
    "third_party" : 3
  },
  "system" : {
    "free_disk_space_mb": 1234
    "cpu_utilization": 0.2
    "is_hamcloud" : false
  }
}

```

## 6.6 Statistic, Status and Telemetry REST API

The statistic and telemetry REST API provides up-to-date information regarding the transmitters and the network via REST. This can be used by e.g. grafana to draw nice graphes or nagios plugins.

### 6.6.1 Telemetry from Transmitters

For authentication refer to section [Auth Service](#). GET /telemetry/transmitters Here all stored telemetry from all transmitters is provided.

Answer: 200 OK See [6.4](#)

GET /telemetry/transmitters/<transmittername> Here all stored telemetry from the specified transmitter is provided.

Answer: 200 OK

See [6.4](#)

GET /telemetry/transmitters/<transmittername>/<section\_of\_telemetry> Here all stored telemetry within the telemetry section from the specified transmitter is provided. Possible sections are 2. Level JSON groups, see [6.4](#).

Examples: onair, telemetry, transmitter\_configuration

Answer: 200 OK

See [6.4](#)

### 6.6.2 Telemetry from Nodes

GET /telemetry/nodes Here all stored telemetry from all nodes is provided.

Answer: 200 OK See [6.5](#)

GET /telemetry/nodes/<nodename> Here all stored telemetry from the specified node is provided.

Answer: 200 OK

See [6.5](#)

## 6.7 Websocket API

The idea is to provide an API for the website and the app to display real-time information without the need of polling. A websocket server is listing to websocket connections. Authentication is done by a custom JOSH handshake. The connection might be encrypted with SSL if using the Internet or plain if using HAMNET.

The data is taken from the **dapnet.telemetry** exchange from the RabbitMQ instance and further other sources if necessary.

For authentication refer to section [Auth Service](#).

Table [6.7](#) lists the main endpoints in the websocket interface:

Endpoint	Microservice
WS /telemetry/transmitters	Summary data of all TX
WS /telemetry/transmitters/<TxName>	Details for TX <TxName>
WS /telemetry/nodes	Summary data of all Nodes
WS /telemetry/nodes/<NodeName>	Details for Node <NodeName>
WS /changes	Database changes

Table 6.16: Websocket endpoints

### 6.7.1 Telemetry from Transmitters - Summary of all TX

URL: `ws://FQDN/telemetry/transmitters`

The data is the same as received from the **dapnet.telemetry** exchange from the RabbitMQ instance. It is defined in section [6.4](#).

The websocket-Server generates an array of JSON Objects which have the name of the transmitter obtained from the RabbitMQ routing key.

The current time slot is also sent in the summary and updated also by its own every time a time slot change happens.

```
{
  "transmitters": [
    "db0abc" : {
      "onair": true,
      "node": {
        "name": "db0xyz",
        "ip": "44.42.23.8",
        "port": 1234,
        "connected": true,
        "connected_since": "2018-07-03T08:00:52.786458Z"
      },
      "ntp": {
        "syncd": true
      },
      "messages": {
        "queued": [123, 123, 123, 123, 123, 123],
        "sent": [123, 123, 123, 123, 123, 123]
      },
      "config": {
        "ip": "123.4.3.2",
        "timeslots": [true, false, ..., false],
        "software": {
          name: "Unipager" | "MMDVM" | "DAPNET-Proxy",

```

```

        version: "v1.2.3", | "20180504" | "v2.3.4"
    },
    "proxy" : {
        "status": "connected" | "connecting" | "disconnected"
    }
},
"db0xyz" : {
    "onair": true,
    "node": {
        ....
    }
},
"current_timeslot" : 12
}

```

## 6.7.2 Telemetry from Transmitters - Details of Transmitter

URL: `ws://FQDN/telemetry/transmitters/<transmittername>`

The data is the same as received from the **dapnet.telemetry** exchange from the RabbitMQ instance. It is defined in section 6.4.

The websocket-Server gives out all the telemetry data from a certain transmitter. The name of the transmitter obtained from the RabbitMQ routing key.

```

{
  "onair": true,
  "node": {
    "ip": "44.42.23.8",
    "port": 1234,
    "connected": true,
    "connected_since": "2018-07-03T08:00:52.786458Z"
  },
  "ntp": {
    "syncd": true,
    "offset": 124,
    "server": ["134.130.4.1", "12.2.3.2"],
  },
  "messages": {
    "queued": [123, 123, 123, 123, 123, 123],
    "sent": [123, 123, 123, 123, 123, 123]
  },
  "temperatures": {
    "unit": "C" | "F" | "K",
    "air_inlet": 12.2,
    "air_outlet": 14.2,
    "transmitter": 42.2,
    "power_amplifier": 45.2,
    "cpu": 93.2,
    "power_supply": 32.4,
    "custom": [
      {"value": 12.2, "description": "Aircon Inlet"},
      {"value": 16.2, "description": "Aircon Outlet"},
      {"value": 12.3, "description": "Fridge Next to Programmer"}
    ]
  },
  "power_supply": {
    "on_battery": false,
    "on_emergency_power": false,
    "dc_input_voltage": 12.4,
    "dc_input_current": 3.23
  },
  "rf_output" : {
    "fwd": 12.2,
    "refl" : 12.2,
    "vswr" : 1.2
  },
  "config": {
    "ip": "123.4.3.2",
    "timeslots" : [true, false, ..., false],
  }
}

```

```

    "software": {
      name: "Unipager" | "MMDVM" | "DAPNET-Proxy",
      version: "v1.2.3", | "20180504" | "v2.3.4",
    },
  },
  "hardware": {
    "platform": "Raspberry Pi 3B+"
  },
  "rf_hardware": {
    "c9000": {
      "name" : "C9000 Compact",
      "<pa_dummy>" : {
        "output_power" : 123,
        "port" : "/dev/ttyUSB0"
      }
      "<rpc>": {
        "version" : "X0S/2.23pre"
      }
    },
    "rasp pager": {
      "name": "Rasp pager",
      "modulation": 13,
      "power": 63,
      "external_pa": false,
      "version": "V2"
    },
    "audio": {
      "name" = "Audio",
      "transmitter": "GM1200" | "T7F" | "GM340" | "FREITEXT",
      "audio_level": 83,
      "tx_delay": 3
    },
    "rfm69": {
      "name" : "RFM69",
      "port": "/dev/ttyUSB0"
    },
    "mmdvm": {
      "name" : "MMDVM",
      "dapnet_exclusive": true
    }
  },
  "proxy" : {
    "status": "connected" | "connecting" | "disconnected"
  }
}

```

### 6.7.3 Telemetry from Nodes - Summary of all Nodes

URL: ws://FQDN/telemetry/nodes

The websocket-Server generates an array of JSON Objects which have the name of the node obtained from the RabbitMQ routing key.

```

{
  "nodes" : [
    "db0sda" : {
      "good_health" : true,
      "connections" : {
        "transmitters" : 123,
        "third_party" : 3
      },
      "system" : {
        "is_hamcloud" : false
      }
    },
    "hamcloud1" : {
      "good_health" : true,
      "connections" : {
        "transmitters" : 658,

```

```

        "third_party" : 25
    },
    "system" : {
        "is_hamcloud" : true
    }
},
....
]
}

```

#### 6.7.4 Telemetry from Transmitters - Details of Node

URL: `ws://FQDN/telemetry/nodes/<nodename>`

The data is the same as received from the **dapnet.telemetry** exchange from the RabbitMQ instance. It is defined in section 6.4.

The websocket-Server gives out all the telemetry data from a certain node. The name of the transmitter obtained from the RabbitMQ routing key.

```

{
  "good_health" : true,
  "microservices" : {
    "database" : {
      "ok" : true,
      "version" : "1.2.3"
    },
    "call" : {
      "ok" : true,
      "version" : "1.2.3"
    },
    "rubric" : {
      "ok" : true,
      "version" : "1.2.3"
    },
    "transmitter" : {
      "ok" : true,
      "version" : "1.2.3"
    },
    "cluster" : {
      "ok" : true,
      "version" : "1.2.3"
    },
    "telemetry" : {
      "ok" : true,
      "version" : "1.2.3"
    },
    "database-changes" : {
      "ok" : true,
      "version" : "1.2.3"
    },
    "statistics" : {
      "ok" : true,
      "version" : "1.2.3"
    },
    "rabbitmq" : {
      "ok" : true,
      "version" : "1.2.3"
    },
    "thirdparty" : {
      "ok" : true,
      "version" : "1.2.3"
    },
  },
  "connections" : {
    "transmitters" : 123,
    "third_party" : 3
  },
  "system" : {
    "free_disk_space_mb": 1234
  }
}

```

```

    "cpu_utilization": 0.2
    "is_hamcloud" : false
  }
}

```

## 6.7.5 Database Changes

URL: ws://FQDN/changes

To inform the website or the app about changes in the CouchDB database, the websocket microservice keeps a connection to the local CouchDB API and receives a stream of updates to the database. As there may be data in the changes that are confidential, the stream is parsed and sent out in a reduced form to the websocket client. Further information: <http://docs.couchdb.org/en/2.0.0/api/database/changes.html>

The format of the updates is:

define/review  
format

### 6.7.5.1 Transmitter related

New transmitter added

```

{
  "type": "transmitter",
  "action" : "added",
  "name": "db0abc",
  "data" : {
    (Data from CouchDB Change feed in processed way)
  }
}

```

Existing transmitter changed

```

{
  "type": "transmitter",
  "action" : "changed",
  "name": "db0abc",
  "data" : {
    (Data from CouchDB Change feed in processed way)
  }
}

```

Transmitter deleted

```

{
  "type": "transmitter",
  "action" : "deleted",
  "name": "db0abc"
}

```

### 6.7.5.2 User related

New User added

```

{
  "type": "user",
  "action" : "added",
  "name": "db1abc",
  "data" : {
    (Data from CouchDB Change feed in processed way)
  }
}

```

Existing user changed



```
{
  "type": "user",
  "action": "changed",
  "name": "db1abc",
  "data": {
    (Data from CouchDB Change feed in processed way)
  }
}
```

User deleted

```
{
  "type": "user",
  "action": "deleted",
  "name": "db1abc"
}
```

### 6.7.5.3 Rubric related

New Rubric added

```
{
  "type": "rubric",
  "action": "added",
  "id": "...",
  "data": {
    (Data from CouchDB Change feed in processed way)
  }
}
```

Existing rubric changed

```
{
  "type": "user",
  "action": "changed",
  "id": "...",
  "data": {
    (Data from CouchDB Change feed in processed way)
  }
}
```

Rubric deleted

```
{
  "type": "user",
  "action": "deleted",
  "id": "..."
}
```

### 6.7.5.4 Rubric content related

New Rubric content added

```
{
  "type": "rubric_content",
  "action": "added",
  "id": "...??",
  "data": {
    (Complete Data dump of all ten rubric messages as stored in CouchDB)
  }
}
```

Existing rubric changed

```
{
  "type": "rubric_content",
  "action": "changed",
  "id": "...",
  "data": {
```

Check  
against  
CouchDB  
structure

```

    (Complete Data dump of all ten rubric messages as stored in CouchDB)
  }
}

```

Rubric content deleted

```

{
  "type": "rubric_content",
  "action" : "deleted",
  "id": "...",
  "data" : {
    (Complete Data dump of all ten rubric messages as stored in CouchDB, some may be empty)
  }
}

```

### 6.7.5.5 Node related

New node added

```

{
  "type": "node",
  "action" : "added",
  "name": "db0abc",
  "data" : {
    (Data from CouchDB Change feed in processed way)
  }
}

```

Existing node changed

```

{
  "type": "node",
  "action" : "changed",
  "name": "db0abc",
  "data" : {
    (Data from CouchDB Change feed in processed way)
  }
}

```

Node deleted

```

{
  "type": "node",
  "action" : "deleted",
  "name": "db1abc"
}

```

## 6.8 CouchDB Documents and Structure

als  
Tabelle  
darstellen

### 6.8.1 Users

```

{
  "_id": "dliabc",
  "password": "<bcrypt hash>",
  "email": "dliabc@darcs.de",
  "roles": "admin",
  "enabled": true,
  "created_on": "2018-07-03T08:00:52.786458Z",
  "created_by": "dh3wr",
  "changed_on": "2018-07-03T08:00:52.786458Z",
  "changed_by": "dh3wr",
  "email_valid": true
  "avatar_picture": <couchdb attachment>
}

```

Table 6.17: CouchDB: Users

Key	Value-Type	Valid Value Range	Example
_id	string		dl1abc
password	string	bcrypt hash	—
email	string		dl1abc@dar
role	string	"admin" "support" "user" "thirdparty.[aprs brandmeister]"	true
enabled	boolean		true
created_on	string	ISO8601	2018-07-08T
changed_on	string	ISO8601	2018-07-08T
changed_by	string	valid user name	dh3wr
email_valid	boolean		true
avatar_picture	couchdb_attachment		

## 6.8.2 Nodes

Table 6.18: CouchDB: Nodes

Key	Value-Type	Valid Value Range	Example
_id	STRING	N/A	db0abc
coordinates	[number; 2]	[lat, lon]	[34.123456, 6.23144]
description	string	whatever	Aachen, Germany
hamcloud	boolean	true/false	true
created_on	string	ISO8601	2018-07-08T11:50:02.168325Z
changed_by	string	valid user name	dh3wr
changed_on	string	ISO8601	2018-07-08T11:50:02.168325Z
changed_by	string	valid user name	dh3wr
owners	[string]	N/A	["dl1abc", "dh3wr", "dl2ic"]
avatar_picture	couchdb_attachment		

```
{
  "_id": "db0abc",
  "auth_key": "super_secret_key",
  "coordinates": [34.123456, -23.123456],
  "description": "some words about that node",
  "hamcloud": true,
  "created_on": "2018-07-03T08:00:52.786458Z",
  "created_by": "dh3wr",
  "changed_on": "2018-07-03T08:00:52.786458Z",
  "changed_by": "dh3wr",
  "owners": ["dl1abc", "dh3wr", "dl2ic"],
  "avatar_picture": <couchdb attachment??>
}
```

Wofür genau braucht man email\_valid?  
- Um ab und zu mal eine Testmail an die User zu schicken, ob sie unter der Email noch erreichbar sind und sonst sie zu löschen.

## 6.8.3 Transmitters

```
{
  "_id": "db0abc",
  "auth_key": "hdjaskhdlj",
  "enabled": true,
  "usage": "personal" | "widerange",
  "coordinates": [34.123456, -23.123456],
  "power": 12.3,
  "antenna": {
    "agl": 23.4,
    "gain": 2.34,
    "type": "omni" | "directional",
    "direction": 123.2,
    "cable_loss": 4.2
  }
}
```

Tabelle weiter machen

Table 6.19: CouchDB: Transmitters

Key	Value-Type	Valid Value Range	Example
_id	string	N/A	db0abc
auth_key	string	N/A	asd2FD3q3rF
enabled	boolean	true/false	true
usage	string	PERSONAL   WIDERANGE	WIDERANGE
coordinates	[number; 2]	[lat, lon]	[34.123456, 6.23144]
power	number	0.001 ...	12.3
created_on	string	ISO8601	2018-07-08T11:50:02.168325Z
changed_by	string	valid user name	dh3wr
changed_on	string	ISO8601	2018-07-08T11:50:02.168325Z
changed_by	string	valid user name	dh3wr
owners	ARRAY of STRING	N/A	["dl1abc", "dh3wr", "dl2ic"]
avatar_picture	couchdb_attachment		

```

}
"owners" : ["dl1abc", "dh3wr", "dl2ic"],
"groups" : ["dl-hh", "dl-all"],
"emergency_power": {
  "available": false,
  "infinite": false,
  "duration": 23*60*60 // seconds
},
"created_on": "2018-07-03T08:00:52.786458Z",
"created_by": "dh3wr",
"changed_on": "2018-07-03T08:00:52.786458Z",
"changed_by": "dh3wr",
"aprs_broadcast": false,
"antenna_pattern" : <couchDB attachment>,
"avatar_picture" : <couchDB attachment>
}

```

## 6.8.4 Subscribers

If type is "Skyper", function is always 3. Keep this in mind

check if  
[] is valid  
JSON

```

{
  "_id" : "dl1abc",
  "description" : "Peter",
  "pagers" : [
    {
      "ric": 123456,
      "function": 0 .. 3,
      "name": "Peters Alphapoc",
      "type" : "UNKNOWN" | "Skyper" | "AlphaPoc" | "QUIX" | "Swissphone" | "SCALL_XT" | "Birdy"
      "enabled" : true
    },
    ...
  ],
  "third_party_services" : ["APRS", "BM"],
  "owner": ["dh3wr", "dl1abc"],
  "groups" : ["rwth-afu"]
}

```

## 6.8.5 Rubrics

```

{
  "_id": "wx-dl-hh"
  "number": 14,
  "description": "Wetter DL-HH",
  "label": "WX DL-HH",
  "transmitter_groups": ["dl-hh", "dl-ns"],
  "transmitters": ["db0abc"],
  "cyclic_transmit": true,

```

```

    "cyclic_transmit_interval": 3600, // seconds
    "owner" : ["dh3wr", "dl1abc"]
}

```

### 6.8.6 Rubric's content

<UUID> of rubric (as defined in ??)

```

{
  "_id" : "<UUID>",
  "rubric": "wx-dl-hh",
  "content": [
    "message1",
    ..,
    "message10"
  ],
}

```

### 6.8.7 MQTT services and subscribers

```

{
  "_id": "APRS",
  "topic": "aprs"
  "subscribers": [
    {
      "name": "example",
      "password": "<bencrypt hash>"
    },
    ...
  ]
}

```