User Manual

Radio Direction Finder Standard JSON Protocol

RHO THETA
Elektronik GmbH

Edited by:

RHOTHETA Elektronik GmbH
Kemmelpark
Dr.-Ingeborg-Haeckel-Str. 2
82418 Murnau
Germany

Tel.: +49 8841 4879 - 0
Fax: +49 8841 4879 - 15

Internet: www.rhotheta.de
E-Mail: email@rhotheta.de

- *Issue: 2023/03/15 [Rev 3.00]*
- *Document-ID: 12-9-4-5-00009-60*

Note

The manufacturer reserves the right to make modifications at any time and without previous information of the here described product.

Index

1 Change Log

Revision	Date	Changes
1.05	08.04.2021	First release
1.06	15.04.2021	Add utc to dfSystemPositionUpdate message
1.07	26.04.2021	Integrate Triangulator Id and Triangulator Status
1.08	28.04.2021	Include Enabled – Key into triangulator status object
1.09	10.06.2021	Change responses, Add additional antenna attenuation
1.10	21.06.2021	Bugfixes and Typos
2.00	30.03.2022	Integrate Operating Mode, Valid Bearing Sector and Sector Blanking
2.00.a	07.04.2022	Spelling Corrections
2.00.b	30.09.2022	Spelling Corrections
3.00	15.03.2023	Integrate Server-Client features

2 Overview

2.1 DF Service

Radio Direction Finder Standard JSON Protocol represents an interface specification to Direction Finder (DF) Service for clients. According to the following figure the DF Service is an application which establishes a network connection to different direction finder systems, processes DF data and provides an interface to multiple clients using internal server.

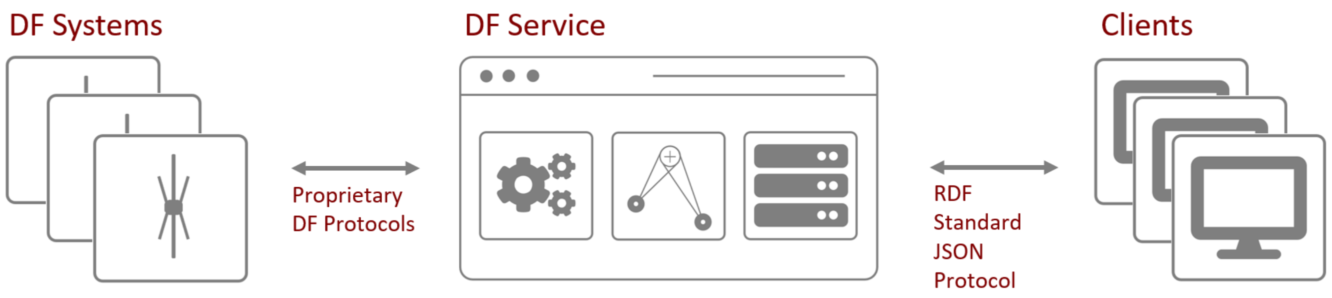


Figure 1: DF Service Overview

The following description should provide a better understanding of the presented network components.

<u>DF Systems</u>

Direction Finder Systems should be understood as real hardware devices, which provide their data via TCP/IP interface. Per definition one DF System has one direction finder antenna. The received signal from the antenna is then processed by so called DF Channels. DF System can contain one to multiple DF channels which are assigned to desired frequency. This concept allows simultaneous radio direction finding on different frequencies.

DF Systems also have to be understood as systems with flexible position and antenna orientation. Beside the applications with fixed installed DF Systems such as Vessel Traffic Service or Air Traffic Control, direction finder systems can be used on aircrafts, vessels, cars, drones and can also be carried by people. Due to this fact DF Systems can not only provide the bearing information on different frequencies, but also their position data such as latitude, longitude, heading and speed. Therefore, DF Systems can also contain or have an interface to additional devices such as GPS, compass, or other available sensors.

In RDF Standard JSON Protocol every DF System and every DF Channel has a unique ID, which is either provided from a DF System or automatically generated, when the corresponding software entity is created.

<u>DF Service</u>

As previously described, the DF Service establishes the network connection to multiple DF Systems. It allows configuration and technical monitoring of all connected systems, their DF channels and corresponding sensor devices. Regardless which RHOTHETA direction finder type is connected to DF Service, the corresponding DF System can be controlled via the same RDF Standard JSON Protocol.

DF Service also supports triangulation from two or three different DF Systems on any desired frequency. When the internal triangulator has a result, the corresponding data is sent to all clients connected.
Finally, DF Service provides a TCP/IP server, which allows connections of multiple clients. When a client connects to the server, it gets the whole configuration and status information immediately after a successful TCP connection. Clients can also send commands to DF service. In this case, the command response is sent out immediately to a client which has commanded the system. The response is either an "accept message" or an "error message". In case of an error message the command was not in the correct format. In case of an accept message, the DF Service has accepted this command and will process it. It does not mean that the command is already executed. If the command has successfully changed the system configuration, the status information is broadcasted immediately after the command execution to all connected clients.

Clients

Clients must be understood as all possible applications, which process direction finder data and triangulation results. The following applications come into question:
- Human machine interfaces (indication of bearing and triangulation results on map)
- Technical monitoring applications
- Data logging applications
- Protocol converter / adapter

2.2 Data Structure

The protocol is based on JSON (Java Script Object Notation) data format, which is an open standard format used in data communication. JSON is available as a human-readable text, which provides accessible approach to current state and data. JSON is also widely supported by the most modern programming languages like C++, C#, Java, Java Script, Python etc.

In general, the protocol messages are separated into three different types:

- output
- command
- error and command response

All messages are laid out in the same template form, a JSON array consisting of **event identifier** and **JSON object with details**.

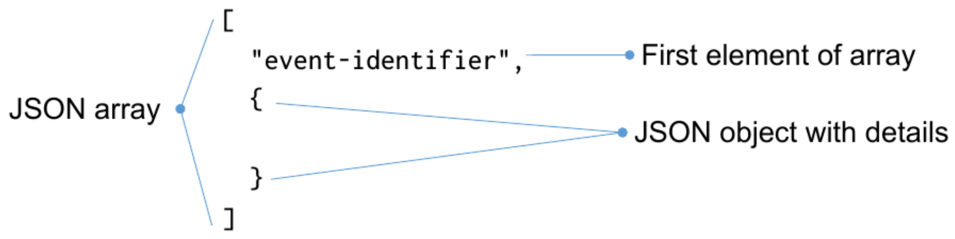


Figure 2: General Protocol Pattern

The "event-identifier" value is always written in camel case, as well as the keys in JSON object. Camel case example: "thisIstheTextWrittenInCamelCase".

2.3 Data Stream

To have a steady stream of data and keep the integrity and encapsulation, a JSON variant called NDJSON[1] is used.

This simplifies the process of receiving and extracting the data from the JSON message, even with problems in bandwidth and traffic congestion.

Each message is represented as one-line JSON, with '\n' following after the end of the message. This would mean each general RHOTHETA protocol JSON array will be in one line, separated with line separator '\n'.

Example 1:
One general RHOTHETA protocol message sent over TCP as NDJSON.

```
["updateTriangulator",{"en":true}]
```

Example 2:
Multiple general RHOTHETA protocol messages sent over TCP as NDJSON.

```
["createDfSystem",{}]
["createDfChannel",{"sysId":"51ccfeaf-f0b7-480c-957e-613661bd9034"}]
```

2.4 Protocol Layer

DF messages can be set on different protocol layers which are then provided by corresponding DF hardware or software system. Please refer to corresponding manual of your system in use. Typical layers are TCP and WebSocket.

2.5 TCP Server

Connection to the server can be established by using IP address of the machine running the server and defined port.
Default port setting is 9999.
Note: If you use a firewall, please allow connection to the specified port.

Example:
If the system is installed on a local PC where client is also running, use IP Address: 127.0.0.1 and port: 9999.

When the client is connected, TCP Server will immediately provide statuses of all the available DF Systems.

Port setting can be changed using a configuration file located in local system storage of the server. See documentation of the appropriate DF Service for details of the configuration files and their locations.

If GUI Application is available, please see corresponding manual on how to use.

[1]NDJSON - Newline Delimited JSON (http://ndjson.org/)

3 Output Messages

Output messages are sent by the DF Service with one of these triggers:

- predefined period of time:
    - Bearing
    - System position
    - System status
    - Triangulator status

- when calculated:
    - CP-SS
    - Triangulation

These messages are automatically sent without the need to configure them initially.

Two different types of output messages are available:

- Measured data:
    - Bearing
    - CP-SS
    - System position
    - Triangulation

- Status data:
    - System status
    - Triangulator status
    - Server Status
    - Client Status
    - Client Connections

3.1 Bearing Message

The bearing message is a JSON object representing the measured data for DF Channel. This object includes all the data relevant for the current bearing data measurement.

First array element: **“bearing”**.
Second array element: JSON Object with bearing data.

JSON Object with bearing data includes:

- **sysId** *[string]* — Unique DF System identification
- **chId** *[string]* — Unique DF Channel identification
- **freq** *[number]* — Frequency of the specified DF Channel
  Default: null
- **sq** *[number]* — Squelch value of the specified DF Channel
  Default: null
- **sqdBm** *[number]* — Squelch value in dBm of the specified DF Channel
  Default: null
- **sqdBuV** *[number]* — Squelch value in dBµV of the specified DF Channel
  Default: null
- **sqdBuVm** *[number]* — Squelch value in dBµV/m of the specified DF Channel
  Default: null
- **a** *[boolean]* — Signal active indicator:
  - True, if bearing is available.
  - If false, bearing is not available or invalid.
    Default: false
- **sbs** *[boolean]* — Self Bearing Suppression indicator:
  - If true, suppression is active.
- **rb** *[number]* — Relative bearing:
  - Valid: [0 - 359.99]
  - If null, bearing is not available.
    Default: null
- **mb** *[number]* — Magnetic bearing:
  - Calculated from relative bearing and heading.
  - Valid: [0 - 359.99]
  - If null, magnetic bearing is not available.
    Default: null
- **tb** *[number]* — True bearing:
  - Calculated from relative bearing and heading.
  - Valid: [0 - 359.99]

  - If null, true bearing is not available.
    Default: null

- **rbL** *[number]* Live relative bearing:
  - Valid: [0 - 359.99]
  - If null, bearing is not available.
    Default: null

- **rbLmax** *[number]* Maximum live relative bearing:
  - Valid: [0 - 359.99]
  - If null, bearing is not available.
    Default: null

- **rbLmin** *[number]* Minimum live relative bearing:
  - Valid: [0 - 359.99]
  - If null, bearing is not available.
    Default: null

- **sl** *[number]* Signal level of the specified frequency:
  - Valid: [0 - 100]
  - If null, signal level is not available.
  - Unit: %
    Default: null

- **sldBm** *[number]* Signal level of the specified frequency in dBm:
  - If null, signal level is not available.
  - Unit: dBm
    Default: null

- **sldBuV** *[number]* Signal level of the specified frequency in dBµV:
  - If null, signal level is not available.
  - Unit: dBµV
    Default: null

- **sldBuVm** *[number]* Signal level of the specified frequency in dBµV/m:
  - If null, signal level is not available.
  - Unit: dBµV/m
    Default: null

- **sd** *[number]* Standard deviation
  Default: 1.0

- **lat** *[number]* Antenna's latitude:
  - Valid: [-90 - 90]
  - If null, Antenna's position is not set.
  - Format: Decimal degrees
  - *Convenient information (also available in DF System's Position)*
    Default: null

- **lon** *[number]* Antenna's longitude:

  - Valid: [-180 - 180]
  - If null, Antenna’s position is not set.
  - Format: Decimal degrees
  - *Convenient information (also available in DF System’s Position)*
    Default: null

- **utc** *[string]* UTC date and time of the DF System:
  - Format: ISO8601
  - If null, date and time are not available.
    Default: null

Timing:
The bearing message is transmitted whenever values in bearing data have been changed, providing Antenna’s bearing data for selected frequency on each cycle.

Cycle speed depends on the DF System in use but is never less than 50 ms.

Example:
```
[
    "bearing",
    {
        "a": true,
        "chId": "13f80eb7-a9df-4998-97d8-e51f84888ac3",
        "freq": 156800000,
        "lat": 54.485947,
        "lon": 11.163944,
        "mb": 35,
        "rb": 45,
        "rbL": null,
        "rbLmax": null,
        "rbLmin": null,
        "sbs": false,
        "sd": 1,
        "sl": 40,
        "sldBm": -114,
        "sldBuV": -7,
        "sldBuVm": 12,
        "sq": 20,
        "sqdBm": -130,
        "sqdBuV": -23,
        "sqdBuVm": -4,
        "sysId": "51ccfeaf-f0b7-480c-957e-613661bd9034",
        "tb": 45,
        "utc": "2021-06-10T14:07:30.380Z"
    }
]
```

3.2 CP-SS Message

The CP-SS message is a JSON object representing the measured data of COSPAS-SARSAT signal.

It contains data received from COSPAS-SARSAT beacon and is only available when COSPAS-SARSAT signal arrives.

First array element: **“cpss”**.
Second array element: JSON Object with COSPAS-SARSAT data.

JSON Object with COSPAS-SARSAT data includes:

◦ **sysId** *[string]*	Unique DF System identification
◦ **chId** *[string]*	Unique DF Channel identification
◦ **sysName** *[string]*	DF System name
◦ **chName** *[string]*	DF Channel name
◦ **bId** *[string]*	Beacon’s ID
◦ **prot** *[string]*	Protocol of the COSPAS-SARSAT message
◦ **lat** *[number]*	Beacon’s latitude: ▪ Valid: [-90 - 90] ▪ If null, beacon’s position is not available. ▪ Format: Decimal degrees ***Default: null***
◦ **lon** *[number]*	Beacon’s longitude: ▪ Valid: [-180 - 180] ▪ If null, beacon’s position is not available. ▪ Format: Decimal degrees ***Default: null***
◦ **cCode** *[number]*	Beacon’s country code: ▪ Three-digit number ▪ If null, country code is not available. ***Default: null***
◦ ***country****[string]*	Beacon’s country ***Default: undefined***
◦ **freq** *[number]*	Frequency on which message was received. ***Default: null***
◦ ***hex*** *[string]*	Full raw message in hex format
◦ ***selfTest*** *[bool]*	Indication if beacon is in self test mode.

- **sysLat** *[number]* Df System's latitude:
  - Valid: [-90 - 90]
  - If null, Df System's position is not available.
  - Format: Decimal degrees

  Default: null

- **sysLon** *[number]* Df System's longitude:
  - Valid: [-180 - 180]
  - If null, Df System's position is not available.
  - Format: Decimal degrees

  Default: null

- **sd** *[number]* Standard deviation:
  - If null, standard deviation is not available.

  Default: 1.0

- **tb** *[number]* True bearing:
  - True bearing of the signal
  - Valid: [0 - 359.99]
  - If null, true bearing is not available.

  Default: null

- **utc** *[string]* UTC date and time of the DF System:
  - Format: ISO8601
  - If null, date and time are not available.

  Default: null

Optional (will be present if available)

- **mmsi** *[string]* Beacon's MMSI ID

- **dst** *[number]* Distance from DF System to Beacon's location:
  - Unit: meters

- **emcode** *[number]* Emergency code

Timing:
COSPAS-SARSAT message is provided when COSPAS-SARSAT message has been successfully decoded by the DF System.

3.3 Triangulation

3.3.1 Triangulation Result Messages

The triangulation result message is a JSON object representing the computed cross-bearing data, using bearing from multiple DF Systems.
This object includes computation info and resulting position.

The triangulation message is a JSON array, which starts with “**triangulation**”.
The second object is a JSON object with details.

First array element: **“triangulation”**.
Second array element: JSON Object with triangulation data.

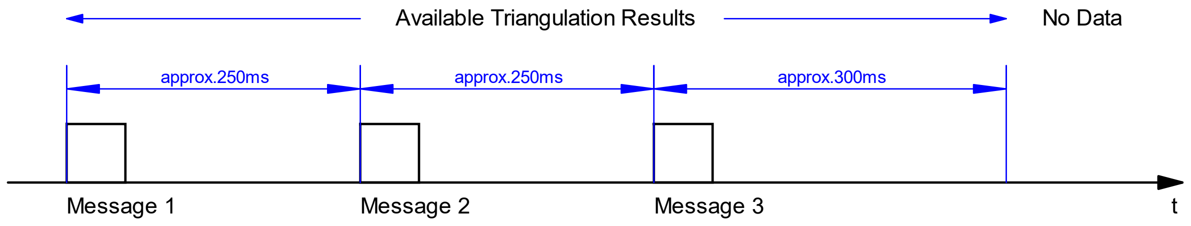
JSON Object with triangulation data includes:

- **triangulatorId** *[string]* Generated unique ID of the triangulator.
- **utc** *[string]* UTC date and time of the DF System
  - Format: ISO8601
- **freq** *[number]* Frequency on which the triangulation is calculated in Hz.
- **lat** *[number]* Latitude of center position:
  - Valid: [-90 … 90]
  - Format: Decimal degrees
- **lon** *[number]* Longitude of center position:
  - Valid: [-180 … 180]
  - Format: Decimal degrees

Example:

```
[
    "triangulation",
    {
        "freq": 156525000,
        "lat": 54.42456,
        "lon": 11.448699999999999,
        "triangulatorId": "771fc48e-a533-4a7a-aef6-47d173759939",
        "utc": "2021-06-10T16:30:23.950"
    }
]
```

Timing:
The triangulation result messages are transmitted only if the triangulator system has the appropriate result. No transmission is occurred otherwise.
When the triangulator has calculated positions, messages are transmitted every 250 ms.



3.3.2 Triangulator Status Messages

The triangulator status messages contain status and settings of the triangulator system.

The triangulator status message is a JSON array which starts with "**triangulatorStatus**". The second object is JSON object with details.

JSON Object with triangulator status data includes:

- **triangulatorId** *[string]* Generated unique ID for the triangulator.

- **en** *[boolean]* Flag which represents if the triangulator is switched ON or OFF.
  - If true, the triangulator is ON.
  - If false, the triangulator is OFF.

- **sectorBlankingActive** *[boolean]* Flag which represents if sector blanking is switched ON or OFF.
  - If true, sector blanking is ON.
  - If false, sector blanking is OFF.

- **generalState** *[string]* Current DF Channel's general state (s. also Chapter 5 "Device States"). This state can be "OFF", "ERROR", "WARNING", "OK".

"OFF" – triangulator is disabled.

"ERROR" – state ERROR will be set, if triangulator is not able to generate triangulation results. This can be the case, if:

- Frequency list is empty.
- DF System list is empty.
- Only one or no DF System is configured.
- DF systems configured for triangulation are in ERROR state.
- Frequencies from the array "frequencies" are not configured in DF systems or only one DF system contains the frequency from the list.
  Example: "ERROR" state
  Adjusted frequencies: 156.8 MHz, 121.5 MHz, 156.525 MHz

DF System 1	DF System 2	DF System 3	Note
156.8 MHz	120.0 MHz	120.0 MHz	Only one frequency is configured
121.5 MHz	121.5 MHz	121.5 MHz	Ok
156.0 MHz	156.0 MHz	156.0 MHz	156.525 MHz is not configured at all

"WARNING" – state WARNING will be set, if triangulator is still able to generate triangulation result, but with limitations. This can be the case, if:

- Triangulator is operating in Test Mode
- One of the configured DF Systems is in ERROR state, the triangulation can still be calculated based on two DF Systems.
- One or more DF Systems are in "WARNING" state.
- Frequencies from the array "frequencies" are partly configured in DF systems.
  Example: "WARNING" state
  Adjusted frequencies: 156.8 MHz, 121.5 MHz, 156.525 MHz

DF System 1	DF System 2	DF System 3	Note
156.8 MHz	156.8 MHz	120.0 MHz	The configuration of the third frequency is missing
121.5 MHz	121.5 MHz	121.5 MHz	Ok
156.525 MHz	156.525 MHz	156.525 MHz	Ok

"OK" – state will be set if all configured DF systems are in OK state and all frequencies from the array "frequencies" are also configured in all DF Systems.

- **state** *[string]* Current Triangulator state as a human readable message

- **serverName** *[string]* Name of the server to which Triangulator belongs to

- **triangulatorName** *[string]* Name of the Triangulator

- **radius** *[number]* Radius defines the area around every DF station where the triangulation is possible. (If the distance between two DF stations is higher than this parameter, no triangulation results will be possible). Format: in meter.

- **testMode** *[boolean]* Flag which indicates if the triangulator is in test mode. In test mode, the triangulator additionally produces test triangulation results on four different frequencies.
  - If true, the triangulator is in test mode.
  - If false, the triangulator is not in test mode.

- **frequencies** *[array]* Contains the list of frequencies which are used for triangulation. Other frequencies are ignored. The empty list results in an "ERROR" state.
  - Format: [number]: Array elements are numbers.

- **systems** *[array]* Contains the list of DF system-Ids which are used for triangulation. Other system-Ids are ignored. The empty list results in an "ERROR" state.
  - Format: [string]: Array elements are strings.

Example:

```
[
    "triangulatorStatus",
    {
        "en": true,
        "frequencies": [
            156800000,
            156525000,
            121500000
        ],
        "generalState": "OK",
        "radius": 50000,
        "serverName": "TestServer",
        "state": "OK",
        "sectorBlankingActive": true,
        "systems": [
            "1111",
            "2222",
            "3333"
        ],
        "testMode": false,
        "triangulatorId": "771fc48e-a533-4a7a-aef6-47d173759939",
        "triangulatorName": "TestTriangulator"
    }
]
```

Timing:
Triangulator status messages are transmitted approx. every 5 seconds. After a command sent to triangulator, the status messages are transmitted immediately after the command execution or status change. Time-Out Error can be indicated if messages do not repeat after 15 seconds.

3.4 System Position Message

The System Position message is a JSON object representing data of DF System's position, speed and orientation.

All these DF System's parameters are equal to its Antenna parameters. Settings for the parameters are available as part of DF System's settings of Antenna.

System Position JSON array is defined as:

First element: **"dfSystemPositionUpdate".**
Second element: DF System Position's JSON object.

System Position's JSON object includes:

- **sysId** *[string]* Unique DF System identification

- **lat** *[number]* Antenna's latitude:
  - Valid: [-90 - 90]
  - If null, Antenna's position is not set.
  - Format: Decimal degrees

    Default: null

- **lon** *[number]* Antenna's longitude:
  - Valid: [-180 - 180]
  - If null, Antenna's position is not set.
  - Format: Decimal degrees

    Default: null

- **alt** *[number]* Current Antenna's altitude:
  - Unit: Meters

    Default: null

- **var** *[number]* Current Antenna's variation:
  - Unit: Degrees

    Default: null

- **hdt** *[number]* Current Antenna's true heading:
  - Valid: [0 - 359.99]
  - Unit: Degrees

    Default: null

- **hdm** *[number]* Current Antenna's magnetic heading:
  - Valid: [0 - 359.99]
  - Unit: Decimal degrees

    Default: null

- **rh** *[number]* Radio Horizon based on antenna height and specified transmitter height (see Settings of transmitter height in the Antenna).
  - Unit: meters

Default: null

- **sog** *[number]* Current GPS speed over ground:
  - Unit: Knots
  - If null, SOG is not available.

- **cog** *[number]* Current GPS course over ground:
  - Valid: [0 - 359.99]
  - Unit: Degrees
  - If null, COG is not available.

- **utc** *[string]* UTC date and time of the DF System
  - Format: ISO8601

Example:

```
[
    "dfSystemPositionUpdate",
    {
        "alt": 40,
        "cog": 29,
        "hdm": 19,
        "hdt": 29,
        "lat": 54.57633333333333,
        "lon": 8.5573333333333334,
        "rh": 35098.555521129856,
        "sog": 23,
        "sysId": "51ccfeaf-f0b7-480c-957e-613661bd9034",
        "utc": "2021-06-10T16:24:48.449Z",
        "var": 10
    }
]
```

Timing:
If the data is unchanged, the message is transmitted every second. Whenever new values are available (from GPS or Compass), message is emitted, but no faster than 100 ms.

Data is also transmitted immediately after the change of antenna settings.

3.5 System Status Message

The System Status message is a JSON object representing the specific DF System's state. It includes the Antenna data, GPS and all DF Channels inside this DF System.

This message is sent in specified interval.
Default interval is 1 second.
Details in System Status message are separated in multiple JSON objects, defining each DF System's element status and data.

First element: **"dfSystemUpdate"**
Second element: DF System's JSON object

DF System's JSON object includes:

◦ **sysId** *[string]*	Unique DF System identification
◦ **name** *[string]*	DF System's name
◦ **serverName** *[string]*	Name of the server to which Df System belongs to.
◦ **state** *[string]*	Current DF System's state as a human readable message
◦ **stateInt** *[number]*	Current DF System's state number (s. Chapter 5 "Device States")
◦ **generalState** *[string]*	Current DF System's general state (s. Chapter 5 "Device States")
◦ **utcSource** *[string]*	Source of UTC date and time. Valid options: ▪ "Local Machine" - Use date and time from the Local Machine (default) ▪ "GPS"- Use date and time from GPS device
◦ **antenna** *[JSON Object]*	Antenna Unit parameters (see 3.5.1)
◦ **gps** *[JSON Object]*	GPS parameters (see 3.5.2)
◦ **dfChannels** *[JSON Array]*	Available DF Channels in the DF System (see 3.5.3)
◦ **validBearingMin** *[number]*	Minimum degree of valid bearing sector: ▪ Valid: [0 - 360] ▪ Unit: Decimal Degree ***Default: 0***
◦ **validBearingMax** *[number]*	Maximum degree of valid bearing sector: ▪ Valid: [0 - 360] ▪ Unit: Decimal Degree ***Default: 360***

3.5.1 ANTENNA

The ANTENNA JSON Object represents all the parameters corresponding to current Antenna Unit state.

- **type** *[string]* Type of the Antenna Unit
  Valid options:
  - "RT-1000-ATC" (***default***)
  - "RT-1000-VTS"
  - "RT-500-M"
  - "RT-800"

- **additionalAttenuation** *[number]* Additional attenuation in dB**.** This value is used to correct the antenna gain, which is defined by antenna type. Usually, the additional attenu ation is the cable loss between the antenna and receiver unit.
  - Unit: dB
    Default: 0 dB

- **correction** *[number]* Antenna Unit's mechanical correction:
  - Valid: [-180 - 180]
  - Unit: Decimal Degree
    Default: null

- **upsideDown** *[bool]* Flag for antenna unit mounting:
  - If true, antenna is mounted upside-down.
  - If false, antenna is mounted standard *(**default**).*

- **orientationMode** *[string]* Antenna's orientation mode
  Valid options:
  - "tn" - True north (***default***)
  - "mn" - Magnetic north
  - "hdt" - True heading from Compass
  - "hdm" - Magnetic heading from Compass
  - "cog" - Course over ground from GPS

- **variationSource** *[string]* Source for the Variation of the DF System
  Valid options:
  - "Manual Input" (***default***)
  - "gps"

- **positionSource** *[string]* Source for the position of the DF System
  Valid options:
  - "Manual Input" (***default***)
  - "gps"

- **altitudeSource** *[string]* Source for the altitude of the DF System
  Valid options:
  - "Manual Input" (***default***)
  - "gps"

- **expectedTransmitterHeight** *[number]*
  Expected transmitter height on which radio horizon calculation is based on.
  - Unit: meters
    Default: 0

- **sd** *[number]* Antenna's standard deviation
  Default: 1.0

- **state** *[string]* Current state as a human readable message

- **generalState** *[string]* Current general state (s. Chapter 5 "Device States")

3.5.2 GPS

The GPS JSON Object represents the current GPS state of the specified DF System.

- **state** *[string]* Current GPS's state as a human readable message

- **stateInt** *[number]* Current GPS's state int (s. Chapter 5 "Device States")

- **generalState** *[string]* Current GPS's general state (s. Chapter 5 "Device States")

- **ipAddress** *[string]* GPS IP address setting:
  Default: ""

- **tcpPort** *[string]* GPS TCP port setting
  Default: ""

3.5.3 DF_CHANNELS

DF_CHANNELS is an array representing all DF Channels available in specified DF System. Each DF Channel is presented as a JSON object with following keys:

- <u>**Required (always present)**</u>
  - **chId** *[string]* Unique DF Channel identification

  - **name** *[string]* DF Channel's name
    Default: ""

  - **protocol** *[string]* Protocol type of the DF Channel
    Valid options:
    - "RT-1000" (***default***)
    - "RT-500-M" - NMEA protocol
    - "RT-800" - NMEA protocol
    - "RT-500-M Antenna Unit"
    - "RT-600 Antenna Unit"
    - "RT-800 Antenna Unit"

  - **operatingMode** *[string]* Operating mode of the DF Channel
    Valid options:

    - For "RT-1000" protocol:

	- "Bearing Mode" - "Marine Scan" ▪ For "RT-500-M" and "RT-800" protocol: - "Bearing Mode" - "Marine Scan" - "CP-SS Scan" - "CP-SS Decode Mode" ▪ Unit protocols: - "Bearing Mode" - "Marine Scan" - "CP-SS Scan" - "CP-SS Decode Mode"
◦ **state** *[string]*	Current DF Channel's state as a human readable message
◦ **stateInt** *[number]*	Current DF Channel's state number (s. Chapter 5 "Device States")
◦ **generalState** *[string]*	Current DF Channel's general state (s. Chapter 5 "Device States")
◦ **rackNumber** *[number]*	Channel rack number ***Default: 0***
◦ **freq** *[number]*	DF Channel's Frequency setting ***Default: null***
◦ **sq** *[number]*	DF Channel's Squelch setting ▪ Valid: [0 - 60] ***Default: null***
◦ **sqdBm** *[number]*	DF Channel's Squelch setting in dBm ***Default: null***
◦ **sqdBuV** *[number]*	DF Channel's Squelch setting in dBµV ***Default: null***
◦ **sqdBuVm** *[number]*	DF Channel's Squelch setting in dBµV/m **Default: null**
◦ **ipAddress** *[string]*	DF Channel's IP address setting **Default: ""**
◦ **tcpPort** *[string]*	DF Channel's TCP port setting **Default: ""**

Timing:
This message is broadcasted every 5 seconds or immediately if status data has been changed, to all connected clients.

Status message is also provided on successful connection to the server, but only to the client connected (no broadcast).

Example:

```
[
    "dfSystemUpdate",
    {
        "antenna": {
            "additionalAttenuation": 0,
            "altitudeSource": "Manual Input",
            "correction": 0,
            "expectedTransmitterHeight": 5,
            "generalState": "OK",
            "orientationMode": "tn",
            "positionSource": "Manual Input",
            "sd": 1,
            "state": "OK",
            "type": "RT-500-M",
            "upsideDown": false,
            "variationSource": "Manual Input"
        },
        "dfChannels": [
            {
                "chId": "13f80eb7-a9df-4998-97d8-e51f84888ac3",
                "freq": 156800000,
                "generalState": "OK",
                "ipAddress": "127.0.0.1",
                "name": "APPROACH",
                "operatingMode": "Bearing Mode",
                "protocol": "RT-500-M",
                "rackNumber": 1,
                "sq": 20,
                "sqdBm": -130,
                "sqdBuV": -23,
                "sqdBuVm": -4,
                "state": "OK",
                "stateInt": 9,
                "tcpPort": "60010"
            }
        ],
        "generalState": "OK",
        "gps": {
            "generalState": "OK",
            "ipAddress": "127.0.0.1",
            "state": "OK",
            "stateInt": 9,
            "tcpPort": "3000"
        },
        "name": "DF-SYSTEM",
        "serverName": "REMOTE-1",
        "state": "OK",
        "stateInt": 0,
        "sysId": "51ccfeaf-f0b7-480c-957e-613661bd9034",
        "utcSource": "gps",
        "validBearingMin": 0,
        "validBearingMax": 360
    }
]
```

3.6 Server Status

The Server status message is a JSON object representing the specific server's state.
This message is sent in specified intervals to each connected client.
As default the server will send this message in the interval of 5 seconds to each connected client.

The server status message may be used by clients as a heartbeat from the server to detect connection problems.

The interval can be setup for each connected client separately by sending the "updateServerStatusInterval" command message to the server.

First element: **"serverStatus"**
Second element: The server's status information as a JSON object

Note in advance concerning Host Names and IP addresses:

In networks using the dynamic host configuration protocol (DHCP) the IP address may change each time the computer is reconnected to the network. The IP address therefore cannot be used to unambiguously identify the client's computer. The hostname is unique within sub networks but might still be ambiguous when leaving the local network. But as the host name is more human readable that an IP address and can be easily retrieved as the local host name of the computer it was decided that a client should provide this information.

Server's JSON object includes:

◦ **hostName** *[string]*	HostName of the computer the server is running on (LocalHostName).
◦ **statusMessage** *[string]*	Current server's state as a human readable message. Server message containing additional error information (any which can be retrieved as is meaningful).
◦ **Status** *[string]*	Current server's device state (s. Chapter 5 "Device States")
◦ **name** *[string]*	Limited to 256 characters. Could include everything what might be useful to describe the usage of the server. Usually this corresponds to the usage of the application. For example: "Fehmarn Belt West" For DF Commander MK2 Applications providing a GUI the name must be entered in the Setup Page. Without GUI the name of the client must be defined in the "generalSettings.json" configuration file before starting the application.

__**Example:**__

As heartbeat the server sends the following “serverStatus” message to each connected client.

```
[
    "serverStatus",
      {
        "hostName": "ServerComputerName"
        "name": "Fehrmann Belt West",
        "status": "OK",
        "statusMessage": "OK",
    }
]
```

Example for content of “generalSettings.json” file:

```
{
    "name": "Technical monitoring Luebeck, WST1"
}
```

__**Timing:**__
This message is broadcasted in the defined heartbeat intervals.
Default: 5 seconds

3.7 Client Status

The Client status message is a JSON object representing the specific client’s state.

This message is optional.

If the connection is setup to check for connection timeouts (see “updateClientStatusTimeout” command) the server will rely on that the client’s status message is sent at least once within the specified timeout. If not, the connection is considered to be down and the server will close the connection.

First element: **“clientStatus”**

__Note in advance concerning Host Names and IP addresses:__

See “Note in advance” of Server Status.

Second element: The client’s status information as a JSON object

Client’s JSON object includes:

◦ **hostName** *[string]*	HostName of the computer the client is running on.
◦ **statusMessage** *[string]*	Current client’s state as a human readable message
◦ **status** *[string]*	Current client’s device state (s. Chapter 5 “Device States”)

◦ **name** *[string] (optional)*	Limited to 256 characters. Could include everything what might be useful to describe the connected client. For example: “Technical monitoring Luebeck, WST1”

Example:

As heartbeat the client sends “clientStatus” to the server.

```
[
    "clientStatus",
    {
        "hostName": "ClientComputerName",
        "name": "Technical monitoring Luebeck, WST1",
        "status": "OK",
        "statusMessage": "OK"
    }
]
```

Timing:
If the server/client connection is setup to use a connection timeout the client must send the message in an interval less than the connection timeout. E.g. if the connection timeout is set to 12 seconds the client should send the “clientStatus” message every five seconds.

3.8 Client Connections

This message is sent to the client as response to the “getClientConnections” command.

First element: **“clientConnections”**

Second element: The client’s status information as a JSON array representing all connected clients. The content is the same as sent by the “clientStatus” output message. Each connected client is presented as a JSON object with the following keys:

◦ **clientConnectionID** *[number]* (optional)	Unique connection ID
◦ **ipAddress** *[string]*	Currently assigned IP Address of the computer the client is running on.
◦ **tcpPort** *[string]* (optional)	This corresponds to the port of the TCP socket on client’s side of the connection. As this information may not be easily retrieved it may be optionally provided.
◦ **state** *[string]*	Current client’s state as a human readable message
◦ **stateInt** *[number]*	Current client’s state number (s. Chapter 5 “Device States”)

◦ **generalState** *[string]*	Current client's general state (s. Chapter 5 "Device States")
◦ **name** *[string]* (optional)	Limited to 256 characters. Could include everything what might be useful to describe the connected client. For example: "Technical monitoring Luebeck, WST1"

Example:

```
[
    "clientConnections", [
        {
            "clientConnectionID": 2,
            "generalState": "OK",
            "ipAddress": "192.168.122.98",
            "name": "Technical monitoring Luebeck, WST1",
            "tcpPort": "61234",
            "state": "Connected",
            "stateInt": 4
        },
        {
            "clientConnectionID": 3,
            "generalState": "OK",
            "ipAddress": "192.168.122.103",
            "name": "Maritime Monitoring System Luebeck",
            "tcpPort": "64321",
            "state": "Connected",
            "stateInt": 4
        }
    ]
]
```

4 Command Messages

Command messages are used to make changes to the DF Service.

Each command contains one task for the DF Service, specified by the command key and JSON object. Therefore, a command describes one action with no ambiguity.

4.1 Response Messages

After a command has been received, DF Service responds immediately only to the requested client with either:

- Error message if command was not formatted correctly e.g., wrong JSON-format.
- Accept message if command is successfully accepted. This means that the DF Service will try to execute this command.

Check status messages to detect the configuration change!

If the command has successfully changed the module configuration (e.g., DF System, DF Channel, Triangulator, etc.), the status information is broadcasted immediately to all connected clients after a change.

If the command was not successfully executed, the status with the unchanged values is transmitted with a specified cycle of appropriate module. (Refer status messages in appropriate module documentation).

Please note that some commands e.g., set a new frequency or squelch level, can also take time to be executed. The timeout time for the not successful change can be defined based on the given network communication.

There is no additional error response to the requested client.

Example error responses:

```
[
    "error",
    {
      "Message" : "JSON data invalid or bad structure"
    }
  ]

  [
    "error",
    {
      "Message" : "JSON data missing event identifier or object."
    }
  ]

  [
    "error",
    {
      "Message" : "Unknown Event Identifier: <event identifier>"
    }
  ]
```

Example accepted responses:

```
[
    "commandAccepted",
    {
       "requestedCommand" : "updateDfSystem"
    }
]
```

4.2 DF System

For DF System changes, three commands are available: create, delete and update.

4.2.1 Create DF System

DF System Create command creates a new, empty DF System.

To create a DF System, send an array with:

First element: **"createDfSystem"**.
Second element:

- Empty object
  or
- JSON object with name

  Optional (used if provided)
  - **name** *[string]* DF System's name

Example 1:
New empty DF System

```
[
    "createDfSystem",
    {}
]
```

Copy and transmit over TCP (CRLF must be added at the end):

```
["createDfSystem",{}]
```

Example 2:
New DF System named TestSystem

```
[
    "createDfSystem",
    {
        "name": "TestSystem"
    }
]
```

Copy and transmit over TCP (CRLF must be added at the end):

```
["createDfSystem",{"name":"TestSystem"}]
```

4.2.2 Delete DF System

DF System Delete command deletes an already existing DF System. To successfully delete a DF System, valid sysId is required.

ATTENTION:
When DF System is deleted, all channels and devices in it are also deleted.

To delete a DF System, send an array with:

First element: **“deleteDfSystem”**
Second element: JSON object with sysId

Required (always present)

- **sysId** *[string]* ID of the DF System to be deleted

Example 1:
Delete existing DF System by sysId.

```
[
    "deleteDfSystem",
    {
        "sysId": "f76708e0-e150-4e16-8f6f-9814feb47cbf"
    }
]
```

Copy and transmit over TCP (CRLF must be added at the end):

```
["deleteDfSystem",{"sysId":"f76708e0-e150-4e16-8f6f-9814feb47cbf"}]
```

4.2.3 Update DF System

DF System Update command modifies existing DF System parameters.

To modify a DF System, send an array with:

First element: **“updateDfSystem”**.
Second element: JSON object with parameters to modify.

<u>Required (always present)</u>

- **sysId** *[string]* — ID of the DF System to be modified.
  Setting to update

<u>Available settings to update *(only one can be changed at a time)*:</u>

- **name** *[string]* — Set DF System’s name.

- **utcSource** *[string]* — Set DF System’s source of the UTC date and time. Options:
  - “Local Machine”
  - “GPS”

- **validBearingMin** *[number]* — Set minimum degree of valid bearing sector.
  - Valid: [0 - 360]
  - Unit: Decimal Degree

- **validBearingMax** *[number]* — Set maximum degree of valid bearing sector.
  - Valid: [0 - 360]
  - Unit: Decimal Degree

- **antenna** *[JSON Object]* **type** *[string]* — Set type of the Antenna Unit (AU). Options:
  - “RT-1000-ATC”
  - “RT-1000-VTS”
  - “RT-500-M”
  - “RT-800”

- **antenna** *[JSON Object]* **correction** *[number]* — Set Mechanical Correction setting of the AU.
  - Valid: [-180 - 180]
  - Unit: Decimal Degree

- **antenna** *[JSON Object]* **upsideDown** *[boolean]* — Set mounting of the AU.
  - True if Antenna Unit is mounted upside-down.
  - False if Antenna Unit is mounted standard.

- **antenna** *[JSON Object]* **orientationMode** *[string]* — Set orientation mode of the AU. Options:
  - “tn” - True North
  - “mn” - Magnetic North
  - “hdt” - True heading from Compass

- “hdm” - Magnetic heading from Compass
- “cog” - Course over ground from GPS

- **antenna** *[JSON Object]*
  variation *[number]* Set variation of the AU.

  variationSource *[string]* Set variation source of the AU.
  Options:
  - “Manual Input” - Use VAR value.
  - “gps” - Use GPS as source for variation, ignore VAR.

- **antenna** *[JSON Object]*
  lat *[number]* Set latitude of the AU.
  - Valid: [-90 - 90]
  - Format: Decimal degrees

  lon *[number]* Set longitude of the AU.
  - Valid: [-180 - 180]
  - Format: Decimal degrees

  positionSource *[string]* Set position source of the AU.
  Options:
  - “Manual Input” - Use LAT and LONG values.
  - “gps” - Use GPS as source for position.

- **antenna** *[JSON Object]*
  alt *[number]* Set altitude of the AU.

  altitudeSource *[string]* Set altitude source of the AU.
  Options:
  - “Manual Input” - Use LAT and LONG value.
  - “gps” - Use GPS as source for altitude, ignore ALT.

- **antenna** *[JSON Object]*
  transmitterHeight *[number]* Set transmitter height, for Radio Horizon calculation.
  - Unit: meters

- **gps** *[JSON Object]*
  active state *[string]* Set active state of the GPS device.
  Options:
  - ON
  - OFF

  ipAddress *[string]* Set IP address of the GPS device.

  tcpPort *[string]* Set TCP port of the GPS device.

Example 1:

Change Antenna Unit’s Orientation mode to Magnetic North.

```
[
    "updateDfSystem",
    {
        "sysId": "51ccfeaf-f0b7-480c-957e-613661bd9034",
        "antenna": {
            "orientationMode": "mn"
        }
    }
]
```

Copy and transmit over TCP (CRLF must be added at the end):

```
["updateDfSystem",{"sysId":"51ccfeaf-f0b7-480c-957e-613661bd9034","antenna":{"orienta-
tionMode":"mn"}}]
```

Example 2:

Change Antenna Unit’s Position.

```
[
    "updateDfSystem",
    {
        "sysId": "51ccfeaf-f0b7-480c-957e-613661bd9034",
        "antenna": {
            "lat": 54.485947,
            "lon": 11.163944
        }
    }
]
```

Copy and transmit over TCP (CRLF must be added at the end):

```
["updateDfSystem",{"sysId":"51ccfeaf-f0b7-480c-957e-613661bd9034","an-
tenna":{"lat":54.485947,"lon":11.163944}}]
```

4.3 DF Channel

For DF Channel changes, three commands are available: create, delete and update.

4.3.1 Create DF Channel

DF Channel Create command creates a new, empty DF Channel in a specified DF System.

To create a DF Channel, send an array with:

First element: **“createDfChannel”**.
Second element: JSON object with sysId.

- **Required (always present)**
  - **sysId** *[string]* DF System’s ID to create the DF channel in.

Example 1:
Create a new, empty DF Channel.

```
[
    "createDfChannel",
    {
        "sysId": "51ccfeaf-f0b7-480c-957e-613661bd9034"
    }
]
```

Copy and transmit over TCP (CRLF must be added at the end):

```
["createDfChannel",{"sysId":"51ccfeaf-f0b7-480c-957e-613661bd9034"}]
```

4.3.2 Delete DF Channel

DF Channel Delete command deletes an already existing DF Channel. To successfully delete a DF Channel, valid chId is required.

To delete a DF Channel, send an array with:

First element: **“deleteDfChannel”**.
Second element: JSON object with sysId and chId.

JSON object with sysId and chId:

Required (always present)

- **sysId** *[string]* DF System’s ID to delete the DF channel in.

- **chId** *[string]* ID of the DF Channel to be deleted.

Example 1:

Delete existing DF Channel by chId.

```
[
    "deleteDfChannel",
    {
        "sysId": "51ccfeaf-f0b7-480c-957e-613661bd9034",
        "chId": "511213aa-bfa4-4fd2-a083-0c41accd1c87"
    }
]
```

Copy and transmit over TCP (CRLF must be added at the end):

```
["deleteDfChannel",{"sysId":"51ccfeaf-f0b7-480c-957e-613661bd9034","chId":"511213aa-
bfa4-4fd2-a083-0c41accd1c87"}]
```

4.3.3 Update DF Channel

DF Channel Update command modifies existing DF Channel parameters.

To modify a DF Channel, send an array with:

First element: **"updateDfChannel"**.
Second element: DF Channel's JSON object.

DF Channel's JSON object:

- **sysId** *[string]* DF System's ID to modify DF Channel in.
- **chId** *[string]* ID of the DF Channel to be modified.
  Setting to update

Available settings to update (only one group can be changed at a time):

- **activeState** *[string]* Set DF Channel's state.
  Options:
  - "ON"
  - "OFF"
- **name** *[string]* Set DF Channel's name.
- **rackNumber** *[number]* Set DF Channel's rack number.
- **protocol** *[string]* Protocol type of the DF Channel.
  Options:
  - "RT-1000" ***(default)***
  - "RT-500-M" - NMEA protocol
  - "RT-800" - NMEA protocol
  - "RT-500-M Antenna Unit"
  - "RT-600 Antenna Unit"
  - "RT-800 Antenna Unit"

- **operatingMode** *[string]* Set operating mode of the DF Channel. Options:
  - For “RT-1000” protocol:
    - “Bearing Mode”
    - “Marine Scan”
  - For “RT-500-M” and “RT-800” protocol:
    - “Bearing Mode”
    - “Marine Scan”
    - “CP-SS Scan”
    - “CP-SS Decode Mode”
  - For “Antenna Unit” protocols:
    - “Bearing Mode”
    - “Marine Scan”
    - “CP-SS Scan”
    - “CP-SS Decode Mode”
- **freq** *[number]* Set DF Channel’s frequency.
- **squelch** *[number]* Set DF Channel’s squelch setting.
  - Valid: [0 - 60]
- **squelchdBm** *[number]* Set DF Channel’s squelch setting in dBm.
- **ipAddress** *[string]* Set DF Channel’s IP address setting.
- **tcpPort** *[string]* Set DF Channel’s TCP port setting.

Example 1:
Change frequency and name of the DF Channel with specified chId.

```
[
    "updateDfChannel",
    {
        "sysId": "50c19d23-897b-4fbd-b66b-782a3d71b3e9",
        "chId": "138a5fe7-09bc-4575-a93c-f1daed691822",
        "freq": 121500000,
        "name": "Emergency"
    }
]
```

Copy and transmit over TCP (CRLF must be added at the end):

```
["updateDfChannel",{"sysId":"50c19d23-897b-4fbd-b66b-782a3d71b3e9","chId":"138a5fe7-09bc-4575-a93c-f1daed691822","freq":121500000,"name":"Emergency"}]
```

Example 2:
Change IP settings of the DF Channel with specified chId.

```
[
    "updateDfChannel",
    {
        "sysId": "50c19d23-897b-4fbd-b66b-782a3d71b3e9",
        "chId": "138a5fe7-09bc-4575-a93c-f1daed691822",
        "ipAddress": "192.168.77.248",
        "tcpPort": "4001"
    }
]
```

Copy and transmit over TCP (CRLF must be added at the end):

```
["updateDfChannel",{"sysId":"50c19d23-897b-4fbd-b66b-782a3d71b3e9","chId":"138a5fe7-
09bc-4575-a93c-f1daed691822","ipAddress":"192.168.77.248","tcpPort":"4001"}]
```

4.4 Triangulator

All parameters of the triangulator can be set with one command at once or also one by one.

Command JSON-Array with:

First element: "**updateTriangulator**".
Second element: JSON object with parameters.

JSON Object:

Required (at least one key for setting is required)

◦ **en** *[boolean]*	Switch ON or OFF the triangulator. ▪ If true, the triangulator is switched ON ▪ If false, the triangulator is switched OFF.
◦ **sectorBlankingActive** *[boolean]*	Switch ON or OFF sector blanking. ▪ If true, the sector blanking is switched ON ▪ If false, the sector blanking is switched OFF.
◦ **radius** *[number]*	Sets the triangulation radius. Radius defines the area around every DF station where the triangulation is possible. (If the distance between two DF stations is higher than this parameter, no triangulation results will be possible). ▪ Valid [0 … 40000000] (40.000 km is the equator length) ▪ Format: Decimal meter ***Default: 1000000 (1000 km)***
◦ **testMode** *[boolean]*	Sets the triangulator in a test mode. Flag which indicates if the triangulator is in test mode. In test mode, the triangulator additionally produces test triangulation results on four different frequencies. ▪ If true, the triangulator is in test mode. ▪ If false, the triangulator is not in test mode.
◦ **frequencies** *[array]*	Sets the list of frequencies which are used for triangulation. Other frequencies will be ignored. ▪ Format: *[number]* Array elements are numbers.
◦ **systems** *[array]*	Sets the list of DF system-Ids which are used for triangulation. Other sys-Ids are ignored. ▪ Format: *[string]* Array elements are strings.

Example 1:

Switch ON Triangulator

```
[
    "updateTriangulator",
    {
        "en": true
    }
]
```

Copy and transmit over TCP (CRLF must be added at the end):

```
["updateTriangulator",{"en":true}]
```

Example 2:

Set triangulator to 30 km radius and switch on the test mode

```
[
    "updateTriangulator",
    {
        "radius": 30000,
        "testMode": true
    }
]
```

Copy and transmit over TCP (CRLF must be added at the end):

```
["updateTriangulator",{"radius":30000,"testMode":true}]
```

Example 3:

Set frequency list

```
[
    "updateTriangulator",
    {
        "frequencies": [
            121500000,
            156800000,
            243000000
        ]
    }
]
```

Copy and transmit over TCP (CRLF must be added at the end):

```
["updateTriangulator",{"frequencies":[121500000,156800000,243000000]}]
```

Example 4:

Combined command:

Switch on triangulator, set the radius to 50 km, switch off test mode, set frequency list, set the system list.

```
[
    "updateTriangulator",
    {
        "en": true,
        "radius": 50000,
        "testMode": false,
        "frequencies": [
            156800000,
            156525000,
            121500000
        ],
        "systems": [
            "1111",
            "2222",
            "3333"
        ]
    }
]
```

Copy and transmit over TCP (CRLF must be added at the end):

```
["updateTriangulator",{"en":true,"radius":50000,"testMode":false,"frequen-
cies":[156800000,156525000,121500000],"systems":["1111","2222","3333"]}]
```

4.5 Get Client Connections

The “getClientConnections” command can be send to servers to query the list of connected clients.

The “getClientConnections” command is a JSON object sent through the active connection.

The active connection may be either a connection to the port the TCP server or the WebSocketServer is listening for incoming connections. This means that before sending the command a connection to one of the valid server ports must have been established.

As response the server will send the message “clientConnections” to the client.

First element: “**getClientConnections**”
Second element: Empty object

Example 1:

```
[
    "getClientConnections",
    {}
]
```

4.6 Register Client

The “registerClient” command is an optional command which may be sent by clients right after the connection to the server has been established to apply additional, useful information of the connected client to the server.

Note in advance concerning Host Names and IP addresses:

In networks using the dynamic host configuration protocol (DHCP) the IP address may change each time the computer is reconnected to the network. The IP address therefore cannot be used to unambiguously identify the client’s computer. The hostname is unique within sub networks but might still be ambiguous when leaving the local network. But as the host name is more human readable that an IP address and can be easily retrieved as the local host name of the computer it was decided that a client should provide this information.

The “registerClient” command is a JSON object representing the active client.

First element: “**registerClient**”
Second element: JSON object with parameters specifying the client.

Client’s JSON object includes:

◦ **hostName** *[string]*	Host name of the computer the client is running on.
◦ **name** *[string] (optional)*	Limited to 256 characters. Could include everything what might be useful to describe the connected client. For example: “Technical monitoring Luebeck, WST1”

Example for “registerClient” message:

```
[
    "registerClient",
    {
        "hostName": "ClientComputerName",
        "name": "Technical monitoring Luebeck, WST1"
    }
]
```

4.7 Heartbeats

To setup heartbeats two commands are available.

4.7.1 Update Server Status Interval

The “updateServerStatusInterval” command may be sent by clients to setup the interval the server is sending the “serverStatus” message to the client.

If a client connects to the JSON server interface of the DF Commander the server will send the “serverStatus” message in default time interval of 5 seconds.
The client may use those messages as a heartbeat from the server to detect connection issues.

The interval in which the server sends the “serverStatus” message can be setup for each connection separately by using the “updateServerStatusInterval” message.

First element: “**updateServerStatusInterval**”
Second element: JSON object with parameters to modify.

Available settings to update:

◦ **interval** *[number]*	Interval in milliseconds. ▪ Range [100 ms .. 300000 ms (= 5 minutes)]

Example:

The client requests the server to send the “serverStatus” heartbeat message
every second.

```
[
    "updateServerStatusInterval",
    {
        "interval": "1000"
    }
]
```

4.7.2 Update Client Status Timeout

The “updateClientStatusTimeout” command may be sent by clients to setup the timeout in which the server expects to receive the “clientStatus” message from the client.

As default the timeout is set to 0 seconds which means that the timeout is disabled.

To modify the timeout, send an array with:

First element: “**updateClientStatusTimeout**”
Second element: JSON object with parameters to modify

Available settings to update:

- **timeout** *[number]*

- A value greater than 0 forces the server to check whether “clientStatus” messages are received within the specified timeout. The client must ensure that those heartbeat messages are sent within the specified timeout.
- With a value less or equal to 0 the server will not check for heartbeat message timeouts. The client may still send heartbeat message but the server will just ignore them.

Example:

The client wants to enable the connection timeout.
If the server does not receive the “clientStatus” messages within 5 seconds the server can assume that the connection is somehow broken.

```
[
    "updateClientStatusTimeout",
    {
        "timeout": "5000"
    }
]
```

5 Device States

All network devices inside or outside DF Systems such as DF channels, GPS devices, compass devices have in general the following error and warning states with the corresponding messages / descriptions.

5.1 General Device State

General State (String)	Changes
OFF	Device is disabled by user or per command. In this case the device generates no error or warning even if it is defective. (E.g., in case of DF channel malfunction, the corresponding DF channel can be set to OFF in order to prohibit the error forwarding to the DF system. Possible monitoring systems would not provide alarm while replacing the units).
ERROR	Device has an error and cannot function as intended.
WARNING	Device has a warning but is functional.
OK	Device is fully functional.

The general device state is directly dependent on device state (s. next chapter) if available.

5.2 Device State (stateInt)

Beside the general state, the detailed device state information is provided - **stateInt**. This information can be used for technical monitoring of direction finder system and also help for quick search of error reasons and efficient repair. The detailed device state is applicable for devices, which have a connection over the network.

State (meaning)	**State Numbers (stateInt)**	**Description**
NOT USED	0	This is place holder for future developments. The device does not have an implemented stateInt indicator.
Off	1	Device is disabled by user. The General Device State is then OFF. All errors or warnings and data are ignored.
Disconnected	2	Device is disconnected. The General Device State is then ERROR. Possible reasons: - Device is manually disconnected by user but not set to OFF. - Server has actively disconnected the device.
Connecting	3	Device is currently connecting. The General Device State is then ERROR. Possible reasons: - The device server is unavailable. - LAN cable is unplugged. (In case of cable interruption, the device state will switch between states "Disconnected" and "Connecting")
Connected	4	Device is connected. The General Device State depends on the next states.
DataTimeOut	5	No data available on device. The network connection is established, but no data is coming. The General Device State is then ERROR. Possible reasons: - The device is either defective or does not produce data. - In case of using serial devices with the combination of serial to LAN converters, an appropriate serial cable can be unplugged from the device or the

		device has no power, but the serial to LAN converter is working. - LAN cable is unplugged.
BadData	6	Incompatible or defective data arriving to device. The General Device State is then ERROR. Possible reasons: - The device is configured with a wrong server. E.g., the DF Channel is connected to the socket of the GPS device server. - The wrong device type is configured.
DeviceError	7	Device has an error making it inoperable. The General Device State is then ERROR. This state indicates that network connection is established, and correct device data is received, but the device itself has an error, which is transmitted over proprietary device protocol. The error reason depends on the device. The detailed information can be looked up in parameter “state” [string] or in the manual of the device.
DeviceWarning	8	Device has a warning, but operable. The General Device State is then WARNING. This state indicates that network connection is established, and correct device data is received, but the device itself produces a warning, which is transmitted over proprietary device protocol. The warning reason depends on the device. The detailed information can be looked up in parameter “state” [string] or in the manual of the device.
Ok	9	Device is fully functional. The General Device State is then OK.