Table of Contents

Introduction

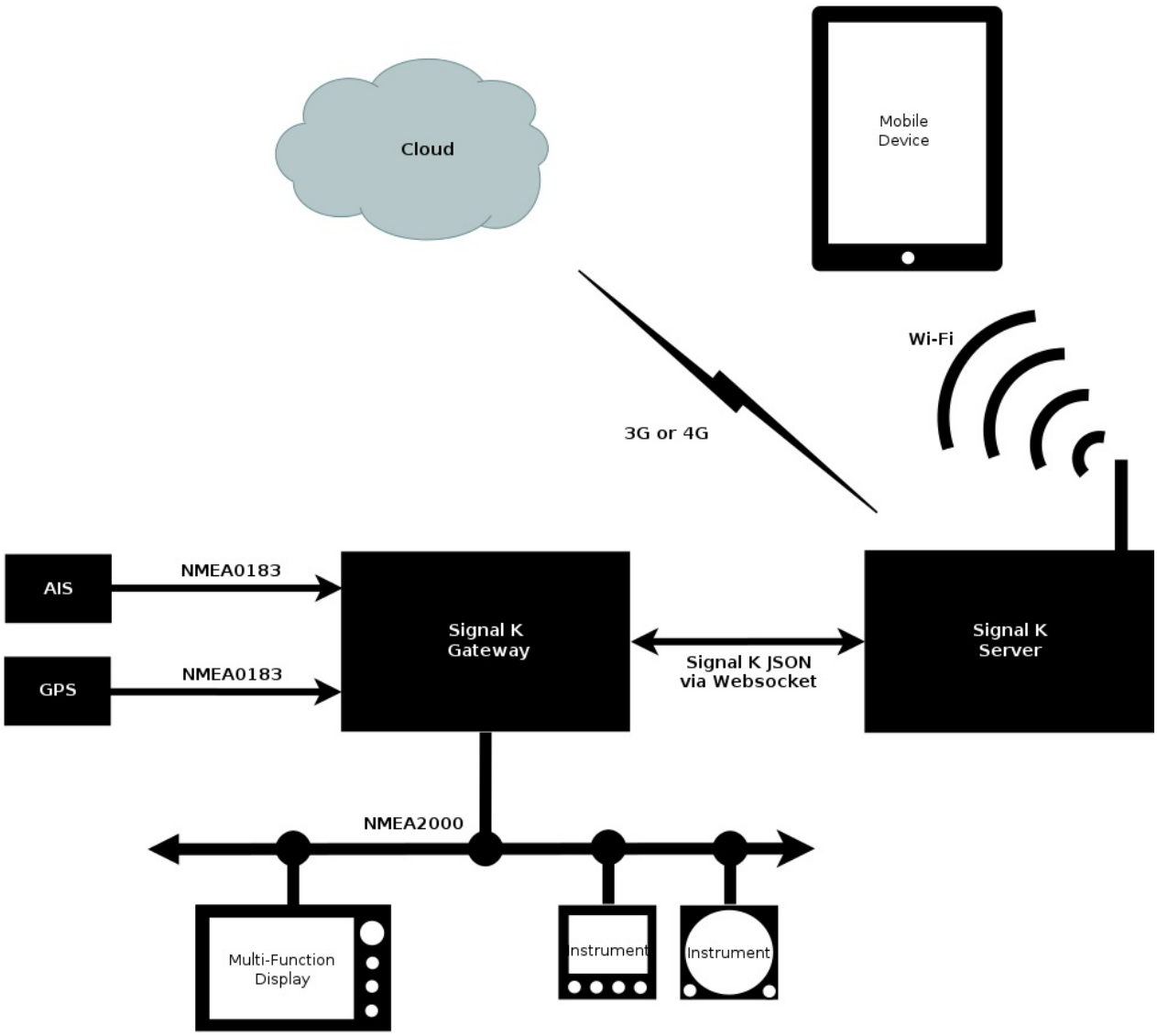
This is the documentation for the Signal K Specification 0.0.1-2 version, which is available in the following formats;

- html (this document) 0.0.1-2
- pdf
- epub
- mobi

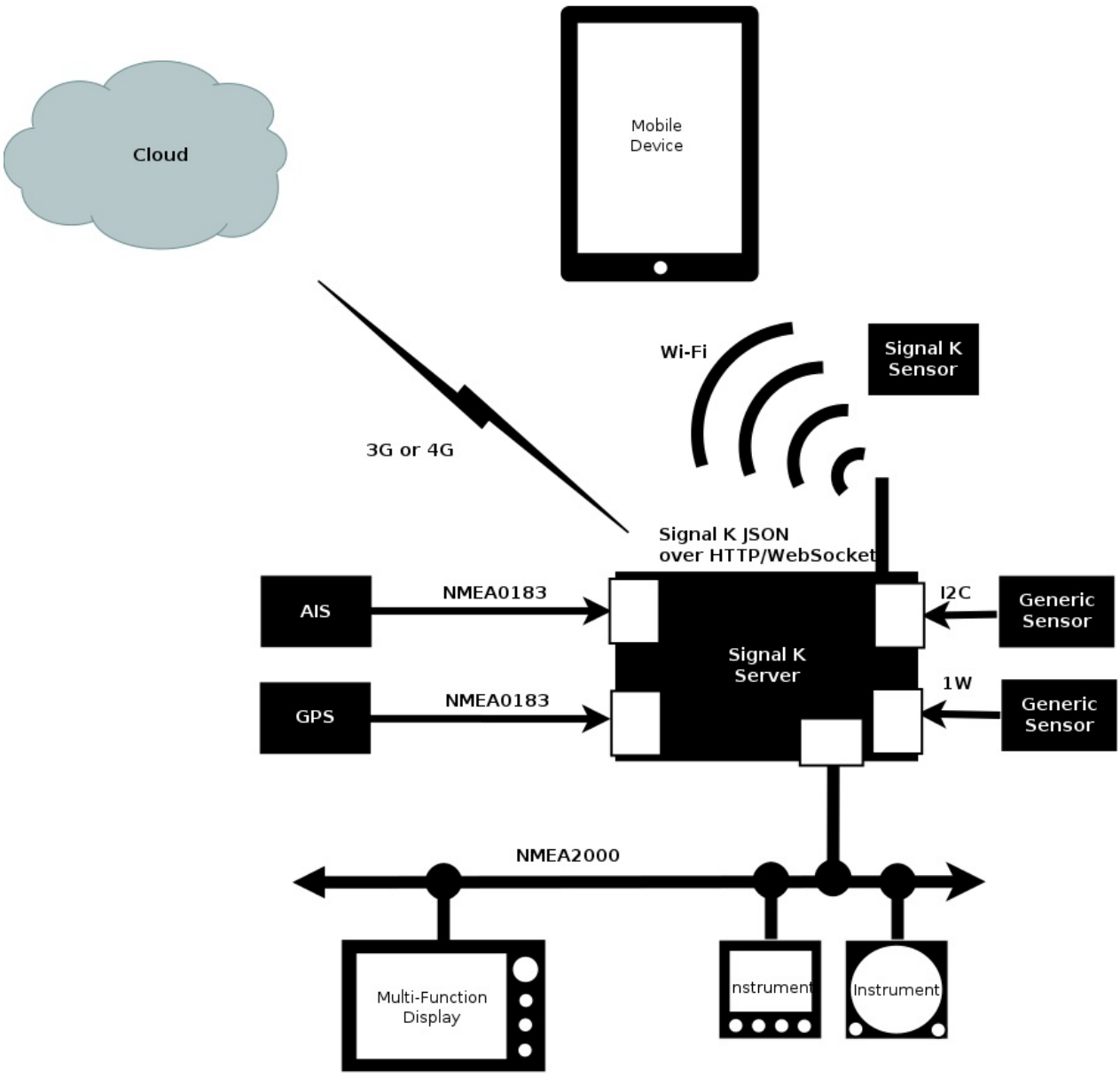
What is Signal K?

Signal K is a modern and open data format for marine use. It is an Internet friendly standard built on standard web technologies, such as JSON and WebSockets. Signal K is Free and Open Source software. This document is licensed under the Creative Commons CC-BY-SA license. All Signal K source code is licensed under the Apache License, Version 2.0. Signal K is developed in the open with help from the marine community. Your ideas and feedback are valuable and welcome.

Signal K is designed to work in harmony with the boat's existing navigation equipment that might use NMEA0183, NMEA2000 or proprietary data protocols, converting and enhancing this information to a modern "web friendly" format that can be shared, processed and displayed on the latest web apps, mobile devices and cloud servers. A typical NMEA based setup consists of an NMEA-Signal K gateway and an optional server. The gateway translates the NMEA data to Signal K and the server can host additional functions like logging, cloud integration and data analysis.



One major advantage of Signal K is the ability to represent data from heterogeneous sources. In addition to traditional NMEA sources data from generic sensors as well as modern Signal K enabled sensors can be fused into a single data model and a single protocol for accessing the data. Another typical setup is a Signal K server with adapters and converters for the different sources.



Signal K Data Model (A.K.A. The Schema)

The Signal K Data Model or schema defines a universal model for marine related information and it is specified as a JSON schema. See the Signal K Data Model section for details.

In traditional marine standards there are many tightly defined messages, each with a specific purpose, but there is no data model to relate them. Furthermore, any device which needs to decode those messages must have a copy of the data dictionary in order to do so. By defining a data model in JSON we can make the messaging layer simpler, and easily extensible. We define consistent units and meta data for each data point in the model. This means that a specific data point (e.g. COG) will always be found at a predictable address.

It also means that a display device such as a chartplotter implementing Signal K does not need to know about the data model beforehand. It can query the central Signal K server on the boat to get all the information it needs to display any data point. This **metadata** may include information such as the unit of measure, minimum and maximum permissible values, alarm thresholds and localized display name for every data point in the model.

Signal K Message Format

Signal K defines methods for combining arbitrary data from the Data Model into valid messages. These messages are in UTF-8 JSON format.

Rather than define hundreds of specific messages, Signal K has a few common message formats which can contain any combination of data from the Data Model. This means that we don't need new messages for every new case, just extra data in the payload. It means that any device can read any message and a device can introduce a new data point which can be understood by existing devices without the need for firmware upgrades.

Signal K Transport Layer

Signal K does not define the transport or wire protocol. Signal K messages are JSON text and can be sent over almost any transport layer. However, we do provide guidance on how to make an initial connection, handle negotiation, subscription, and disconnection for a given transport (e.g. TCP/IP or serial).

Where possible we use well established standards like HTTPS, REST, WebSockets, MQTT, STOMP, etc. But within each method there are always many options in message addressing, formating, or transfer (GET, POST), etc.

The goal is to try to establish sensible conventions for each protocol, to make development and interconnection more predictable.

Signal K Implementations

The Signal K project has Open Source reference server implementations in Node and Java and example web apps and reference code. There are also commercial Signal K applications and solutions, including mobile apps available on the Apple App and

Android Play Stores, as well as hardware products like iKommunicate.

Getting Started Using Signal K

You can start using Signal K by

- connecting to the demo server on the Internet with any web browser
- installing either the Node or Java server on any computer
- getting some hardware for your boat, such as a Raspberry Pi, suitable USB adapters for your boat's network (NMEA 0183, NMEA 2000 or roll your own with I2C sensors) and installing Node or Java server
- purchasing a commercial Signal K gateway such as an iKommunicate by Digital Yacht
- installing OpenPlotter, which includes a Signal K server

Once you have a server running (or you start by using the demo server) you can install some Signal K supporting mobile apps such as

- NMEARemote by Zapfware (iOS)
- OceanIX (Android)

Getting Started in Developing with Signal K

- Example HTML5 Applications
- Signal K JavaScript Client
- iKommunicate Developer's Guide

Signal K Data Model

Signal K defines two data formats, full and delta, for representing and transmitting data.

In additiong the 'sparse' format is the same as the full format, but doesn't contain a full tree, just parts of the full tree.

Full format

The simplest format is the full format, which is the complete Signal K data model as a JSON string.

```
{
  "vessels": {
    "urn:mrn:signalk:uuid:c0d79334-4e25-4245-8892-54e8ccc8021d": {
      "version": "0.1",
      "name": "motu",
      "mmsi": "2345678",
      "source": "self",
      "timezone": "NZDT",
      "navigation": {
        "state": {
          "value": "sailing",
          "source": "self",
          "timestamp": "2014-03-24T00:15:41Z"
        },
        "headingTrue": {
          "value": 2.3114,
          "$source": "nmea0183-1.II",
          "sentence": "HDT",
          "timestamp": "2014-03-24T00:15:41Z"
        },
        "speedThroughWater": {
          "value": 2.556,
          "$source": "n2k-1.160",
          "pgn": 128259,
          "timestamp": "2014-03-24T00:15:41Z"
        },
        "position": {
          "longitude": 23.53885,
          "latitude": 60.0844,
          "$source": "nmea0183-2.GP",
          "timestamp": "2014-03-24T00:15:42Z",
          "sentence": "GLL"
        }
      }
    }
  }
}
```


The message is UTF-8 ASCII text, and the top-level attribute(key) is always "vessels". Below this level is a list of vessels, identified by their MMSI number or a generated unique id. There may be many vessels, if data has been received from AIS or other sources. The format for each vessel's data uses the same standard Signal K structure, but may not have the same content, i.e. you won't have as much data about other vessels as you have about your own.

The values are always SI units, and always the same units for the same key. I.e. `speedOverGround` is always meters per second, never knots, km/hr, or miles/hr. This means you never have to send 'units' with data, the units are specific for a key, and defined in the data schema. A simplified version of the JSON schema with the units is available in Keys Reference in Appendix A.

The ordering of keys is also not important, they can occur in any order. In this area Signal K follows normal JSON standards.

The full format is useful for backups, and for populating a secondary device, or just updating all data, a kind of 'this is the current state' message.

However sending the full data model will be wasteful of bandwidth and CPU, when the majority of data does not change often. So we want to be able to send parts of the model (i.e. parts of the hierarchical tree).

Sparse format

The sparse format is the same as the full format but only contains a limited part of the tree. This can be one or more data values.

```
{
  "vessels": {
    "self": {
      "navigation": {
        "position": {
          "latitude": -41.2936935424,
          "longitude": 173.2470855712
        }
      }
    }
  }
}
```

Mix and match of misc values are also valid:

```
{
  "vessels": {
    "self": {
      "navigation": {
        "courseOverGroundTrue": {
          "value": 11.9600000381
        },
        "position": {
          "latitude": -41.2936935424,
          "longitude": 173.2470855712,
          "altitude": 0
          }
        }
      }
    }
  }
}
```

This mix of any combination of values means we don't need to create multiple message types to send different combinations of data. Just assemble whatever you want and send it. When parsing an incoming message a device should skip values it has no interest in, or doesn't recognise. Hence we avoid the problem of multiple message definitions for the same or similar data, and we avoid having to decode multiple messages with fixed formats.

Delta format

While building the reference servers and clients it was apparent that a third type of message format was useful. This format specifically sends changes to the full data model. This was useful for a number of technical reasons, especially in clients or sensors that did not hold a copy of the data model.

The format looks like this (pretty printed):

```
{
    "context": "vessels.urn:mrn:imo:mmsi:234567890",
    "updates": [{
        "source": {
            "type": "NMEA2000",
            "src": "017",
            "pgn": 127488,
            "label": "N2000-01.017"
        },
        "timestamp": "2010-01-07T07:18:44Z",
        "values": [{
            "path": "propulsion.0.revolutions",
            "value": 16.341667
        }, {
            "path": "propulsion.0.boostPressure",
            "value": 45500.0
        }, {
            "path": "propulsion.0.tiltTrim",
            "value": 48
        }]
    }]
}
```

In more detail we have the header section:

```
{
  "context": "vessels.urn:mrn:imo:mmsi:234567890",
  "updates": [
    ...data goes here...
  ]
}
```

A delta message can be recognised from the other types by the topmost level having `updates` property. `updates` is the only required property. If `context` is missing it is assumed that the data is related to the `self` context.

Context is a path from the root of the full tree to the *container object*, which for vessel related data must refer to the vessel directly under `vessels`. The delimiter in the context path is `.` (period). In this case the context is `vessels.urn:mrn:imo:mmsi:234567890`. All subsequent data is relative to that location.

The `updates` holds an array (JSON array) of updates, each of which has a `source` and a JSON array of `values`.

```
{
  "source": {
    "device": "/dev/actisense",
    "src": "115",
    "pgn": "128267"
  },
  "timestamp": "2014-08-15-16:00:00.081",
  "values": [
    {
      "path": "navigation.courseOverGroundTrue",
      "value": 2.971
    },
    {
      "path": "navigation.speedOverGround",
      "value": 3.85
    }
  ]
}
```

An `update` has a single `source` value and it applies to each of the `values` items. In cases where you can get data from only a single source the source may be omitted and the receiver may fill it in when multiplexing data from several sources.

A Signal K producer may not have access to a real time clock or UTC time. In these cases timestamp should be omitted. Elements in the Signal K processing chain, like a server receiving data from a producer, should fill in timestamp if it is missing in the incoming delta message.

Each `value` item is then simply a pair of `path` and `value`. The `path` must be a *leaf path*: it must be a path to a leaf the of the full model. A leaf is where the actual value of the Signal K property is and where `timestamp`, `$source` and `values` properties are in the full model. The value is often a scalar - a numeric value, as in the example above, but it can also be an object. For example a `navigation.position` value would be an object like `{"latitude": -41.2936935424, "longitude": 173.2470855712}`.

Message Integrity

Many messaging systems specify checksums or other forms of message integrity checking. In Signal K we assume a reliable transport will guarantee a valid message. This is true of TCP/IP and some other transports but not always the case. For other transports (eg RS232 serial) a specific extended data format will apply, which is suited to that transport. Hence at the message level no checksum or other tests need to be made.

Encoding/Decoding

The JSON message format is supported across most programming environments and can be handled with any convenient library.

On micro-controllers with limited RAM it is wise to read and write using streaming rather than hold the whole message in precious RAM. There is an implementation of Signal K JSON streaming on an Arduino Mega (4K RAM) in the related Freeboard project.

Multiple Values for a Key

There are two use cases for multiple data:

- Multiple instances of a common device - eg two engines or multiple batteries
- Multiple devices providing duplicate data - multiple values for the same Signal K key from different sensors, eg COG from both compass and gps or multiple depth sounders

Multiple instances of a common device

Some parts of the Signal K schema are **device oriented**.

For example in electrical domain you have the concept of multiple batteries. Each battery has multiple, common quantities like voltage, current and temperature. Here we have chosen to organise the Signal K data model hierarchy by instance: multiple batteries are represented as for example `electrical.batteries.starter` and `electrical.batteries.house`. Then underneath that prefix you have the different properties and quantities.

This organisation allows a user interface to organise the individual readings in meaningul groups and you can query all the values related to that piece of equipment via REST API. It maintains the primary requirement that a given data value have a fixed and unique uri, but gives flexibility in the structure and complexities of data.

The same device centric organisation is used in `propulsion` subschema, where the most common use case is having dual engine setup with `propulsion.port` and `propulsion.starboard`.

The values `starter`, `house`, `port` and `starboard` are examples and not specified in the schema. You are free to use application specific values within the regexp specified in the JSON schema.

Multiple devices providing duplicate data

It is quite possible for a key value to come from more than one device. eg position (lat/lon) could come from several gps enabled devices, and multiple depth sounders are not uncommon. We need a consistent way to handle this.

All the incoming values may well be valid in their own context, and it is feasible that all of them may be wanted, for instance, displaying depth under each hull on a catamaran.

Hence discarding or averaging is not a solution, and since signalk is unable to derive the best way to handle multiple values it must always fall to a default action, with human over-ride when needed.

The solution presented below has flaws. See https://github.com/SignalK/specification/issues/48 for discussion.

In signal K we can leverage the above method and simply store all the devices in the tree under the main item, and have the main items `source` reference the options. Lets consider this for `courseOverGroundTrue`

If its the first value for the key, it becomes the default value and looks like this:

```
{
  "vessels": {
    "self": {
      "navigation": {
        "courseOverGroundTrue": {
          "value": 102.29,
          "source": "vessels.self.sources.n2k.actisense-115-129026"
        }
      },
      "sources": {
        "n2k": {
          "actisense-115-129026": {
            "value": 102.29,
            "bus": "/dev/actisense",
            "timestamp": "2014-08-15-16: 00: 01.083",
            "src": "115",
            "pgn": "129026"
          }
        }
      }
    }
  }
}
```

It has come from device `vessels.self.sources.n2k.actisense-115-129026`, where further details can be found.

If another value with different source arrives, we add the source with a unique name, so both values are in there - if its our preferred source (from persistent config) we auto-switch to it, otherwise we just record it. It look like this:

```
{
  "vessels": {
    "self": {
      "navigation": {
        "courseOverGroundTrue": {
          "timestamp": "2014-08-15-16: 00: 01.083",
          "value": 102.29,
          "source": "vessels.self.sources.n2k.actisense-115-129026"
        }
      },
      "sources": {
        "n2k": {
          "actisense-115-129026": {
            "value": 102.29,
            "bus": "/dev/actisense",
            "timestamp": "2014-08-15-16: 00: 01.083",
            "src": "115",
            "pgn": "129026"
          },
          "actisense-201-130577": {
            "value": 102.29,
            "bus": "/dev/actisense",
            "timestamp": "2014-08-15-16: 00: 00.085",
            "src": "201",
            "pgn": "130577"
          }
        }
      }
    }
  }
}
```

Rules

Now simple rules can apply to obtain the default, or any specific value:

- The implementation must ensure that the `key.value` holds an appropriate value. This will be easy if there is only one, and will probably be user configured if more.
- If the `source` value is `string` then it is a reference key to the source object, and can be a relative or absolute signalk key.
- The `source` (as a reference string) also provides a mechanism to handle deprecated keys.

- If the `source` value is a `json object` then it holds meta data on the source of the value.
- Alternate sources must be discovered manually, or by implementation specific meta-data.

To see all the entries, use the REST api or subscribe to the parent object. A given device may choose to subscribe to a specific entry in the object, allowing multiple displays of the key, or users of the various values. The 'list' verb used in a query message can provide available keys.

Unique names

The identifier for each device should be unique within the server, and possibly be constructed as follows:

```
n2k: producerid-sourceid-pgn (producer id from server configuration, others from n
2k data) - NOTE: will change, currently under discussion.
nmea0183: producerid-talkerid-sentence (like n2k)
signalk: any valid string matching regex [a-zA-Z0-9-]. eg alphabet, hyphens, and 0
 to 9
```

(The nmea0183 talker id is not in the schema as I write this, it will be added shortly)

Metadata

The Use Cases

Let's assume we have engine1.rpm as a key/value in Signal K. We want to display it on our dashboard, and monitor alarms for temp, oil, rpm etc.

We can drop a generic dial gauge on our dash and display rpm, but it can't know maxRpm, or alarms unless its an engine-specific gauge, and knows where to look in the Signal K schema. So we will end up with a profusion of role specific gauges to maintain. We also have non standard key names for max, min, high, low, etc. which pollute the schema.

Currently the Signal K server has a set of specific alarm keys. These grow over time and are becoming awkward. The server can only monitor these specific keys at present as there is no mechanism for arbitrary alarm definition.

Metadata for a Data Value

Each data key should have an optional `.meta` object. This holds data in a standard way which enables the max/min/alarm and display to be automatically derived.

```
{
  "displayName": "Tachometer, Engine 1",
  "shortName": "RPM",
  "warnMethod": "visual",
  "warnMessage": "any text",
  "alarmMethod": "sound",
  "alarmMessage": "any text",
  "zones": [
    {"lower":0.0,"upper":500,"state":"alarm", "message":"Stopped or very slow Rpm"
},
    {"lower":500,"upper":3000,"state":"normal", "message":""},
    {"lower":3000,"upper":3500,"state":"warn", "message":"Approaching maximum rpm"
},
    {"lower":3500,"upper":9999,"state":"alarm", "message":"Exceeding maximum rpm"}
  ]
}
```

Since the settings object is always the same, the tachometer can now limit its range, and display green, yellow, and red sectors. The generic gauge can now perform this role, with correct labels etc.

The alarms problem is also improved, as the server can run a background process to monitor any key that has a `.meta` object, and raise a generic alarm event. By recursing the tree the alarm monitoring can find the source (engine1), giving the alarm context. See [[Alarm Handling]]

The alarms functionality then becomes generic, and grows with the spec. This is may be the case for other functionality also.

Meta.units value

All keys in the specification must have `units`. If a client requests the `meta.units` for a valid key eg `GET /signalk/v1/api/vessels/123456789/navigation/speedThroughWater/meta/units`, the REST interface MUST return proper value.

See https://github.com/SignalK/specification/blob/0.0.1-2/keyswithmetadata.json

Default Configuration

Other than a few standard keys it is unlikely that the `.meta` can have global defaults, as it is very vessel specific (e.g. a sail boat will have speeds from 0-15kts, a ski boat will have 0-50kts). So the values will have to be configured by the user on the individual vessel as required.

It is probably possible to have profiles that set a range of default `.meta`, e.g. sail vessel, or motor vessel, and if two vessels have the same engine, then the engine profiles will also tend to be the same.

Alarm Management

An alarm watch is set by setting the `meta.zones` array appropriately. A background process on the server checks for alarm conditions on any attribute with a `meta.zones` array. If the keys value is within a zone the server sets an alarm key similar to `vessels.self.notifications.[original_key_suffix]`, eg an alarm set on `vessels.self.navigation.courseOverGroundTrue` will become `vessels.self.notifications.navigation.courseOverGroundTrue`.

The object found at this key should contain the following:

```
{
  "message": "any text",
  "state": "[normal|alert|warn|alarm|emergency]"
}
```

Other Benefits

The common profiles should be exportable and importable. This would allow manufacturers or other users to create profiles for specific products or use cases, which could then be imported to a vessel.

This may also have possibilities for race control or charter management. For instance a limit on lat/lon would raise an 'Out of Bounds' email on a charter vessel.

A lot of the current max/min/alarm values could be removed to simplify and standardise the spec.

Permissions Model

The permissions model for Signal K is based on the UNIX file permissions model. This was first developed in the late 1970's and is still perfectly suited to the internet today, so its got to be a pretty sound model!.

So we adapted it for Signal K. See http://www.tutorialspoint.com/unix/unix-file-permission.htm

Each key in Signal K has an optional `_attr` value.

```
"vessels": {
    "self":{
             //the usual signal k keys, navigation, environment, etc

       "_attr":{                // filesystem specific data, eg security, possibl
y more later
                "_mode": 640,         // unix style permissions, often written in
`owner:group:other` form, `-rw-r-----`
                "_owner" : "self",    // owner, surprisingly. The user who created
 the item, sometimes a virtual user like 'self'
                "_group": "self"      // group
             }
           }
        }
```

By default the `vessels.self` key has the above `_attr`. This effectively means that only the current vessels 'owner' can read and write from this key or any of its sub-keys. It also allows users in group `self` to read the data. This provides a way to give additional programs or users read-only access. In the above case an external user connecting from outside the vessel and requesting vessel data would receive `{}`, eg nothing.

Note:keys beginning with `_` are always stripped from signal k messages

Since the above is a default, Signal K devices that lack the resources to implement security should always be installed behind a suitable gateway that can provide security. Again, the simplest security is the default read-write only within the local vessel (typically the current network). This makes a basic implementation as simple as possible.

The permissions apply recursively to all sub-keys, unless specifically overwritten. You can only provide a **narrowing** change in permissions, eg less than the parent directory. In the above case if the permissions for `vessels.self.navigation.position` were set to

`"_mode" : 644`, it would have no effect as access is blocked at the `vessels.self` key. The `vessels.self` _attr must now also be `"_mode" : 644`, and all its other subkeys explicitly set to `"_mode" : 640`

Hence setting complex permissions are likely beyond the typical user. For this reason we believe there should be a choice of default permission 'templates' for the signal K tree. Users would select their preference from a config screen. A paranoid user may prefer the above setup, another may chose to allow basic data similar to AIS (position, cog, speed, etc), and others may expose much more.

Templates also allow sharing of data for specific uses or needs, like a social group, or a marina.

Exposing everything (`"_mode" : 666`) would be dangerous - it would potentially allow external users to gain control of the vessels systems, however it is useful for demos and software development. All signal K implementations should always consider the potential danger of such permissions, and protect users if possible.

The implementation of proper security is the responsibility of the Signal K software implementation provider.

By manipulating the `_attr` values for the Signal K keys, and creating suitable users and groups a sophisticated and well proven security model for vessel data can be created.

Ports, Urls and Versioning

Short Names

- `self` refers to the current vessel. Normally used in `vessels.self...`.

Ports

The Signal K HTTP and WebSocket services SHOULD be found on the usual HTTP/S ports (80 or 443). The services SHOULD be found on the same port, but may be configured for independent ports and MAY be configured for ports other than HTTP/S.

A Signal K server MAY offer Signal K over TCP or UDP, these services SHOULD be on port 55555[1].

If an alternate port is needed it SHOULD be an arbitrary high port in the range 49152–65535[2].

URL Prefix

The Signal K applications start from the `/signalk` root. This provides some protection against name collisions with other applications on the same server. Therefore the Signal K entry point will always be found by loading `http(s)://«host»:«port»/signalk`.

Versioning

The version(s) of the Signal K API that a server supports SHALL be available as a JSON object available at `/signalk`:

```
{
    "endpoints": {
        "v1": {
            "version": "1.1.2",
            "signalk-http": "http://192.168.1.2/signalk/v1/api/",
            "signalk-ws": "ws://192.168.1.2:34567/signalk/v1/stream"
        },
        "v3": {
            "version": "3.0",
            "signalk-http": "signalk/v3/api/",
            "signalk-ws": "ws://192.168.1.2/signalk/v3/stream",
            "signalk-tcp": "tcp://192.168.1.2:34568"
        }

    }
}
```

This response is defined by the `discovery.json` schema. In this example, the server supports two versions of the specification: `1.1.2` and `3.0`. For each version, the server indicates which transport protocols it supports and the URL that can be used to access that protocol's endpoint; in the example, the `1.1.2` REST endpoint is located at `http://192.168.1.2/signalk/v1/api/`. Clients should use one of these published endpoints based on the protocol version they wish to use.

The server must only return valid URLs and should use IANA standard protocol names such as `http`. However, a server may support unofficial protocols and may return additional protocol names; for example, the response above indicates the server supports a `signalk-tcp` stream over TCP at on port `34568`.

A server may return relative URIs that the client must resolve against the base of the original request.

REST/HTTP API: /signalk/v1/api/

Note the trailing slash in the path.

The base URL MUST provide a Signal K document that is valid according to the full Signal K schema specification. The contents SHOULD be all the current values of the data items the server knows.

If the path following the base is a valid Signal K path `GET` MUST retrieve the Signal K branch named by the path; e.g.
`/signalk/v1/api/vessels/123456789/navigation/speedThroughWater` returns


```
{
    "value": 2.55,
    "source": {
        "type": "NMEA0183",
        "src": "VHW",
        "label": "signalk-parser-nmea0183"
    },
    "timestamp": "2015-08-31T05:45:36.000Z"
}
```

Streaming WebSocket API: /signalk/v1/stream

Initiates a WebSocket connection that will start streaming the server's updates as Signal K delta messages. You can specify the contents of the stream by using a specific URL:

- ws://hostname/signalk/v1/stream?subscribe=self
- ws://hostname/signalk/v1/stream?subscribe=all
- ws://hostname/signalk/v1/stream?subscribe=none

With no query parameter the default is `self`, which will stream the data related to the `self` object. `all` will stream all the updates the server sees and `none` will stream only the heartbeat, until the client issues subscribe messages in the WebSocket stream.

If a server does not support some streaming options listed in here it must respond with http status code `501 Not Implemented`.

See Subscription Protocol for more details.

Connection Hello

Upon connection a 'hello' message is sent as follows:

```
{
  "version": "1.1.2",
  "timestamp": "2015-04-13T01:13:50.524Z",
  "self": "123456789"
}
```

Discovery and Connection Establishment

Service Discovery

A Signal K server SHOULD advertise its services using DNS Service Discovery (DNS-SD) via Mutlicast DNS (mDNS); also known as Bonjour. The server MUST provide DNS Service (SRV) Records and Text (TXT) Records describing the Signal K interfaces it provides. These service identifiers are:

- `_http._tcp` for the server's web interface
- `_signalk-http._tcp` for the Signal K REST API
- `_signalk-ws._tcp` for the WebSocket data stream

If a server is providing Signal K via secure versions of HTTP or WebSockets then they MUST be able to provide a redirection to the secure versions of these protocols.

If a Signal K server is using DNS-SD, it MUST provide the following parameters (key/value pairs) in the TXT record portion of the DNS-SD advertisement:

- `txtvers` is a US-ASCII decimal number identifying the version of the DNS-SD record. Currently, this MUST have a value of 1
- `roles` specifies which roles the server is capable of providing. See Roles below for details

The server MAY provide the following values:

- `self` is the unique identifier of the vessel using the URN format specified for the `uuid` field in the Signal K schema. It may also use the URN format specified for the `mmsi` field in the Signal K schema if it exists.
- `swname` is the name of the Signal K server software, e.g. signalk-server-node
- `swvers` is the version of the Signal K server software

An example DNS-SD record set is shown below.

```
Service data for service 'signalk-http' of type '_signalk-http._tcp' in domain 'lo
cal' on 4.0:
    Host 10-1-1-40.local (10.1.1.40),
    port 80,
    TXT data: [
        'txtvers=1',
        'roles=master,main',
        'self=urn:mrn:signalk:uuid:c0d79334-4e25-4245-8892-54e8ccc8021d',
        'swname=signalk-server',
        'swvers=0.1.23'
        ]

Service data for service 'signalk-ws' of type '_signalk-ws._tcp' in domain 'local'
 on 4.0:
    Host 10-1-1-40.local (10.1.1.40),
    port 3000,
    TXT data: [
        'txtvers=1',
        'roles=master,main',
        'self=urn:mrn:signalk:uuid:c0d79334-4e25-4245-8892-54e8ccc8021d',
        'swname=signalk-server',
        'swvers=0.1.23'
        ]
```

These records are advertising a Signal K server with the HTTP REST API on port 80 and the WebSocket data stream on port 3000. The server identifies as having the `master` and `main` roles and provides a `self` identifier as a UUID.

Roles

The four possible values for `roles` are `master`, `slave`, `main`, and `aux`. These are defined below.

Master

`master` is the canonical source for identity and configuration information for the entire vessel.

If there is only one master on the vessel, then it should also provide the main role. The combination of master and main informs a client that this server is actively providing identifying information.

Main and Aux

If there are more than one masters on the vessel, EXACTLY ONE server should advertise both master and main. All other masters should advertise master and aux. Clients should only use the master aux servers for identifying information if the master main is not available.

Any server identifying as master MUST be able to provide at a minimum the unique identifier (self) for the vessel.

Slave

Any server providing the `slave` role should retrieve identity and configuration information from the master server. Slave servers MAY provide configuration and identity information for themselves, but this identity MUST NOT be considered valid for the entire vesssel.

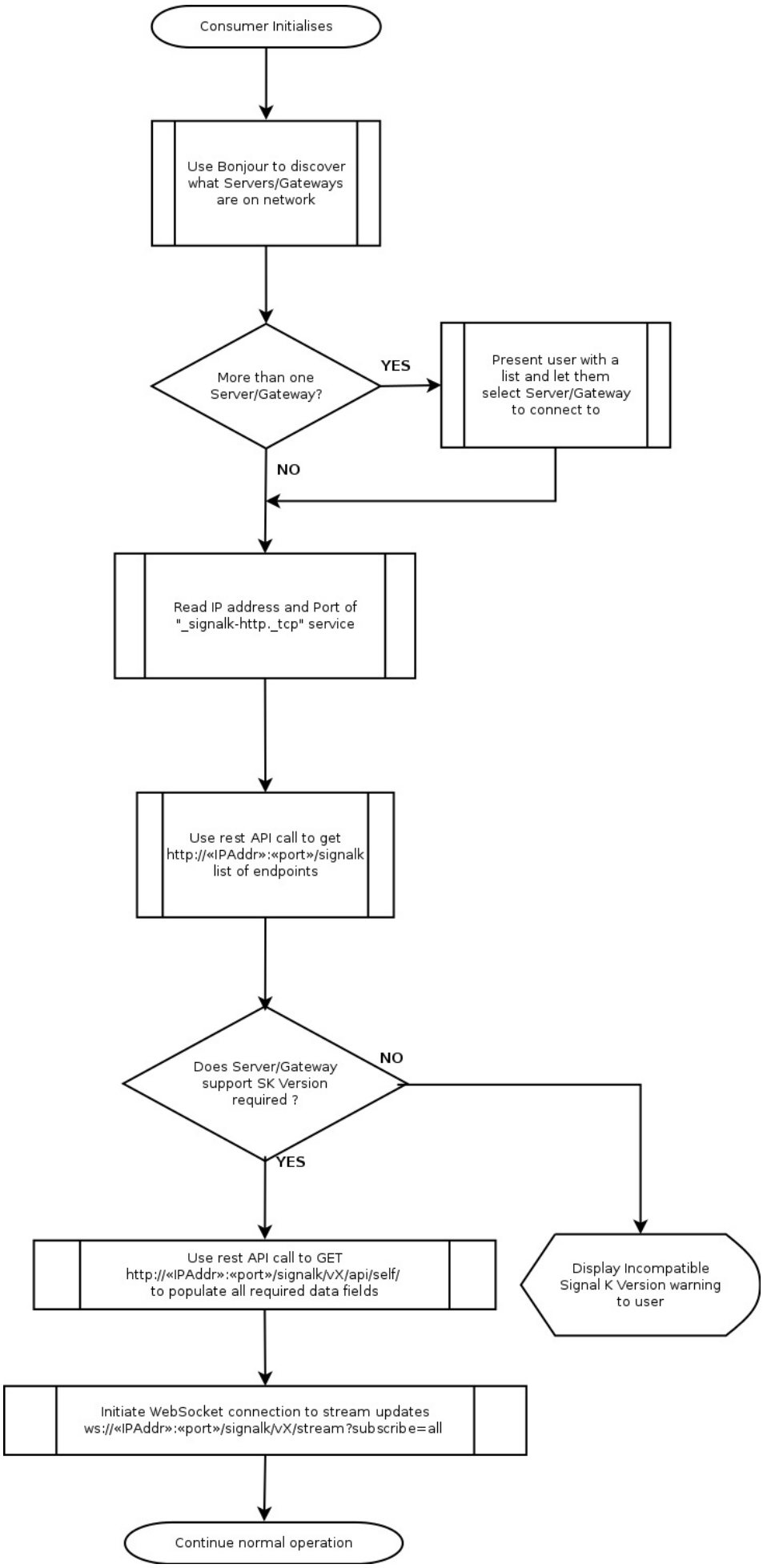
Main and Aux

The use of main and aux have not been defined for the slave role at this time.

Connection Establishment

Using the information above a web client or HTTP capable device can discover and connect to a Signal K server using the following process:

- Query for Signal K services using mDNS
- Connect to the host and port advertised as 'signalk-http' via HTTP (e.g. `http://10.1.1.40:80`)
- Per the Ports, Urls and Versioning section, make a GET request for `/signalk` to retrieve a JSON object containing an `endpoints` JSON object
- Make further REST calls for more specific data, or open a websocket connection to start streaming updates.


Consumer Initialises
Use Bonjour to discover what Servers/Gateways are on network
More than one Server/Gateway?
YES
Present user with a list and let them select Server/Gateway to connect to
NO
Read IP address and Port of "_signalk-http._tcp" service
Use rest API call to get http://«IPAddr»:«port»/signalk list of endpoints
Does Server/Gateway support SK Version required ?
NO
YES
Use rest API call to GET http://«IPAddr»:«port»/signalk/vX/api/self/ to populate all required data fields
Display Incompatible Signal K Version warning to user
Initiate WebSocket connection to stream updates ws://«IPAddr»:«port»/signalk/vX/stream?subscribe=all
Continue normal operation

Subscription Protocol

Introduction

By default a Signal K server will provide a new WebSocket client with a delta stream of the `vessels.self` record, as updates are received from sources. E.g. `/signalk/v1/stream` will provide the following delta stream, every time the log value changes .

```
{
  "context": "vessels",
   "updates": [{
      "source": {
        "pgn": "128275",
        "device": "/dev/actisense",
        "timestamp": "2014-08-15-16:00:05.538",
        "src": "115"
      },
      "values": [
        {
          "path": "navigation.logTrip",
          "value": 43374
        },
        {
          "path": "navigation.log",
          "value": 17404540
        }]
     }
     ]
}
```

> Below we refer to WebSockets, but the same process works in the same way over any transport. E.g. for a raw TCP connection the connection causes the above message to be sent, and sending the subscribe messages will have the same effect as described here.

This can be a lot of messages, many you may not need, especially if `vessel.self` has many sensors, or other data sources. Generally you will want to subscribe to a much smaller range of data.

First you will want to unsubscribe from the current default (or you may have already connected with `ws://hostname/signalk/v1/stream?subscribe=none`). To unsubscribe all create an `unsubscribe` message with wildcards and send the message over the websocket connection:

```
{
  "context": "*",
  "unsubscribe": [
    {
      "path": "*"
    }
  ]
}
```

To subscribe to the required criteria send a suitable subscribe message:

```
{
  "context": "vessels.self",
  "subscribe": [
    {
      "path": "navigation.speedThroughWater",
      "period": 1000,
      "format": "delta",
      "policy": "ideal",
      "minPeriod": 200
    },
    {
      "path": "navigation.logTrip",
      "period": 10000
    }
  ]
}
```

- `path=[path.to.key]` is appended to the context to specify subsets of the context. The path value can use the wildcard `*`. A wildcard in the middle of a path (`propulsion/*/oilTemperature`) allows any value for that part and a wildcard at the end (`propulsion/port/*`) matches all paths beginning with the specified prefix.

The following are optional, included above only for example as it uses defaults anyway:

- `period=[millisecs]` becomes the transmission rate, e.g. every `period/1000` seconds. Default=1000
- `format=[delta|full]` specifies delta or full format. Default: delta
- `policy=[instant|ideal|fixed]`. Default: ideal

    - `instant` means send all changes as fast as they are received, but no faster than `minPeriod`. With this policy the client has an immediate copy of the current state of the server.
    - `ideal` means use `instant` policy, but if no changes are received before `period`, then resend the last known values.eg send changes asap, but send the value every `period` millisecs anyway, whether changed or not.
    - `fixed` means simply send the last known values every `period`.
- `minPeriod=[millisecs]` becomes the fastest message transmission rate allowed, e.g. every `minPeriod/1000` seconds. This is only relevant for policy='instant' to avoid swamping the client or network.

You can subscribe to multiple data keys multiple times, from multiple apps or devices. Each app or device simply subscribes to the data it requires, and the server and/or client implementation may combine subscriptions to avoid duplication as it prefers on a per connection basis. At the same time it is good practice to open the minimum connections necessary, for instance one websocket connection shared bewteen an instrument panel with many gauges, rather then one websocket connection per gauge.

Multiple value handling in subscriptions

A subscription to a key is for the *simple* value of the key, eg for a subscription to `navigation.speedThroughWater` we expect to get `navigation.speedThroughWater.value`

If we want the `values` we need to recover the `values` object by a REST or `json get` message (see below), and subscribe to `navigation.speedThroughWater.values.n2kFromFile.43.value`

Single use, or intermittent data

When data is required once only, or upon request the `subscribe/unsubscribe` method should not be used. If the client is http capable the REST api is a good choice, or use `get/list/put` messages over websockets or tcp.

GET/PUT/LIST variants

The `get/list/put` messages work in the same way as their `GET/PUT` REST equivalents, returning a json result for the requested path, once only. They exist to allow REST like functionality for devices without HTTP capability.

```
{
  "context": "vessels.self",
  "get": [
    {
      "path": "environment.depth.belowTransducer"
    }
  ]
}
```

```
{
  "context": "vessels",
   "put": [{
      "source": {
        "pgn": "128275",
        "device": "/dev/actisense",
        "timestamp": "2014-08-15-16:00:05.538",
        "src": "115"
      },
      "values": [
        {
          "path": "navigation.logTrip",
          "value": 43374
        }]
     }
     ]
}
```

Use Cases and Proposed Solutions

Local boat individual instruments

A gauge-type display for just one or a few data items for the 'self' vessel should be able to specify that it only wants those items for the self vessel.

This can be achieved by a default WebSocket connection `/signalk/v1/stream?subcribe=none`, then sending a JSON message:

```
{
  "context": "vessels.self",
  "subscribe": [
    {
      "path": "environment.depth.belowTransducer"
    },
    {
      "path": "navigation.speedThroughWater"
    }
  ]
}
```

The JSON format is also viable over a simple TCP or serial transport, and is therefore supported as the primary subscription method.

Map display with all known vessel positions & directions, served over 3G cellular connection

```
{
  "context": "vessels.*",
  "subscribe": [
    {
      "path": "navigation.position",
      "period": 120000,
      "policy": "fixed"
    },
    {
      "path": "navigation.courseOverGround",
      "period": 120000,
      "policy": "fixed"
    }
  ]
}
```

The result is a delta message of the Signal K data with just position and courseOverGround branches for all known vessels, sent every 2 minutes (120 seconds) even if no data has been updated.

Position of a certain vessel, immediately it changes, but once per minute at most

```
{
  "context": "vessels.230029970",
  "subscribe": [
    {
      "path": "navigation.position",
      "minPeriod": 60000,
      "policy": "instant"
    }
  ]
}
```

The result will be delta position messages for vessel 230029970, broadcast whenever it changes, but with minimum interval of 60 seconds. Messages are delayed to meet the minimum interval with newer messages overriding the previous message in the buffer.

Alarm, Alert, and Notification Handling

Handling alarms, alerts, and notifications in Signal K is a multi-stage process. Alarms, alerts and notifications are all handled the same way, and are all referred to as alarms below.

We need a flexible model to define alarm conditions, and a standard way to announce and record them.

Alarm Process

- Define alarm states as zones in the meta object attached to any Signal K value. See [[Metadata for Data Values]]
- If the value is within an alarm zone raise the defined alarm.
- If the value goes out of the zone, remove the alarm by setting its value to null
- Alarms are raised by placing an alarm object in the `vessels.self.notifications` tree

Expected implementation behaviour

- The server (or device) should monitor the current value and compare it to the defined zones.
- If a value enters an alarm zone, then a key is written to `vessels.self.notifications..`
- If a value leaves an alarm zone, then the key is removed from `vessels.self.notifications..`
- Alarms raised are monitored by an alarm process on the server, which takes appropriate action, sounding alarms, or displaying messages.
- Clients interested in alarms can subcribe to the `vessels.self.notifications...` tree in the usual way and be informed of alarms in the same way as normal signalk keys.
- When an alarms is removed, a delta should be sent to subscribers with the path and a null value.

Example

eg If we exceed our anchor alarm radius: `vessels.self.navigation.anchor.currentRadius` enters `vessels.self.navigation.anchor.currentRadius.meta.zones[[50,500, "alarm", "Dragging anchor!"]],`

The alarm is : `vessels.self.notifications.navigation.anchor.currentRadius`

The alarm object is

```
{
  "method": ["sound"],
  "state": "alarm",
  "message": "Dragging anchor!",
  "timestamp": ..,
  "source": {..}
}
```

The server alarm manager will see this new entry and turn on the alarm. Using a manager process allows flexibility in situations where multiple alarms are triggered and your vessel is a mass of flashing and beeping. eg A single 'Pause' button can give you 5-10 minutes to take action, stopping annoying noise, and removing popup messages from screens.

Since the `vessels.self.notifications` tree mirrors the other data in the signal k model, we can selectively watch or react to specific branches or keys. When displaying multiple alarms a screen can also sort and filter them.

Other Alarms

Above we have discussed monitoring existing values and raising alarms. There are other alarms that must be considered, eg MOB, fire, sinking etc, and misc alerts "GPS signal lost".etc.

The `vessels.[uuid].notifications` tree is the same as any other Signal k branch. Keys can be added and removed as required in the usual way. Since the branch is being monitored we only need to add a key of any sort to create a suitable alarm.

In the case of an emergency, create a unique key: The alarm is : `vessels.[uuid].notifications.[alarm.key]`

The alarm object is

```
{
  "method": ["visual", "sound"],
  "state": "emergency",
  "message": "Man Overboard!",
  ...
}
```

Alarm objects that have been raised this way must be cleared manually, or by the process that created them. You can use any suitable path, keeping in mind the context of the alarm.

eg In the case of an alert, create a unique key by generating a path: The alarm is : `vessels.[uuid].notifications.navigation.gnss`

The alarm object is

```
{
  "method": ["visual"],
  "state": "alert",
  "message": "GPS signal lost!",
  ...
}
```

Well Known Names

Some alarms are especially important, eg MOB. This is a list of keys for special alarms.

- `..notifications.mob.*`
- `..notifications.fire.*`
- `..notifications.sinking.*`
- `..notifications.flooding.*`
- `..notifications.collision.*`
- `..notifications.grounding.*`
- `..notifications.listing.*`
- `..notifications.adrift.*`
- `..notifications.piracy.*`
- `..notifications.abandon.*`

An example to send an MOB alarm from an N2K source, the gateway would convert and send something like:

```
{
  "context": "vessels.urn:mrn:signalk:uuid:c0d79334-4e25-4245-8892-54e8ccc8021d",
  "updates": [
    {
      "source": {
         ...
        "timestamp": "2014-08-15-16:00:05.538",
      },
      "values": [
        {
          "path": "notifications.mob",
          "value": {
            "message": "MOB",
            "state": "emergency",
            "method": ["visual", "sound"]
          }
        }
      ]
    }
  }
}
```

The resulting full signalk tree would be:

```
{
  "vessels": {
    "urn:mrn:signalk:uuid:c0d79334-4e25-4245-8892-54e8ccc8021d": {
      "notifications": {
        "mob":{
          "message": "MOB",
          "timestamp": "2014-04-10T08:33:53Z",
          "source": {
             ...
          },
          "state": "emergency",
          "method": ["visual", "sound"]
        }
      }
    }
  }
}
```

To clear the alarm condition, send:

```
{
  "context": "vessels.urn:mrn:signalk:uuid:c0d79334-4e25-4245-8892-54e8ccc8021d",
  "updates": [
    {
      "source": {
        ...
        "timestamp": "2014-08-15-16:00:05.538",
      },
      "values": [
        {
          "path": "notifications.mob",
          "value": null
        }
      ]
    }
  ]
}
```

Multiple cases of the same alarm

Should multiple cases of the same alarm occur (eg a gps loses signal, then a second gps loses signal) the alarms are handled the same as any other multiple values in signalk. However alarms will tend to be re-issued whenever the underlying data changes.

The servers alarm monitoring processes are expected to be smart enough to know that the anchor alarm is triggered, and its not necessary to raise a second copy of the same alarm, after all there is only one boat dragging!

This may be handled differently for notifications. It may be useful to know that your gps's are all failing intermittently, or that . Hence the handling of multiple copies of alarms is an implementation issue, and may vary.

The key should be unique

If we have an alarm `vessels.self.notifications.navigation.anchor.currentRadius` and we attempt to write another higher in the same tree at `vessels.self.notifications` it must not replace or remove the existing alarm. Since the `meta.zones` structure is only valid on signalk leaf values this occurs naturally in most circumstances. But it is possible to set an alarm value arbitrarily (eg MOB) and care should be taken in implementations that keys do not overwrite existing paths.

Background and Design Rationale

Arrays Are Evil

In Signal K every datapoint should have a predictable and unique uri (address). What we want to maintain is to know that vessels.self.data.temp is always at that uri, and what the json form is. So if its an array thats workable. If its a json object with many instance keys, each which has a arbitrary name and the same internal structure that works too.

In fact the two forms represent the same data but have different uris and thats the crux. Essentially the first is data.item[collection], where data.item[1] is instance 1, eg the second (0 based) item in the array.

This is no different from data.item as the json object, and data.item.1 as the instance, with the name '1'.

From a code perspective its similar too, the object just has an array of keys. But with objects data.temp.instanceName.value is reliable and always the same.

Does that apply for data.temp[1].value? eg how do you reliably get data.temp.air.value with arrays?

In signalk or java or js, if I have two values in the array, and add one, then remove the first, suddenly the subscriptions are all wrong. The temp[1] did point to the second (0 based) object in the array, but after removing the first its now temp[0]. Subscriptions to temp[1] are now broken.

For an object temp[air] always gets the temp.air. Adding or removing other keys does not affect 'air'.

The array problem can be overcome by programming - but basically thats just fixing a problem that can be easily avoided by not using arrays.

This is a quick start for any-one that would like to contribute. Its roughly from technically unskilled to skilled, top to bottom. Dont be afraid to ask for help. Each task will probably start with a new thread for more details on the Google groups (https://groups.google.com/forum/#!forum/signalk). Be patient, civil, and persistent :-)

Completely unskilled at boat electronics:

- Join https://groups.google.com/forum/#!forum/signalk - as the user base grows, so does awareness.
- Tell others, spread the word
- Fly a Signal K flag from your boat
- If you have special skills (eg motors, batteries, navigation, etc) help us extend the Signal K protocol by identifying what we need to cover.
- Ask manufacturers about Signal K support
- Ask questions about what you dont understand, and collate the answers for us to put on the website.

Can do own installs, handyman, but not IT skilled.

- Try an install of Raspberry Pi and WIFI, document exactly how you did it, so others can follow.

Website or documentation skills

- Help us maintain the website, and improve the documents

Good computer skills, but not programming

- Download and try the java server (https://github.com/SignalK/signalk-server-java) and node server (https://github.com/SignalK/signalk-server-node) and the various apps and clients. Help test and identify issues, help improve documents so others can follow easier.
- Help with User manuals!

Systems engineer

- Help other users, help with scripts, develop and maintain install processes, managing our web sites, etc.
- Examples:
  - Create Debian packages of the Signal K software for easy installation to Raspbian

Software developer

- Download and test/fix our stuff, add improvements, join the team and help code, develop support in your own software.

Microprocessors

- Improve our Arduino stuff, add your own, incorporate Signal K into your products.

Signal K Data Model Reference

This document is meant as the human-oriented reference to accompany the actual JSON Schema specification and is produced from the schema files. Any changes to the reference material below should be made to the original schema files.

Signal K uses SI units almost everywhere, with the exception of geographic coordinates. The following units are in use:

- A : Ampere
- C : Coulomb
- Hz : Hertz
- ISO-8601 (UTC) : ISO-8601 string representation of time in Universal Time Coordinated
- J : Joule
- K : Kelvin
- Pa : Pascal
- V : Volt
- W : Watt
- deg : Degree
- kg : Kilogram
- m : Meter
- m/s : Meters per second
- m2 : Square meter
- m3 : Cubic meter
- m3/s : Cubic meter per second
- rad : Radian
- rad/s : Radian per second
- ratio : Ratio
- s : Second

Keys

/self

Description: This holds the key (UUID, MMSI or URL) of this vessel, the actual data is in the vessels array.

/vessels

Description: A wrapper object for vessel objects, each describing vessels in range, including this vessel.

/vessels/<RegExp>

Title: vessel

Description: This regex pattern is used for validation of an MMSI or Signal K UUID identifier for the vessel. Examples: urn:mrn:imo:mmsi:230099999 urn:mrn:signalk:uuid:c0d79334-4e25-4245-8892-54e8ccc8021d

/vessels/<RegExp>/url

Description: URL based identity of the vessel, if available.

/vessels/<RegExp>/mmsi

Description: MMSI number of the vessel, if available.

/vessels/<RegExp>/uuid

Description: A unique Signal K flavoured maritime resource identifier, assigned by the server.

/vessels/<RegExp>/name

Description: The common name of the vessel

/vessels/<RegExp>/flag

Description: The country of ship registration, or flag state of the vessel

/vessels/<RegExp>/port

Description: The home port of the vessel

/vessels/<RegExp>/registrations

Description: The various registrations of the vessel.

/vessels/<RegExp>/registrations/imo

Description: The IMO number of the vessel.

/vessels/<RegExp>/registrations/national

Description: The national registration number of the vessel.

/vessels/<RegExp>/registrations/national/<RegExp>

Description: This regex pattern is used for validating the identifier for the registration

/vessels/<RegExp>/registrations/national/<RegExp>/country

Description: The ISO 3166-2 country code.

/vessels/<RegExp>/registrations/national/<RegExp>/registration

Description: The registration code

/vessels/<RegExp>/registrations/national/<RegExp>/description

Description: The registration description

/vessels/<RegExp>/registrations/local

Description: A local or state registration number of the vessel.

/vessels/<RegExp>/registrations/local/<RegExp>

Description: This regex pattern is used for validating the identifier for the registration

/vessels/<RegExp>/registrations/local/<RegExp>/registration

Description: The registration code

/vessels/<RegExp>/registrations/local/<RegExp>/description

Description: The registration description

/vessels/<RegExp>/registrations/other

Description: Other registration or permits for the vessel.

/vessels/<RegExp>/registrations/other/<RegExp>

Description: This regex pattern is used for validating the identifier for the registration

/vessels/<RegExp>/registrations/other/<RegExp>/registration

Description: The registration code

/vessels/<RegExp>/registrations/other/<RegExp>/description

Description: The registration description

/vessels/<RegExp>/communication

Title: communication

Description: Communication data including Radio, Telephone, E-Mail, etc.

/vessels/<RegExp>/communication/callsignVhf

Description: Callsign for VHF communication

/vessels/<RegExp>/communication/callsignHf

Description: Callsign for HF communication

/vessels/<RegExp>/communication/phoneNumber

Description: Phone number of skipper

/vessels/<RegExp>/communication/emailHf

Description: Email address to be used for HF email (Winmail, Airmail, Sailmail)

/vessels/<RegExp>/communication/email

Description: Regular email for the skipper

/vessels/<RegExp>/communication/satPhoneNumber

Description: Satellite phone number for vessel.

/vessels/<RegExp>/communication/skipperName

Description: Full name of the skipper of the vessel.

/vessels/<RegExp>/communication/crewNames

Description: Array with the names of the crew

/vessels/<RegExp>/environment

Title: environment

Description: Environmental data measured locally including Depth, Wind, Temp, etc.

/vessels/<RegExp>/environment/outside

Description: Environmental conditions outside of the vessel's hull

/vessels/<RegExp>/environment/outside/temperature

Units: K (Kelvin)

Description: Current outside air temperature

/vessels/<RegExp>/environment/outside/dewPointTemperature

Units: K (Kelvin)

Description: Current outside dew point temperature

/vessels/<RegExp>/environment/outside/apparentWindChillTemperature

Units: K (Kelvin)

Description: Current outside apparent wind chill temperature

/vessels/<RegExp>/environment/outside/theoreticalWindChillTemperature

Units: K (Kelvin)

Description: Current outside theoretical wind chill temperature

/vessels/<RegExp>/environment/outside/heatIndexTemperature

Units: K (Kelvin)

Description: Current outside heat index temperature

/vessels/<RegExp>/environment/outside/pressure

Units: Pa (Pascal)

Description: Current outside air ambient pressure

/vessels/<RegExp>/environment/outside/humidity

Units: ratio (Ratio)

Description: Current outside air relative humidity

/vessels/<RegExp>/environment/outside/illuminance

Units: Lux (undefined)

Description: Current outside ambient light flux.

/vessels/<RegExp>/environment/inside

Description: Environmental conditions inside the vessel's hull

/vessels/<RegExp>/environment/inside/temperature

Units: K (Kelvin)

Description: Current inside air temperature

/vessels/<RegExp>/environment/inside/humidity

Units: ratio (Ratio)

Description: Current inside air relative humidity

/vessels/<RegExp>/environment/inside/engineRoom

Description: Current engine room air temperature

/vessels/<RegExp>/environment/inside/engineRoom/temperature

Units: K (Kelvin)

Description: Temperature

/vessels/<RegExp>/environment/inside/mainCabin

Description: Current main cabin air temperature

/vessels/<RegExp>/environment/inside/mainCabin/temperature

Units: K (Kelvin)

Description: Temperature

/vessels/<RegExp>/environment/inside/refrigerator

Description: Current refrigerator temperature

/vessels/<RegExp>/environment/inside/refrigerator/temperature

Units: K (Kelvin)

Description: Temperature

/vessels/<RegExp>/environment/inside/freezer

Description: Current freezer temperature

/vessels/<RegExp>/environment/inside/freezer/temperature

Units: K (Kelvin)

Description: Temperature

/vessels/<RegExp>/environment/inside/heating

Description: Current heating temperature

/vessels/<RegExp>/environment/inside/heating/temperature

Units: K (Kelvin)

Description: Temperature

/vessels/<RegExp>/environment/water

Description: Environmental conditions of the water that the vessel is sailing in

/vessels/<RegExp>/environment/water/temperature

Units: K (Kelvin)

Description: Current water temperature

/vessels/<RegExp>/environment/water/salinity

Units: ratio (Ratio)

Description: Water salinity

/vessels/<RegExp>/environment/water/liveWell

Description: Current livewell temperature

/vessels/<RegExp>/environment/water/liveWell/temperature

Units: K (Kelvin)

Description: Temperature

/vessels/<RegExp>/environment/water/baitWell

Description: Current baitwell air temperature

/vessels/<RegExp>/environment/water/baitWell/temperature

Units: K (Kelvin)

Description: Temperature

/vessels/<RegExp>/environment/depth

Title: depth

Description: Depth related data

/vessels/<RegExp>/environment/depth/belowKeel

Units: m (Meter)

Description: Depth below keel

/vessels/<RegExp>/environment/depth/belowTransducer

Units: m (Meter)

Description: Depth below Transducer

/vessels/<RegExp>/environment/depth/belowSurface

Units: m (Meter)

Description: Depth from surface

/vessels/<RegExp>/environment/depth/transducerToKeel

Units: m (Meter)

Description: Depth from the transducer to the bottom of the keel

/vessels/<RegExp>/environment/depth/surfaceToTransducer

Units: m (Meter)

Description: Depth transducer is below the water surface

/vessels/<RegExp>/environment/current

Title: current

Description: Direction and strength of current affecting the vessel

Fields:

- drift (The speed component of the water current vector), units: m/s (Meters per

second)
- setTrue (The direction component of the water current vector referenced to true (geographic) north), units: rad (Radian)
- setMagnetic (The direction component of the water current vector referenced to magnetic north), units: rad (Radian)

/vessels/<RegExp>/environment/tide

Title: tide

Description: Tide data

/vessels/<RegExp>/environment/tide/heightHigh

Units: m (Meter)

Description: Next high tide height relative to lowest astronomical tide (LAT/Chart Datum)

/vessels/<RegExp>/environment/tide/heightNow

Units: m (Meter)

Description: The current tide height relative to lowest astronomical tide (LAT/Chart Datum)

/vessels/<RegExp>/environment/tide/heightLow

Units: m (Meter)

Description: The next low tide height relative to lowest astronomical tide (LAT/Chart Datum)

/vessels/<RegExp>/environment/tide/timeLow

Units: RFC 3339 (UTC) (undefined)

Description: Time of the next low tide in UTC

/vessels/<RegExp>/environment/tide/timeHigh

Units: RFC 3339 (UTC) (undefined)

Description: Time of next high tide in UTC

/vessels/<RegExp>/environment/heave

Units: m (Meter)

Description: Vertical movement of the vessel due to waves

/vessels/<RegExp>/environment/wind

Title: wind

Description: Wind data.

/vessels/<RegExp>/environment/wind/angleApparent

Units: rad (Radian)

Description: Apparent wind angle, negative to port

/vessels/<RegExp>/environment/wind/angleTrueGround

Units: rad (Radian)

Description: True wind angle based on speed over ground, negative to port

/vessels/<RegExp>/environment/wind/angleTrueWater

Units: rad (Radian)

Description: True wind angle based on speed through water, negative to port

---

/vessels/<RegExp>/environment/wind/directionChangeAlarm

Units: rad (Radian)

Description: The angle the wind needs to shift to raise an alarm

---

/vessels/<RegExp>/environment/wind/directionTrue

Units: rad (Radian)

Description: The wind direction relative to true north

/vessels/<RegExp>/environment/wind/directionMagnetic

Units: rad (Radian)

Description: The wind direction relative to magnetic north

/vessels/<RegExp>/environment/wind/speedTrue

Units: m/s (Meters per second)

Description: Wind speed over water (as calculated from speedApparent and vessel's speed through water)

/vessels/<RegExp>/environment/wind/speedOverGround

Units: m/s (Meters per second)

Description: Wind speed over ground (as calculated from speedApparent and vessel's speed over ground)

/vessels/<RegExp>/environment/wind/speedApparent

Units: m/s (Meters per second)

Description: Apparent wind speed

/vessels/<RegExp>/environment/time

Description: A time reference for the vessel. All clocks on the vessel dispaying local time should use the timezone offset here. If a timezoneRegion is supplied the timezone must also be supplied. If timezoneRegion is supplied that should be displayed by UIs in preference to simply timezone. ie 12:05 (Europe/London) should be displayed in preference to 12:05 (UTC+01:00)

Fields:

- millis (Milliseconds since the UNIX epoch (1970-01-01 00:00:00))
- timezoneOffset (Onboard timezone offset from UTC in hours and minutes (-)hhmm. +ve means east of Greenwich. For use by UIs)
- timezoneRegion (Onboard timezone offset as listed in the IANA timezone database (tz database))

/vessels/<RegExp>/environment/mode

Description: Mode of the vessel based on the current conditions. Can be combined with navigation.state to control vessel signals eg switch to night mode for instrumentation and lights, or make sound signals for fog.

/vessels/<RegExp>/navigation

Title: navigation

Description: Navigation data including Position, Course to next WP information, etc.

/vessels/<RegExp>/navigation/lights

Title: Navigation lights

Description: Current state of the vessels navigation lights

/vessels/<RegExp>/navigation/courseOverGroundMagnetic

Units: rad (Radian)

Description: Course over ground (magnetic)

/vessels/<RegExp>/navigation/courseOverGroundTrue

Units: rad (Radian)

Description: Course over ground (true)

/vessels/<RegExp>/navigation/courseRhumbline

Title: Course

Description: Course information computed with Rhumbline

/vessels/<RegExp>/navigation/courseRhumbline/crossTrackError

Units: m (Meter)

Description: The distance from the vessel's present position to the closest point on a line (track) between previousPoint and nextPoint. A negative number indicates that the vessel is currently to the left of this line (and thus must steer right to compensate), a positive number means the vessel is to the right of the line (steer left to compensate).

/vessels/<RegExp>/navigation/courseRhumbline/bearingTrackTrue

Units: rad (Radian)

Description: The bearing of a line between previousPoint and nextPoint, relative to true north.

/vessels/<RegExp>/navigation/courseRhumbline/bearingTrackMagnetic

Units: rad (Radian)

Description: The bearing of a line between previousPoint and nextPoint, relative to magnetic north.

/vessels/<RegExp>/navigation/courseRhumbline/activeRoute

Description: Data required if sailing to an active route, defined in resources.

/vessels/<RegExp>/navigation/courseRhumbline/activeRoute/href

Description: A reference (URL) to the presently active route, in resources.

/vessels/<RegExp>/navigation/courseRhumbline/activeRoute/estimatedTimeOfArrival

Description: The estimated time of arrival at the end of the current route

---

/vessels/<RegExp>/navigation/courseRhumbline/activeRoute/startTime

Description: The time this route was activated

/vessels/<RegExp>/navigation/courseRhumbline/nextPoint

Description: The point on earth the vessel's presently navigating towards

Fields:

- position (The position of nextPoint in two dimensions)
  - latitude
  - longitude

/vessels/<RegExp>/navigation/courseRhumbline/nextPoint/distance

Units: m (Meter)

Description: The distance in meters between the vessel's present position and the nextPoint

---

/vessels/<RegExp>/navigation/courseRhumbline/nextPoint/bearingTrue

Units: rad (Radian)

Description: The bearing of a line between the vessel's current position and nextPoint, relative to true north

---

/vessels/<RegExp>/navigation/courseRhumbline/nextPoint/bearingMagnetic

Units: rad (Radian)

Description: The bearing of a line between the vessel's current position and nextPoint, relative to magnetic north

/vessels/<RegExp>/navigation/courseRhumbline/nextPoint/velocityMadeGood

Units: m/s (Meters per second)

Description: The velocity component of the vessel towards the nextPoint

/vessels/<RegExp>/navigation/courseRhumbline/nextPoint/timeToGo

Units: s (Second)

Description: Time in seconds to reach nextPoint's perpendicular) with current speed & direction

/vessels/<RegExp>/navigation/courseRhumbline/previousPoint

Description: The point on earth the vessel's presently navigating from

Fields:

- position (The position of lastPoint in two dimensions)
  - latitude
  - longitude

/vessels/<RegExp>/navigation/courseRhumbline/previousPoint/distance

Units: m (Meter)

Description: The distance in meters between previousPoint and the vessel's present position

/vessels/<RegExp>/navigation/courseGreatCircle

Title: Course

Description: Course information computed with Great Circle

/vessels/<RegExp>/navigation/courseGreatCircle/crossTrackError

Units: m (Meter)

Description: The distance from the vessel's present position to the closest point on a line (track) between previousPoint and nextPoint. A negative number indicates that the vessel is currently to the left of this line (and thus must steer right to compensate), a positive number means the vessel is to the right of the line (steer left to compensate).

/vessels/<RegExp>/navigation/courseGreatCircle/bearingTrackTrue

Units: rad (Radian)

Description: The bearing of a line between previousPoint and nextPoint, relative to true north.

/vessels/<RegExp>/navigation/courseGreatCircle/bearingTrackMagnetic

Units: rad (Radian)

Description: The bearing of a line between previousPoint and nextPoint, relative to magnetic north.

/vessels/<RegExp>/navigation/courseGreatCircle/activeRoute

Description: Data required if sailing to an active route, defined in resources.

/vessels/<RegExp>/navigation/courseGreatCircle/activeRoute/href

Description: A reference (URL) to the presently active route, in resources.

/vessels/<RegExp>/navigation/courseGreatCircle/activeRoute/estimatedTimeOfArrival

Description: The estimated time of arrival at the end of the current route

---

/vessels/<RegExp>/navigation/courseGreatCircle/activeRoute/startTime

Description: The time this route was activated

/vessels/<RegExp>/navigation/courseGreatCircle/nextPoint

Description: The point on earth the vessel's presently navigating towards

Fields:

- position (The position of nextPoint in two dimensions)
  - latitude
  - longitude

/vessels/<RegExp>/navigation/courseGreatCircle/nextPoint/distance

Units: m (Meter)

Description: The distance in meters between the vessel's present position and the nextPoint

/vessels/<RegExp>/navigation/courseGreatCircle/nextPoint/bearingTrue

Units: rad (Radian)

Description: The bearing of a line between the vessel's current position and nextPoint, relative to true north

/vessels/<RegExp>/navigation/courseGreatCircle/nextPoint/bearingMagnetic

Units: rad (Radian)

Description: The bearing of a line between the vessel's current position and nextPoint, relative to magnetic north

/vessels/<RegExp>/navigation/courseGreatCircle/nextPoint/velocityMadeGood

Units: m/s (Meters per second)

Description: The velocity component of the vessel towards the nextPoint

/vessels/<RegExp>/navigation/courseGreatCircle/nextPoint/timeToGo

Units: s (Second)

Description: Time in seconds to reach nextPoint's perpendicular) with current speed & direction

/vessels/<RegExp>/navigation/courseGreatCircle/previousPoint

Description: The point on earth the vessel's presently navigating from

Fields:

- position (The position of lastPoint in two dimensions)
  - latitude
  - longitude

/vessels/<RegExp>/navigation/courseGreatCircle/previousPoint/distance

Units: m (Meter)

Description: The distance in meters between previousPoint and the vessel's present position

---

/vessels/<RegExp>/navigation/racing

Description: Specific navigational data related to yacht racing.

/vessels/<RegExp>/navigation/racing/startLineStb

Description: Position of starboard start mark

/vessels/<RegExp>/navigation/racing/startLinePort

Description: Position of starboard start mark

/vessels/<RegExp>/navigation/racing/distanceStartline

Units: m (Meter)

Description: The current distance to the start line

/vessels/<RegExp>/navigation/racing/timeToStart

Units: s (Second)

Description: Time left before start

/vessels/<RegExp>/navigation/racing/timePortDown

Units: s (Second)

Description: Time to arrive at the start line on port, turning downwind

/vessels/<RegExp>/navigation/racing/timePortUp

Units: s (Second)

Description: Time to arrive at the start line on port, turning upwind

/vessels/<RegExp>/navigation/racing/timeStbdDown

Units: s (Second)

Description: Time to arrive at the start line on starboard, turning downwind

/vessels/<RegExp>/navigation/racing/timeStbdUp

Units: s (Second)

Description: Time to arrive at the start line on starboard, turning upwind

/vessels/<RegExp>/navigation/racing/distanceLayline

Units: m (Meter)

Description: The current distance to the layline

/vessels/<RegExp>/navigation/magneticVariation

Units: rad (Radian)

Description: The magnetic variation (declination) at the current position

/vessels/<RegExp>/navigation/magneticVariationAgeOfService

Units: s (Second)

Description: Seconds since the 1st Jan 1970 that the variation calculation was made

/vessels/<RegExp>/navigation/destination

Title: destination

Description: The intended destination of this trip

Fields:

- eta (RFC 3339 (UTC only without local offset) string representing date and time.), units: RFC 3339 (UTC) (undefined)
- waypoint (UUID of destination waypoint)

/vessels/<RegExp>/navigation/gnss

Title: gnss

Description: Global satellite navigation meta information

Fields:

- methodQuality (Quality of the satellite fix)

    - value, enum:
        - no GPS
        - GNSS Fix
        - DGNSS fix
        - Precise GNSS
        - RTK fixed integer
        - RTK float
        - Estimated (DR) mode
        - Manual input
        - Simulator mode
        - Error
- integrity (Integrity of the satellite fix)
    - value, enum:
        - no Integrity checking
        - Safe
        - Caution
        - Unsafe

/vessels/<RegExp>/navigation/gnss/satellites

Description: Number of satellites

/vessels/<RegExp>/navigation/gnss/antennaAltitude

Units: m (Meter)

Description: Altitude of antenna

/vessels/<RegExp>/navigation/gnss/horizontalDilution

Description: Horizontal Dilution of Precision

/vessels/<RegExp>/navigation/gnss/positionDilution

Description: Positional Dilution of Precision

/vessels/<RegExp>/navigation/gnss/geoidalSeparation

Description: Difference between WGS84 earth ellipsoid and mean sea level

/vessels/<RegExp>/navigation/gnss/differentialAge

Units: s (Second)

Description: Age of DGPS data

/vessels/<RegExp>/navigation/gnss/differentialReference

Description: ID of DGPS base station

/vessels/<RegExp>/navigation/headingMagnetic

Units: rad (Radian)

Description: Current magnetic heading of the vessel

/vessels/<RegExp>/navigation/headingTrue

Units: rad (Radian)

Description: The current true heading of the vessel

/vessels/<RegExp>/navigation/position

Title: position

Description: The position of the vessel in 2 or 3 dimensions (WGS84 datum)

Fields:

- longitude (Longitude), units: deg (Degree)
- latitude (Latitude), units: deg (Degree)
- altitude (Altitude), units: m (Meter)

/vessels/<RegExp>/navigation/attitude

Title: Attitude

Description: Vessel attitude: roll, pitch and yaw

Fields:

- roll (Vessel roll, +ve is list to starboard), units: rad (Radian)
- pitch (Pitch, +ve is bow up), units: rad (Radian)
- yaw (Yaw, +ve is heading change to starboard), units: rad (Radian)

/vessels/<RegExp>/navigation/rateOfTurn

Units: rad/s (Radian per second)

Description: Rate of turn (+ve is change to starboard)

/vessels/<RegExp>/navigation/speedOverGround

Units: m/s (Meters per second)

Description: Vessel speed over ground

/vessels/<RegExp>/navigation/speedThroughWater

Units: m/s (Meters per second)

Description: Vessel speed through the water

/vessels/<RegExp>/navigation/speedThroughWaterTransverse

Units: m/s (Meters per second)

Description: Transverse speed through the water (Leeway)

/vessels/<RegExp>/navigation/speedThroughWaterLongitudinal

Units: m/s (Meters per second)

Description: Longitudinal speed through the water

/vessels/<RegExp>/navigation/leewayAngle

Units: rad (Radian)

Description: Leeway Angle derived from the longitudinal and transverse speeds through the water

/vessels/<RegExp>/navigation/log

Units: m (Meter)

Description: Log value

/vessels/<RegExp>/navigation/trip

Description: Trip data

/vessels/<RegExp>/navigation/trip/log

Units: m (Meter)

Description: Trip log value

/vessels/<RegExp>/navigation/trip/lastReset

Description: Trip log reset time

/vessels/<RegExp>/navigation/state

Title: state

Description: Current navigational state of the vessel

/vessels/<RegExp>/navigation/anchor

Title: anchor

Description: The anchor data, for anchor watch etc

/vessels/<RegExp>/navigation/anchor/maxRadius

Units: m (Meter)

Description: Radius of anchor alarm boundary. The distance from anchor to the center of the boat

/vessels/<RegExp>/navigation/anchor/currentRadius

Units: m (Meter)

Description: Current distance to anchor

/vessels/<RegExp>/navigation/anchor/position

Title: position

Description: The actual anchor position of the vessel in 3 dimensions, probably an estimate at best

Fields:

- longitude (Longitude), units: deg (Degree)
- latitude (Latitude), units: deg (Degree)
- altitude (Altitude), units: m (Meter)

/vessels/<RegExp>/navigation/datetime

Description: Time and Date from the GNSS Positioning System

Fields:

- gnssTimeSource (Source of GNSS Date and Time), enum:
  - GPS
  - GLONASS
  - Galileo
  - Beidou
  - IRNSS
  - Radio Signal
  - Internet
  - Local clock

/vessels/<RegExp>/propulsion

Title: propulsion

Description: Engine data, each engine identified by a unique name i.e. Port_Engine

---

/vessels/<RegExp>/propulsion/<RegExp>

Description: This regex pattern is used for validation of the identifier for the propulsion unit

---

/vessels/<RegExp>/propulsion/<RegExp>/label

Description: Human readable label for the propulsion unit

/vessels/<RegExp>/propulsion/<RegExp>/state

Description: The current state of the engine

/vessels/<RegExp>/propulsion/<RegExp>/revolutions

Units: Hz (Hertz)

Description: Engine revolutions (x60 for RPM)

/vessels/<RegExp>/propulsion/<RegExp>/temperature

Units: K (Kelvin)

Description: Engine temperature

/vessels/<RegExp>/propulsion/<RegExp>/oilTemperature

Units: K (Kelvin)

Description: Oil temperature

/vessels/<RegExp>/propulsion/<RegExp>/oilPressure

Units: Pa (Pascal)

Description: Oil pressure

/vessels/<RegExp>/propulsion/<RegExp>/alternatorVoltage

Units: V (Volt)

Description: Alternator voltage

/vessels/<RegExp>/propulsion/<RegExp>/runTime

Units: s (Second)

Description: Total running time for engine (Engine Hours in seconds)

/vessels/<RegExp>/propulsion/<RegExp>/coolantTemperature

Units: K (Kelvin)

Description: Coolant temperature

/vessels/<RegExp>/propulsion/<RegExp>/coolantPressure

Units: Pa (Pascal)

Description: Coolant pressure

/vessels/<RegExp>/propulsion/<RegExp>/boostPressure

Units: Pa (Pascal)

Description: Engine boost (turbo, supercharger) pressure

/vessels/<RegExp>/propulsion/<RegExp>/engineLoad

Units: ratio (Ratio)

Description: Engine load ratio, 0<=ratio<=1, 1 is 100%

/vessels/<RegExp>/propulsion/<RegExp>/engineTorque

Units: ratio (Ratio)

Description: Engine torque ratio, 0<=ratio<=1, 1 is 100%

/vessels/<RegExp>/propulsion/<RegExp>/transmission

Description: The transmission (gear box) of the named engine

/vessels/<RegExp>/propulsion/<RegExp>/transmission/gear

Description: Currently selected gear the engine is in i.e. Forward, Reverse, etc.

/vessels/<RegExp>/propulsion/<RegExp>/transmission/gearRatio

Units: ratio (Ratio)

Description: Gear ratio, engine rotations per propeller shaft rotation

/vessels/<RegExp>/propulsion/<RegExp>/transmission/oilTemperature

Units: K (Kelvin)

Description: Oil temperature

/vessels/<RegExp>/propulsion/<RegExp>/transmission/oilPressure

Units: Pa (Pascal)

Description: Oil pressure

/vessels/<RegExp>/propulsion/<RegExp>/drive

Description: Data about the engine's drive.

/vessels/<RegExp>/propulsion/<RegExp>/drive/type

Description: The type of drive the boat has i.e Outboard, shaft, jet, etc.

Enum values:

- saildrive
- shaft
- outboard
- jet
- pod
- other

/vessels/<RegExp>/propulsion/<RegExp>/drive/trimState

Units: ratio (Ratio)

Description: Trim/tilt state, 0<=ratio<=1, 1 is 100% up

/vessels/<RegExp>/propulsion/<RegExp>/drive/thrustAngle

Units: rad (Radian)

Description: Current thrust angle for steerable drives, +ve is thrust to Starboard

/vessels/<RegExp>/propulsion/<RegExp>/drive/propeller

Description: Data about the drive's propeller (pitch and slip)

/vessels/<RegExp>/propulsion/<RegExp>/fuel

Description: Data about the engine's Fuel Supply

/vessels/<RegExp>/propulsion/<RegExp>/fuel/type

Description: Fuel type

Enum values:

- diesel
- petrol
- electric
- coal/wood
- other

/vessels/<RegExp>/propulsion/<RegExp>/fuel/used

Units: m3 (Cubic meter)

Description: Used fuel since last reset. Resetting is at user discretion

/vessels/<RegExp>/propulsion/<RegExp>/fuel/pressure

Units: Pa (Pascal)

Description: Fuel pressure

/vessels/<RegExp>/propulsion/<RegExp>/fuel/rate

Units: m3/s (Cubic meter per second)

Description: Fuel rate of consumption

/vessels/<RegExp>/propulsion/<RegExp>/fuel/economyRate

Units: m3/s (Cubic meter per second)

Description: Economy fuel rate of consumption

/vessels/<RegExp>/propulsion/<RegExp>/fuel/averageRate

Units: m3/s (Cubic meter per second)

Description: Average fuel rate of consumption

/vessels/<RegExp>/propulsion/<RegExp>/exhaustTemperature

Units: K (Kelvin)

Description: Exhaust temperature

/vessels/<RegExp>/electrical

Title: electrical

Description: Electrical data, each electrical device indentified by a unique name i.e. Battery_1

/vessels/<RegExp>/electrical/batteries

Description: Data about the vessel's batteries

/vessels/<RegExp>/electrical/batteries/<RegExp>

Title: Battery keyed by instance id

Description: Batteries, one or many, within the vessel

/vessels/<RegExp>/electrical/batteries/<RegExp>/temperature

Title: temperature

Description: Additional / unique temperatures associated with a battery

Fields:

- limitDischargeLower (Operational minimum temperature limit for battery discharge), units: K (Kelvin)
- limitDischargeUpper (Operational maximum temperature limit for battery discharge), units: K (Kelvin)
- limitRechargeLower (Operational minimum temperature limit for battery recharging), units: K (Kelvin)
- limitRechargeUpper (Operational maximum temperature limit for battery recharging), units: K (Kelvin)
- warnUpper (Upper operational temperature limit), units: K (Kelvin)
- warnLower (Lower operational temperature limit), units: K (Kelvin)
- faultUpper (Upper fault temperature limit - device may disable/disconnect), units: K (Kelvin)
- faultLower (Lower fault temperature limit - device may disable/disconnect), units: K (Kelvin)

/vessels/<RegExp>/electrical/batteries/<RegExp>/capacity

Title: capacity

Description: Data about the battery's capacity

/vessels/<RegExp>/electrical/batteries/<RegExp>/capacity/nominal

Units: J (Joule)

Description: The capacity of battery as specified by the manufacturer

/vessels/<RegExp>/electrical/batteries/<RegExp>/capacity/actual

Units: J (Joule)

Description: The measured capacity of battery. This may change over time and will likely deviate from the nominal capacity.

/vessels/<RegExp>/electrical/batteries/<RegExp>/capacity/remaining

Units: J (Joule)

Description: Capacity remaining in battery

/vessels/<RegExp>/electrical/batteries/<RegExp>/capacity/dischargeLimit

Units: J (Joule)

Description: Minimum capacity to be left in the battery while discharging

/vessels/<RegExp>/electrical/batteries/<RegExp>/capacity/stateOfCharge

Units: ratio (Ratio)

Description: State of charge, 1 = 100%

/vessels/<RegExp>/electrical/batteries/<RegExp>/capacity/stateOfHealth

Units: ratio (Ratio)

Description: State of Health, 1 = 100%

/vessels/<RegExp>/electrical/batteries/<RegExp>/capacity/dischargeSinceFull

Units: C (Coulomb)

Description: Cumulative discharge since battery was last full

/vessels/<RegExp>/electrical/batteries/<RegExp>/capacity/timeRemaining

Units: s (Second)

Description: Time to discharge to discharge limit at current rate

/vessels/<RegExp>/electrical/batteries/<RegExp>/lifetimeDischarge

Units: C (Coulomb)

Description: Cumulative charge discharged from battery over operational lifetime of battery

/vessels/<RegExp>/electrical/batteries/<RegExp>/lifetimeRecharge

Units: C (Coulomb)

Description: Cumulative charge recharged into battery over operational lifetime of battery

/vessels/<RegExp>/electrical/batteries/<RegExp>/associatedBus

Description: Name of BUS device is associated with

/vessels/<RegExp>/electrical/batteries/<RegExp>/voltage

Units: V (Volt)

Description: Voltage measured as close as possible to the device

/vessels/<RegExp>/electrical/batteries/<RegExp>/voltage/ripple

Units: V (Volt)

Description: DC Ripple voltage

/vessels/<RegExp>/electrical/batteries/<RegExp>/current

Units: A (Ampere)

Description: Current flowing out (+ve) or in (-ve) to the device

/vessels/<RegExp>/electrical/inverters

Description: Data about the Inverter that has both DC and AC quantities

/vessels/<RegExp>/electrical/inverters/<RegExp>

Title: Inverter

Description: DC to AC inverter, one or many, within the vessel

/vessels/<RegExp>/electrical/inverters/<RegExp>/dc

Title: DC Quantities

Description: DC common quantities

/vessels/<RegExp>/electrical/inverters/<RegExp>/dc/associatedBus

Description: Name of BUS device is associated with

/vessels/<RegExp>/electrical/inverters/<RegExp>/dc/voltage

Units: V (Volt)

Description: Voltage measured as close as possible to the device

/vessels/<RegExp>/electrical/inverters/<RegExp>/dc/voltage/ripple

Units: V (Volt)

Description: DC Ripple voltage

/vessels/<RegExp>/electrical/inverters/<RegExp>/dc/current

Units: A (Ampere)

Description: Current flowing out (+ve) or in (-ve) to the device

/vessels/<RegExp>/electrical/inverters/<RegExp>/dc/temperature

Title: temperature

Description: Temperature measured within or on the device

Fields:

- warnUpper (Upper operational temperature limit), units: K (Kelvin)
- warnLower (Lower operational temperature limit), units: K (Kelvin)
- faultUpper (Upper fault temperature limit - device may disable/disconnect), units: K (Kelvin)
- faultLower (Lower fault temperature limit - device may disable/disconnect), units: K (Kelvin)

/vessels/<RegExp>/electrical/inverters/<RegExp>/ac

Title: AC Quantities

Description: AC equipment common quantities

/vessels/<RegExp>/electrical/inverters/<RegExp>/ac/associatedBus

Description: Name of BUS source is assocated with (if applicable, may = NULL)

/vessels/<RegExp>/electrical/inverters/<RegExp>/ac/lineNeutralVoltage

Units: V (Volt)

Description: RMS voltage measured between phase and neutral.

/vessels/<RegExp>/electrical/inverters/<RegExp>/ac/lineLineVoltage

Units: V (Volt)

Description: RMS voltage measured between phases

/vessels/<RegExp>/electrical/inverters/<RegExp>/ac/current

Units: A (Ampere)

Description: RMS current

/vessels/<RegExp>/electrical/inverters/<RegExp>/ac/frequency

Units: Hz (Hertz)

Description: AC frequency.

/vessels/<RegExp>/electrical/inverters/<RegExp>/ac/reactivePower

Units: W (Watt)

Description: Reactive power

/vessels/<RegExp>/electrical/inverters/<RegExp>/ac/powerFactor

Description: Power factor

/vessels/<RegExp>/electrical/inverters/<RegExp>/ac/powerFactorLagging

Description: Lead/lag status.

Enum values:

- leading
- lagging
- error
- not available

/vessels/<RegExp>/electrical/inverters/<RegExp>/ac/realPower

Units: W (Watt)

Description: Real power.

/vessels/<RegExp>/electrical/inverters/<RegExp>/ac/apparentPower

Units: W (Watt)

Description: Apparent power.

/vessels/<RegExp>/electrical/inverters/<RegExp>/mode

Description: Mode of inverter

/vessels/<RegExp>/electrical/chargers

Description: Data about the Charger that has both DC and AC quantities

/vessels/<RegExp>/electrical/chargers/<RegExp>

Title: Charger

Description: Battery charger

/vessels/<RegExp>/electrical/chargers/<RegExp>/associatedBus

Description: Name of BUS device is associated with

/vessels/<RegExp>/electrical/chargers/<RegExp>/voltage

Units: V (Volt)

Description: Voltage measured as close as possible to the device

/vessels/<RegExp>/electrical/chargers/<RegExp>/voltage/ripple

Units: V (Volt)

Description: DC Ripple voltage

/vessels/<RegExp>/electrical/chargers/<RegExp>/current

Units: A (Ampere)

Description: Current flowing out (+ve) or in (-ve) to the device

/vessels/<RegExp>/electrical/chargers/<RegExp>/temperature

Title: temperature

Description: Temperature measured within or on the device

Fields:

- warnUpper (Upper operational temperature limit), units: K (Kelvin)
- warnLower (Lower operational temperature limit), units: K (Kelvin)

- faultUpper (Upper fault temperature limit - device may disable/disconnect), units: K (Kelvin)
- faultLower (Lower fault temperature limit - device may disable/disconnect), units: K (Kelvin)

/vessels/<RegExp>/electrical/chargers/<RegExp>/mode

Description: Charging mode i.e. float, overcharge, etc.

/vessels/<RegExp>/electrical/ac

Description: AC buses

/vessels/<RegExp>/electrical/ac/<RegExp>

Title: AC Bus keyed by instance id

Description: AC Bus, one or many, within the vessel

/vessels/<RegExp>/electrical/ac/<RegExp>/phase

Description: Single or A,B or C in 3 Phase systems

/vessels/<RegExp>/electrical/ac/<RegExp>/phase/(single)|([A-C])

Title: AC Quantities

Description: AC equipment common quantities

/vessels/<RegExp>/electrical/ac/<RegExp>/phase/(single)|([A-C])/associatedBus

Description: Name of BUS source is assocated with (if applicable, may = NULL)

/vessels/<RegExp>/electrical/ac/<RegExp>/phase/(single)|([A-C])/lineNeutralVoltage

Units: V (Volt)

Description: RMS voltage measured between phase and neutral.

/vessels/<RegExp>/electrical/ac/<RegExp>/phase/(single)|([A-C])/lineLineVoltage

Units: V (Volt)

Description: RMS voltage measured between phases

/vessels/<RegExp>/electrical/ac/<RegExp>/phase/(single)|([A-C])/current

Units: A (Ampere)

Description: RMS current

/vessels/<RegExp>/electrical/ac/<RegExp>/phase/(single)|([A-C])/frequency

Units: Hz (Hertz)

Description: AC frequency.

/vessels/<RegExp>/electrical/ac/<RegExp>/phase/(single)|([A-C])/reactivePower

Units: W (Watt)

Description: Reactive power

/vessels/<RegExp>/electrical/ac/<RegExp>/phase/(single)|([A-C])/powerFactor

Description: Power factor

/vessels/<RegExp>/electrical/ac/<RegExp>/phase/(single)|([A-C])/powerFactorLagging

Description: Lead/lag status.

Enum values:

- leading
- lagging
- error
- not available

/vessels/<RegExp>/electrical/ac/<RegExp>/phase/(single)|([A-C])/realPower

Units: W (Watt)

Description: Real power.

/vessels/<RegExp>/electrical/ac/<RegExp>/phase/(single)|([A-C])/apparentPower

Units: W (Watt)

Description: Apparent power.

/vessels/<RegExp>/notifications

Title: notifications

Description: Notifications currently raised. Major categories have well-defined names, but the tree can be extended by any hierarchical structure

---

/vessels/<RegExp>/notifications/mob

Description: Man overboard

Fields:

- method (Method to use to raise notifications)
- state (Current notification state), enum:
  - normal
  - alert
  - warn
  - alarm
  - emergency
- message (Message to display or speak)

/vessels/<RegExp>/notifications/fire

Description: Fire onboard

Fields:

- method (Method to use to raise notifications)
- state (Current notification state), enum:
  - normal
  - alert
  - warn
  - alarm
  - emergency
- message (Message to display or speak)

/vessels/<RegExp>/notifications/sinking

Description: Vessel is sinking

Fields:

- method (Method to use to raise notifications)
- state (Current notification state), enum:
  - normal
  - alert
  - warn
  - alarm
  - emergency
- message (Message to display or speak)

/vessels/<RegExp>/notifications/flooding

Description: Vessel is flooding

Fields:

- method (Method to use to raise notifications)
- state (Current notification state), enum:
  - normal
  - alert
  - warn
  - alarm
  - emergency
- message (Message to display or speak)

/vessels/<RegExp>/notifications/collision

Description: In collision with another vessel or object

Fields:

- method (Method to use to raise notifications)
- state (Current notification state), enum:
  - normal

    - alert
    - warn
    - alarm
    - emergency
- message (Message to display or speak)

/vessels/<RegExp>/notifications/grounding

Description: Vessel grounding

Fields:

- method (Method to use to raise notifications)
- state (Current notification state), enum:
    - normal
    - alert
    - warn
    - alarm
    - emergency
- message (Message to display or speak)

/vessels/<RegExp>/notifications/listing

Description: Vessel is listing

Fields:

- method (Method to use to raise notifications)
- state (Current notification state), enum:
    - normal
    - alert
    - warn
    - alarm
    - emergency
- message (Message to display or speak)

/vessels/<RegExp>/notifications/adrift

Description: Vessel is adrift

Fields:

- method (Method to use to raise notifications)
- state (Current notification state), enum:
  - normal
  - alert
  - warn
  - alarm
  - emergency
- message (Message to display or speak)

/vessels/<RegExp>/notifications/piracy

Description: Under attack or danger from pirates

Fields:

- method (Method to use to raise notifications)
- state (Current notification state), enum:
  - normal
  - alert
  - warn
  - alarm
  - emergency
- message (Message to display or speak)

/vessels/<RegExp>/notifications/abandon

Description: Abandon ship

Fields:

- method (Method to use to raise notifications)
- state (Current notification state), enum:

    - normal
    - alert
    - warn
    - alarm
    - emergency
- message (Message to display or speak)

/vessels/<RegExp>/notifications/<RegExp>

Description: This regex pattern is used for validation of the path of the alarm

---

/vessels/<RegExp>/steering

Title: steering

Description: Vessel steering data for steering controls (not Autopilot 'Nav Data')

/vessels/<RegExp>/steering/rudderAngle

Units: rad (Radian)

Description: Current rudder angle, +ve is rudder to Starboard

---

/vessels/<RegExp>/steering/rudderAngleTarget

Units: rad (Radian)

Description: The angle the rudder should move to, +ve is rudder to Starboard

---

/vessels/<RegExp>/steering/autopilot

Title: autopilot

Description: Autopilot data

/vessels/<RegExp>/steering/autopilot/state

Description: Autopilot state

/vessels/<RegExp>/steering/autopilot/mode

Description: Operational mode

/vessels/<RegExp>/steering/autopilot/target

Title: target

Description: Autopilot target

/vessels/<RegExp>/steering/autopilot/target/windAngleApparent

Units: rad (Radian)

Description: Target angle to steer, relative to Apparent wind +port -starboard

/vessels/<RegExp>/steering/autopilot/target/headingTrue

Units: rad (Radian)

Description: Target heading for autopilot, relative to North

/vessels/<RegExp>/steering/autopilot/target/headingMagnetic

Units: rad (Radian)

Description: Target heading for autopilot, relative to Magnetic North

/vessels/<RegExp>/steering/autopilot/deadZone

Units: rad (Radian)

Description: Dead zone to ignore for rudder corrections

/vessels/<RegExp>/steering/autopilot/backlash

Units: rad (Radian)

Description: Slack in the rudder drive mechanism

/vessels/<RegExp>/steering/autopilot/gain

Description: Auto-pilot gain, higher number equals more rudder movement for a given turn

/vessels/<RegExp>/steering/autopilot/maxDriveCurrent

Units: A (Ampere)

Description: Maximum current to use to drive servo

/vessels/<RegExp>/steering/autopilot/maxDriveRate

Units: rad/s (Radian per second)

Description: Maximum rudder rotation speed

/vessels/<RegExp>/steering/autopilot/portLock

Units: rad (Radian)

Description: Position of servo on port lock

/vessels/<RegExp>/steering/autopilot/starboardLock

Units: rad (Radian)

Description: Position of servo on starboard lock

/vessels/<RegExp>/tanks

Title: tanks

Description: Tank data, each tank indentified by a unique name i.e. FreshWater_2

/vessels/<RegExp>/tanks/freshWater

Description: Fresh water tank (drinking)

/vessels/<RegExp>/tanks/freshWater/<RegExp>

Description: Tank, one or many, within the vessel

/vessels/<RegExp>/tanks/freshWater/<RegExp>/name

Description: The name of the tank. Useful if multiple tanks of a certain type are on board

/vessels/<RegExp>/tanks/freshWater/<RegExp>/type

Description: The type of tank

Enum values:

- petrol
- fresh water
- greywater

- holding
- lpg
- diesel
- rum

/vessels/<RegExp>/tanks/freshWater/<RegExp>/capacity

Units: m3 (Cubic meter)

Description: Total capacity

/vessels/<RegExp>/tanks/freshWater/<RegExp>/currentLevel

Units: ratio (Ratio)

Description: Level of fluid in tank 0-100%

/vessels/<RegExp>/tanks/freshWater/<RegExp>/currentVolume

Units: m3 (Cubic meter)

Description: Volume of fluid in tank

/vessels/<RegExp>/tanks/wasteWater

Description: Waste water tank (grey water)

/vessels/<RegExp>/tanks/wasteWater/<RegExp>

Description: Tank, one or many, within the vessel

/vessels/<RegExp>/tanks/wasteWater/<RegExp>/name

Description: The name of the tank. Useful if multiple tanks of a certain type are on board

---

/vessels/<RegExp>/tanks/wasteWater/<RegExp>/type

Description: The type of tank

Enum values:

- petrol
- fresh water
- greywater
- holding
- lpg
- diesel
- rum

/vessels/<RegExp>/tanks/wasteWater/<RegExp>/capacity

Units: m3 (Cubic meter)

Description: Total capacity

/vessels/<RegExp>/tanks/wasteWater/<RegExp>/currentLevel

Units: ratio (Ratio)

Description: Level of fluid in tank 0-100%

/vessels/<RegExp>/tanks/wasteWater/<RegExp>/currentVolume

Units: m3 (Cubic meter)

Description: Volume of fluid in tank

/vessels/<RegExp>/tanks/blackWater

Description: Black water tank (sewage)

/vessels/<RegExp>/tanks/blackWater/<RegExp>

Description: Tank, one or many, within the vessel

/vessels/<RegExp>/tanks/blackWater/<RegExp>/name

Description: The name of the tank. Useful if multiple tanks of a certain type are on board

---

/vessels/<RegExp>/tanks/blackWater/<RegExp>/type

Description: The type of tank

Enum values:

- petrol
- fresh water
- greywater
- holding
- lpg
- diesel
- rum

/vessels/<RegExp>/tanks/blackWater/<RegExp>/capacity

Units: m3 (Cubic meter)

Description: Total capacity

/vessels/<RegExp>/tanks/blackWater/<RegExp>/currentLevel

Units: ratio (Ratio)

Description: Level of fluid in tank 0-100%

/vessels/<RegExp>/tanks/blackWater/<RegExp>/currentVolume

Units: m3 (Cubic meter)

Description: Volume of fluid in tank

/vessels/<RegExp>/tanks/fuel

Description: Fuel tank (petrol or diesel)

/vessels/<RegExp>/tanks/fuel/<RegExp>

Description: Tank, one or many, within the vessel

/vessels/<RegExp>/tanks/fuel/<RegExp>/name

Description: The name of the tank. Useful if multiple tanks of a certain type are on board

/vessels/<RegExp>/tanks/fuel/<RegExp>/type

Description: The type of tank

Enum values:

- petrol
- fresh water
- greywater
- holding
- lpg
- diesel
- rum

/vessels/<RegExp>/tanks/fuel/<RegExp>/capacity

Units: m3 (Cubic meter)

Description: Total capacity

/vessels/<RegExp>/tanks/fuel/<RegExp>/currentLevel

Units: ratio (Ratio)

Description: Level of fluid in tank 0-100%

/vessels/<RegExp>/tanks/fuel/<RegExp>/currentVolume

Units: m3 (Cubic meter)

Description: Volume of fluid in tank

/vessels/<RegExp>/tanks/lubrication

Description: Lubrication tank (oil or grease)

/vessels/<RegExp>/tanks/lubrication/<RegExp>

Description: Tank, one or many, within the vessel

/vessels/<RegExp>/tanks/lubrication/<RegExp>/name

Description: The name of the tank. Useful if multiple tanks of a certain type are on board

---

/vessels/<RegExp>/tanks/lubrication/<RegExp>/type

Description: The type of tank

Enum values:

- petrol
- fresh water
- greywater
- holding
- lpg
- diesel
- rum

/vessels/<RegExp>/tanks/lubrication/<RegExp>/capacity

Units: m3 (Cubic meter)

Description: Total capacity

/vessels/<RegExp>/tanks/lubrication/<RegExp>/currentLevel

Units: ratio (Ratio)

Description: Level of fluid in tank 0-100%

/vessels/<RegExp>/tanks/lubrication/<RegExp>/currentVolume

Units: m3 (Cubic meter)

Description: Volume of fluid in tank

/vessels/<RegExp>/tanks/liveWell

Description: Live tank (fish)

/vessels/<RegExp>/tanks/liveWell/<RegExp>

Description: Tank, one or many, within the vessel

/vessels/<RegExp>/tanks/liveWell/<RegExp>/name

Description: The name of the tank. Useful if multiple tanks of a certain type are on board

---

/vessels/<RegExp>/tanks/liveWell/<RegExp>/type

Description: The type of tank

Enum values:

- petrol
- fresh water
- greywater
- holding
- lpg
- diesel
- rum

/vessels/<RegExp>/tanks/liveWell/<RegExp>/capacity

Units: m3 (Cubic meter)

Description: Total capacity

/vessels/<RegExp>/tanks/liveWell/<RegExp>/currentLevel

Units: ratio (Ratio)

Description: Level of fluid in tank 0-100%

/vessels/<RegExp>/tanks/liveWell/<RegExp>/currentVolume

Units: m3 (Cubic meter)

Description: Volume of fluid in tank

/vessels/<RegExp>/design

Title: design

Description: Design/dimensional data of this vessel

/vessels/<RegExp>/design/displacement

Units: kg (Kilogram)

Description: The displacement of the vessel

/vessels/<RegExp>/design/draft

Title: draft

Description: The draft of the vessel

Fields:

- minimum (The minimum draft of the vessel), units: m (Meter)
- maximum (The maximum draft of the vessel), units: m (Meter)
- canoe (The draft of the vessel without protrusions such as keel, centerboard,

rudder), units: m (Meter)

/vessels/<RegExp>/design/length

Title: length

Description: The various lengths of the vessel

Fields:

- overall (Length overall), units: m (Meter)
- hull (Length of hull), units: m (Meter)
- waterline (Length at waterline), units: m (Meter)

/vessels/<RegExp>/design/keel

Title: keel

Description: Information about the vessel's keel

Fields:

- type (The type of keel.), enum:
  - long
  - fin
  - flare
  - bulb
  - wing
  - centerboard
  - kanting
  - lifting
  - daggerboard

/vessels/<RegExp>/design/keel/angle

Units: rad (Radian)

Description: A number indicating at which angle the keel currently is (in case of a canting keel), negative to port.

/vessels/<RegExp>/design/keel/lift

Units: ratio (Ratio)

Description: In the case of a lifting keel, centreboard or daggerboard, the part of the keel which is extended. 0 is 'all the way up' and 1 is 'all the way down'. 0.8 would be 80% down.

/vessels/<RegExp>/design/beam

Units: m (Meter)

Description: Beam length

/vessels/<RegExp>/design/airHeight

Units: m (Meter)

Description: Total height of the vessel

/vessels/<RegExp>/design/rigging

Title: rigging

Description: Information about the vessel's rigging

Fields:

- configuration (The configuration of the rigging)
- masts (The number of masts on the vessel.)

/vessels/<RegExp>/sails

Title: sails

Description: Sails data

/vessels/<RegExp>/sails/inventory

Description: An object containing a description of each sail available to the vessel crew

---

/vessels/<RegExp>/sails/inventory/<RegExp>

Description: 'sail' data type.

Fields:

- name (An unique identifier by which the crew identifies a sail)
- type (The type of sail)
- material (The material the sail is made from (optional))
- brand (The brand of the sail (optional))
- active (Indicates wether this sail is currently in use or not)
- area (The total area of this sail in square meters), units: m2 (Square meter)
- minimumWind (The minimum wind speed this sail can be used with), units: m/s (Meters per second)
- maximumWind (The maximum wind speed this sail can be used with), units: m/s (Meters per second)

/vessels/<RegExp>/sails/area

Description: An object containing information about the vessels' sails.

/vessels/<RegExp>/sails/area/total

Units: m2 (Square meter)

Description: The total area of all sails on the vessel

/vessels/<RegExp>/sails/area/active

Units: m2 (Square meter)

Description: The total area of the sails currently in use on the vessel

---

/vessels/<RegExp>/sensors

Title: sensors

Description: Sensors, their state, and data.

/vessels/<RegExp>/sensors/<RegExp>

Title: sensor

Description: This regex pattern is used for validation UUID identifier for the sensor

---

/vessels/<RegExp>/sensors/<RegExp>/name

Description: The common name of the sensor

/vessels/<RegExp>/sensors/<RegExp>/sensorType

Description: The datamodel definition of the sensor data. FIXME - need to create a definitions lib of sensor datamodel types

---

/vessels/<RegExp>/sensors/<RegExp>/sensorData

Description: The data of the sensor data. FIXME - need to ref the definitions of sensor types

---

/vessels/<RegExp>/sensors/<RegExp>/fromBow

Description: The distance from the bow to the sensor location

/vessels/<RegExp>/sensors/<RegExp>/fromCenter

Description: The distance from the centerline to the sensor location, -ve to starboard, +ve to port

/vessels/<RegExp>/performance

Title: performance

Description: Performance Sailing data including VMG, Polar Speed, tack angle, etc.

/vessels/<RegExp>/performance/polarSpeed

Units: m/s (Meters per second)

Description: The current polar speed based on current polar diagram, trueWindSpeed and truewindAngle.

/vessels/<RegExp>/performance/polarSpeedRatio

Units: ratio (Ratio)

Description: The ratio of current speed through water to the polar speed.

/vessels/<RegExp>/performance/velocityMadeGood

Units: m/s (Meters per second)

Description: The current velocity made good derived from the speed through water and appearant wind angle. A positive value is heading to upwind, negative to downwind.

/vessels/<RegExp>/performance/velocityMadeGoodToWaypoint

Units: m/s (Meters per second)

Description: The current velocity made good to the next waypoint derived from the speedOverGround, courseOverGround.

/vessels/<RegExp>/performance/beatAngle

Units: rad (Radian)

Description: The true wind beat angle for the best velocity made good based on current current polar diagram and trueWindSpeed.

/vessels/<RegExp>/performance/beatAngleVelocityMadeGood

Units: m/s (Meters per second)

Description: The velocity made good for the beat angle.

/vessels/<RegExp>/performance/beatAngleTargetSpeed

Units: m/s (Meters per second)

Description: The target speed for the beat angle.

/vessels/<RegExp>/performance/gybeAngle

Units: rad (Radian)

Description: The true wind gybe angle for the best velocity made good downwind based on current polar diagram and trueWindSpeed.

/vessels/<RegExp>/performance/gybeAngleVelocityMadeGood

Units: m/s (Meters per second)

Description: The velocity made good for the gybe angle

/vessels/<RegExp>/performance/gybeAngleTargetSpeed

Units: m/s (Meters per second)

Description: The target speed for the gybe angle.

/vessels/<RegExp>/performance/targetAngle

Units: rad (Radian)

Description: The true wind gybe or beat angle for the best velocity made good downwind or upwind based on current polar diagram and trueWindSpeed.

/vessels/<RegExp>/performance/targetSpeed

Units: m/s (Meters per second)

Description: The target speed for the beat angle or gybe angle, which ever is applicable.

/vessels/<RegExp>/performance/leeway

Units: rad (Radian)

Description: Current leeway

/vessels/<RegExp>/performance/tackMagnetic

Units: rad (Radian)

Description: Magnetic heading on opposite tack.

/vessels/<RegExp>/performance/tackTrue

Units: rad (Radian)

Description: True heading on opposite tack.

/resources

Title: resources

Description: Resources to aid in navigation and operation of the vessel including waypoints, routes, notes, etc.

/resources/charts

Title: chart

Description: A holder for charts, each named with their chart code

---

/resources/charts/<RegExp>

Description: A chart

Fields:

- name (Chart common name)
- identifier (Chart number)
- description (A description of the chart)
- tilemapUrl (A url to the tilemap of the chart for use in TMS chartplotting apps)
- region (Region related to note. A pointer to a region UUID. Alternative to geohash)
- geohash (Position related to chart. Alternative to region)
- chartUrl (A url to the chart file's storage location)
- scale (The scale of the chart, the larger number from 1:200000)

- chartLayers (If the chart format is WMS, the layers enabled for the chart.)
- bounds (The bounds of the chart. An array containing the position of the upper left corner, and the lower right corner. Useful when the chart isn't inherently geo-referenced.)
- chartFormat (The format of the chart), enum:
    - gif
    - geotiff
    - kap
    - png
    - jpg
    - kml
    - wkt
    - topojson
    - geojson
    - gpx
    - tms
    - wms
    - S-57
    - S-63
    - svg
    - other

/resources/routes

Title: route

Description: A holder for routes, each named with a UUID

/resources/routes/<RegExp>

Description: A route, named with a UUID

Fields:

- name (Route's common name)
- description (A description of the route)
- distance (Total distance from start to end), units: m (Meter)

- start (The waypoint UUID at the start of the route)
- end (The waypoint UUID at the end of the route)
- feature (A Geo JSON feature object which describes the route between the waypoints)
    - type, enum:
        - Feature
    - geometry
        - type, enum:
            - LineString
        - coordinates (An array of two or more positions)
- properties (Additional data of any type)
- id

/resources/notes

Title: notes

Description: A holder for notes about regions, each named with a UUID. Notes might include navigation or cruising info, images, or anything

---

/resources/notes/<RegExp>

Description: A note about a region, named with a UUID. Notes might include navigation or cruising info, images, or anything

Fields:

- title (Note's common name)
- description (A textual description of the note)
- region (Region related to note. A pointer to a region UUID. Alternative to position or geohash)
- geohash (Position related to note. Alternative to region or position)
- mimeType (MIME type of the note)
- url (Location of the note)

/resources/notes/<RegExp>/position

Title: position

Description: Position related to note. Alternative to region or geohash

Fields:

- longitude (Longitude), units: deg (Degree)
- latitude (Latitude), units: deg (Degree)
- altitude (Altitude), units: m (Meter)

/resources/regions

Title: region

Description: A holder for regions, each named with UUID

/resources/regions/<RegExp>

Description: A region of interest, each named with a UUID

Fields:

- geohash (geohash of the approximate boundary of this region)
- feature (A Geo JSON feature object which describes the regions boundary)
  - type, enum:
    - Feature
  - geometry
  - properties (Additional data of any type)
  - id

/resources/waypoints

Title: waypoints

Description: A holder for waypoints, each named with a UUID

/resources/waypoints/<RegExp>

Description: A waypoint, named with a UUID

/resources/waypoints/<RegExp>/position

Title: position

Description: The position in 3 dimensions

Fields:

- longitude (Longitude), units: deg (Degree)
- latitude (Latitude), units: deg (Degree)
- altitude (Altitude), units: m (Meter)

/resources/waypoints/<RegExp>/feature

Title: Feature

Description: A Geo JSON feature object

/resources/waypoints/<RegExp>/feature/type

Description: [missing]

Enum values:

- Feature

/resources/waypoints/<RegExp>/feature/geometry

Title: Point

Description: [missing]

/resources/waypoints/<RegExp>/feature/geometry/type

Description: [missing]

Enum values:

- Point

/resources/waypoints/<RegExp>/feature/geometry/coordinates

Description: A single position, in x,y order (Lon, Lat)

/resources/waypoints/<RegExp>/feature/properties

Description: Additional data of any type

/resources/waypoints/<RegExp>/feature/id

Description: [missing]

/version

Description: Version of the schema and APIs that this data is using in Canonical format i.e. V1.0.0.

Change Log

v0.0.1-1 (2017/03/19 16:07 +00:00)

- #348 Add note that meta.units MUST be returned for valid keys (@rob42)
- #345 Added trip.log, and trip.lastReset (@rob42)
- #340 Add slack badge (@bkp7)
- #336 Pattern for version (@bkp7)
- #328 Added timezoneRegion to environment.time (@bkp7)
- #321 Fix notifications schema (@bkp7)
- #335 Added illuminance (@rob42)
- #325 Improvements to the chart model (@emilecantin)
- #319 Add changelog generation (@tkurki)
- #327 Updated Schema so that date-time must be in UTC (@bkp7)
- #324 Define timestamp as JSON Schema date-time (@bkp7)
- #320 Fix schema files to be valid against http://json-schema.org/draft-04/schema# (@bkp7)
- #299 Generate documentation for object types from local files (@joux3)
- #312 Specify vessel context and leaf path for delta (@tkurki)
- #313 Itemize course properties (@tkurki)

v0.0.1-0 (2016/12/27 19:54 +00:00)

- #309 Improve Top level signalk overview (@sumps)
- #272 Update multiple values documentation (@tkurki)
- #301 Add support for dollarpath in FullSignalK (@tkurki)
- #306 Removed refererence to Boundary Layer (@sumps)
- #283 Add Transverse Water Speed and Leeway Angle (@sumps)
- #300 Expand on the Discovery Section (@timmathews)
- #290 Make processSchemaFiles produce keyswithmetadata.json (@tkurki)
- #284 Steering group in line with autopilot communication (@joabakk)
- #278 Validate missing schema references & fix missing references (@joux3)
- #270 Use wildcard context in example, mention wildcard (@tkurki)
- #265 Add Changelog (@tkurki)
- #274 Resolve relative references in the same file (@joux3)
- #269 RFC0004 :Replace JsonPath with wildcard in subscription paths (@tkurki)

- #225 Add defaults overlay (@timmathews)
- #257 Reorganized steering group, added test (@joabakk)
- #266 Cleanup of sources schema (@thomasonw)
- #260 Clarify gitbook-docs/README.md (@timmathews)
- #261 Clean ups (@thomasonw)
- #255 Gitbook Documentation mvp (@tkurki)
- #245 Make tank capacity numberValue (@tkurki)
- #238 Source handling for NMEA and non-NMEA sources (@tkurki)
- #232 Added missing message types (@rob42)
- #233 Added chart scale, and some new charts types (@rob42)
- #230 Added examples, and tidied descriptions (@rob42)
- #231 add description to enum values (@sailoog)
- #228 Include enums in keyswithmetadata.json (@tkurki)
- #221 Add GC/RL distinction (@tkurki)
- #223 Consistent Use of JSON-schema draft 04 Format (@timmathews)
- #222 Tank Senders only provide tank level value (@sumps)
- #211 Alternative proposal for "course" object in navigation (@fabdrol)
- #217 Reorganise temperatures a bit (@tkurki)
- #202 Add general & n2k specific info to sources (@tkurki)
- #193 Added racing parameters (@joabakk)
- #204 Renamed refridgeration to refrigerator (@joabakk)
- #201 Fix keys with metadata (@tkurki)
- #200 Added State of Health to Batteries (@sumps)
- #185 Reorganise electrical: remove ac/dc distinction, branches for equipment types (@tkurki)
- #186 Temp, humidity, pressure reorganisation (@tkurki)
- #195 Refactor tanks (@tkurki)
- #194 Unit cleanup and added prop slip dependancies (@tkurki)
- #190 Improve descriptions (@joabakk)
- #182 Replace maxRevolutions with zones usage (@tkurki)
- #175 Added agnostic target speed and target angle (@joabakk)
- #176 Remove legacy Alarm objects from Environment Tree (@sumps)
- #165 Add engine load & torque (@tkurki)
- #118 Multiple values (take 2) (@tkurki)
- #161 Convert zone in meta to an object (@rob42)
- #162 Rename alarms branch to notifications (@rob42)
- #142 Master waypoints ref (@rob42)
- #160 Remove alarm uuid from path as its not needed. (@rob42)

- #159 Update vessel.json (@joabakk)
- #152 Improve alarms - add alert and emergency (@rob42)
- #120 Improve propulsion (@tkurki)
- #119 Rework electrical dc: batteries (@tkurki)
- #151 Use SPDX abbreviation of the license (@tkurki)
- #148 Add schema and tests for endpoint discovery (@jboynes)
- #149 Remove unneeded references to lodash (@jboynes)
- #145 Added registration structure (@rob42)
- #139 Added lights to navigation (@rob42)
- #143 Add env.mode - take2 - Add source/timestamp and test (@rob42)
- #141 Added environment.mode (@rob42)
- #140 Added IMO, flag, port, and reg number (@rob42)
- #136 Add datetime with source information to the Signal K model (@tkurki)
- #137 Master add systime (@rob42)
- #132 Express air pressure change rate as a rate (@jboynes)
- #129 Replace last use of floatValue with numberValue (@jboynes)
- #128 Removed CallsignDsc (@sumps)
- #121 Add Heave to environment.json (@sumps)
- #117 Additional GNSS field added (@sumps)
- #114 Change all units to (derived) SI units (fix issue #30) (@canboat)
- #113 Implement my own suggestion in issue #112 (@canboat)
- #110 Identities, take two (@fabdrol)
- #109 Add NMEA0183 sentence and talker to source (@tkurki)
- #107 Fix schema references and add all subschemas for validation (@tkurki)
- #105 Housekeeping: cleanup of all schemas (@fabdrol)
- #103 Relocate current from navigation to environment (@timmathews)
- #101 Split angleTrue (@tkurki)
- #87 Improve validation (@tkurki)
- #88 Change pgn type to number (@tkurki)
- #81 Format the design group (@MariusVolkhart)
- #78 Format the propulsion group (@MariusVolkhart)
- #79 Format the electrical_dc group (@MariusVolkhart)
- #77 Format the resources group (@MariusVolkhart)
- #80 Format the navigation group (@MariusVolkhart)
- #82 Format the alarms group (@MariusVolkhart)
- #83 Added an UUID property to vessel.json (@fabdrol)
- #75 Format all enums to the same style (@MariusVolkhart)
- #70 Format vessel.json (@MariusVolkhart)

- #54 Pull of restructured ELECTRICAL JSON schema (@thomasonw)
- #52 Electrical, AC (@timmathews)
- #64 Message schemas (@rob42)
- #51 Group roll pitch and yaw (@timmathews)
- #65 Updates to the "design" group, in order to accommodate vessels with a variable keel or centerboard and some other changes (@fabdrol)
- #67 Separate group for sails (@fabdrol)
- #55 Labels for SignalK data items (@tkurki)
- #63 More lenient delta schema (@tkurki)
- #62 Change subschema loading (@tkurki)
- #61 Added n2kMappings (@tkurki)
- #59 Added performance group to vessel.json (@zapfware)
- #57 Define specification for Performance data (@zapfware)
- #58 Add waypoint.distanceActual, correct units for log & logTrip (@zapfware)
- #50 Fix propulsion and sensors and some cleanup (@rob42)
- #44 Reorganize current group (@timmathews)
- #43 Adds source and timestamp to activeRoute (@timmathews)
- #42 Rename navigation.currentRoute (@timmathews)
- #34 Update layout (@timmathews)
- #29 Delta schema & validation (@tkurki)
- #31 Standardize formatting in schema JSON files (@timmathews)
- #27 Changes related to n2k integration (@tkurki)
- #26 Move source/timestamp so it only occurs on leaf nodes (@rob42)
- #24 Indent with 4 spaces (@tkurki)
- #23 Wind angle & direction (@tkurki)
- #21 Added sensors, design data, and anchor data (@rob42)
- #17 Added definition for GNSS object (@fabdrol)
- #15 Resources - add headers etc (@rob42)
- #14 Try to add headers (@rob42)
- #12 Add signalk header to docsun page (@rob42)
- #11 Replace experimental JavaScript calls (@timmathews)
- #9 Resources (@rob42)
- #5 Added alarms group (@rob42)
- #6 Added basic principles (@rob42)
- #4 Fix to docson.js (@timmathews)
- #3 Separated schemas (@fabdrol)
- #1 Added instructions for publishing spec to GH Pages (@timmathews)