JSON AIS.en

HamFi

Sisällysluettelo

JSON AIS transmission protocol

This is a **work-in-progress** specification for passing parsed AIS data in human- and computer-readable JSON format. If you have ideas for this specification, please email Heikki Hannikainen, OH7LZB (hessu at hes dot iki dot fi).

This protocol is useful for transmitting AIS data between AIS receiver sites and live AIS database services, and also for transmitting AIS data between AIS database services which choose to trust each other and exchange data.

This specification was initially implemented by Lekkas Dimitris of marinetraffic.com (https://web.archive.org/web/20131106091747/http://www.marinetraffic.com/) , and documented by Heikki Hannikainen of aprs.fi (https://web.archive.org/web/20131106091747/http://aprs.fi/) . Additional feedback has been received from Tapio Sokura, OH2KKU.

Description of message structure

This document defines three levels of JSON messages.

First, after decoding the AIS packets they are encoded into **AIS packet messages**. They contain data from the AIS message itself.

[redacted]

lot of packets having the same transmission path ("ReceivingSite1", "aprs.fi", "marinetraffic.com"), so it is not necessary to transmit the path separately with each AIS packet message.

The packet group messages are then, optionally, enclosed in a **transport message**. The transport message contains identifiers for detecting that the received message is a valid JSON AIS message, and might at a later phase contain a protocol version number, if the protocol needs to be extended in a non-compatible way. The transport message can be used when packet group messages are transmitted over HTTP pushing/polling, but might not be necessary when a stream of packet group messages are transmitted over a persistent TCP connection, since the relevant handshaking information can be exchanged once when the connection is opened.

Message fields

Mandatory fields in all AIS packet messages

- **msgtype**
  - Integer. This is the type of the AIS message. 5 for ship the ship information packet, 1/2/3 for the position packet, which is sent more often. If the data is being converted from the ShipPlotter HTTP export format, it is no longer in AIS packet format, and the data can be sent with msgtype="sp", with fields of types 3 and 5 in the same packet message.
- **mmsi**
  - Integer. This is the MMSI number of the ship.
- **rxtime**
  - The UTC time when this message was received by the AIS receiving station, "YYYYMMDDHHMMSS". This field should be included in all JSON AIS packet messages. It will be used to filter out duplicate and old AIS messages when they are forwarded between database systems. The clock of the receiving station needs to be NTP synchronized to reliable clock servers, so that the timestamps can be trusted and used to figure out which packet from a certain ship is the most recent.
- **servertime**
  - This UTC timestamp ("YYYYMMDDHHMMSS") is only included when AIS packet messages are returned in HTTP polling (GET) reply. It specifies the time when that particular AIS message was received by the server generating the reply. This is the timestamp used when comparing the packets against the firsttime and lasttime URL parameters of the HTTP GET request.

AIS Packet type 5: ship & voyage static data

If you don't have data for a certain field, do not include that key at all.

- **imo**
  - Integer. The IMO number of the ship.
- **callsign**
  - String. The callsign of the ship.
- **shipname**
  - String. The name of the ship.
- **shiptype**
  - Integer. The type ID of the ship.
- **length**
  - Integer. The length of the ship in meters.
- **width**
  - Integer. The width of the ship in meters.
- **ref_front**
  - Integer. The distance in meters from the front of the ship to the GPS receiver (reported lat/lon).
- **ref_left**

- **destination**
  - String. The destination of the voyage.
- **eta**
  - Estimated time of arrival (ETA) in UTC, in the format of the *timestamp* field described above

AIS Packet types 1-3 and 18-19: ship position

Packet types 1-3 are used by AIS class A transponders (usually large commercial vessels) and types 18-19 by class B transponders (pleasure boats and other small vessels).

- **lat**
  - Float. The latitude for the current position of the ship in decimal degrees, south negative.
- **lon**
  - Float. The longitude for the current position in decimal degrees, west negative.
- **speed**
  - Float. The speed of the ship, in knots.
- **course**
  - Float. The course of the ship, in degrees, 0.0 - 359.9. -1 == not available.
- **heading**
  - Integer. The heading of the ship, in degrees, 0-359. -1 == not available.
- **status**
  - Integer. The AIS navigation status code of the ship.

AIS Packet type 24: class B vessel static information

Type 24 packets are different from most other AIS packets in that the data fields of the packet vary depending on the packet subtype, called part (number). The actual data fields are identical in their interpretation and contents to the corresponding class A (packet type 5) fields, except *vendorid* (doesn't exist for class A transponders).

- **partno**
  - Integer. Packet 24 part number. This is the only common field, in addition to mandatory fields described earlier, to all packets of this type. Currently defined part numbers are 0 (A) and 1 (B).

Part 0 (A) fields:

- **shipname**
  - String. Name of the vessel.

Part 1 (B) fields:

- **shiptype**
  - Integer. The type ID of the vessel.
- **vendorid**
  - String. Identifies the manufacturer and/or model of the AIS transponder.
- **callsign**
  - String. Callsign of the vessel.
- **length**
  - Integer. Length of the ship in meters.
- **width**
  - Integer. Width of the ship in meters.
- **ref_front**
  - Integer. Distance in meters from the front of the ship to the GPS receiver (reported lat/lon).
- **ref_left**

P

- **path**
  - Array, describing the path this message has traveled. When a message is forwarded, the system transmitting the message **prepends** it's information (name and optional URL) in the path. The last entry in this array is the AIS receiving station. Each path component is an associative array containing two keys:
    - **name** - the name of the receiving site or database service (must be ASCII, character range: A-Z a-z 0-9 . - _ /)
    - **url** - an URL for the receiving site
- **msgs**
  - An array of **AIS packet messages** which have traveled the same path.

Example messages

These examples have been formatted with extra whitespace for readability. JSON doesn't care about whitespace in the elements, so while these are valid JSON AIS messages, they should be transmitted in a more compact format (no line feeds and spaces between the elements).

AIS packet messages

An AIS position report for a single ship:

```
{
    "msgtype": 3,
    "mmsi":2320787,
    "status":14,
    "speed":0,
    "lon":-1.11023795604706,
    "lat":50.7996215820313,
    "course":0,
    "heading":0,
    "timestamp":"20081013091401"
}
```

An AIS ship/voyage information report:

```
{
    "msgtype": 5,
    "mmsi":211189000,
    "imo":8705383,
    "callsign":"DQEJ",
    "shipname":"SASSNITZ",
    "shiptype":69,
    "length":0,
    "width":0,
    "eta":"20081014101000",
    "draught":58,
    "destination":"TRELLEBORG/SASSNITZ",
    "timestamp":"20081013091717"
}
```

Packet group messages

A complete packet group message with path data and a couple of ship positions. These positions were received by the receiving station called "OH7LZB", who gave it to marinetraffic, who in turn forwarded the data to aprs.fi.

```
{
        { "name": "OH7LZB" }
    ],
    "msgs": [
        {
             "msgtype": 3, "mmsi":2320787, "status":14, "speed":0,
             "lon":-1.11023795604706, "lat":50.7996215820313,
             "course":0, "heading":0, "timestamp":"20081013091401"
        },
        {
             "msgtype": 3, "mmsi":1850257, "status":14, "speed":4,
             "lon":-3.26425604706, "lat":50.264556,
             "course":30, "heading":32, "timestamp":"20081013091403"
        },
        {
             "msgtype": 3, "mmsi":9124762, "status":14, "speed":10,
             "lon":2.12512412, "lat":52.153213,
             "course":321, "heading":319, "timestamp":"20081013091405"
        },
        {
             "msgtype": 5, "mmsi":211189000, "imo":8705383, "callsign":"DQEJ",
             "shipname":"SASSNITZ", "shiptype":69, "length":0, "width":0,
             "eta":"20081015103000", "draught":58,
             "destination":"TRELLEBORG/SASSNITZ", "timestamp":"20081013091717"
        }
    ]
}
```


HTTP polling (GET)

This format is used for passing JSON AIS messages using a polling mechanism. A client receives AIS data from a server by doing HTTP GET requests from a specific URL (for example, http://ais-server.example.com/jsonais/get). The server should require authentication (for example, using HTTP BASIC authentication, or by using a secret URL component). If no authentication is done, and anyone is allowed to upload data without ways to restrict abuse, the system will eventually be spammed or abused with rubbish data.

The client can request data updated within a specific time range by using the **firsttime** and **lasttime** parameters. Time is specified in UTC format, YYYYMMDDHHMMSS (13 October 2008 12:34:56 UTC would be 20081013123456). The matching logic is **firsttime <= updatetime < lasttime**. With a polling interval of 60 seconds, at t=20081013120100 the client would ask for firsttime=20081013120000&lasttime=20081013120100, and at t=20081013120200 it would ask for firsttime=20081013120100&lasttime=20081013120200). This way the client will only receive each updated point once, and the server can optimize it's backend database queries too.

The time used in the firsttime/lasttime queries is UTC time when the **packet message** was received by the server. This corresponds to the **servertime** key in the **packet message**, not rxtime. This ensures that all packets are correctly transferred from the server to the client.

The client can also require excluding data which has passed through a specified JSON AIS server. This is done using the URL parameter **excludepath**. For example, **excludepath=aprs.fi** would filter out all entries which have "name"="aprs.fi" in the path.

The JSON AIS messages are returned by the server in **transport message** which specifies the protocol name (jsonais) and contains an array of **packet group messages**. This container is used so that we can extend the protocol later by adding fields here.

Fields of the HTTP transport message:

- **protocol**

[redacted]

Should be used by the decoding system to check whether the clock of the other system is synchronized, or if the systems are lagging. The decoded message should be discarded, and an error message should be sent back if this timestamp varies by more than 5 seconds from the NTP synchronized clock of the decoding host.

- **groups**
  - An array of **packet group messages**.

The content-type given by the server should be **application/json**, which is the content-type specified by the JSON RFC.

Example:

```
{
    "protocol": "jsonais",
    "encodetime": "20081210215137",
    "groups": [
        { "path": [ ..path1... ], "msgs": [ { "msgtype: 5, .... }, { "msgtype": 3 }, ... ] },
        { "path": [ ..path2... ], "msgs": [ { "msgtype: 5, .... }, { "msgtype": 3 }, ... ] },
        { "path": [ ..path3... ], "msgs": [ { "msgtype: 5, .... }, { "msgtype": 3 }, ... ] }
    ]
}
```

HTTP push (POST)

This format is used for passing JSON AIS messages using a pushing mechanism. A client transmits AIS data to a server by doing HTTP POST requests to a specific URL (for example, http://ais-server.example.com/jsonais/post). This is generally the easiest way to post JSON AIS data from a receiving station to a server, as it will get through mostly all firewalls and proxies.

The array of AIS messages are formatted in a **HTTP transport message**, similarly to the HTTP GET method, and then encoded in a variable named **jsonais** of a HTTP POST request. Standard HTTP POST encoding methods are used - they can handle binary data, so JSON should go in just fine.

JSON AIS stream over TCP

This protocol is used for passing JSON AIS messages over a TCP stream. This has the least network overhead, since the connection is open all the time. It also provides the least latency, since the messages can be transmitted at any time, without a need for a polling timer.

A trusted server should authenticate all clients. If unauthenticated connections are allowed in a network of database services, it is certain that at some point rubbish and spam will enter the network.

Each connection only carries data to one direction - from the client to the server. If bidirectional communication is needed, two connections should be formed.

Login procedure

1. The client connects to the TCP port of the server
2. The server, optionally, checks the IP address of the client against an access list (white/blacklist)
3. The client sends a login message containing an username and a password
4. The server sends an acknowledge message to the client, either denying the connection or permitting it
5. The client starts sending AIS data to the server, in the form of **group messages**.

email addresses or plain names. Because this communication is typically not encrypted, the password should be specific to the AIS application. If the AIS server system has features which require a higher level of security, it might choose to use separate passwords for authenticating the users on the SSL-encrypted web site, and to assign separate random passwords for each user for the purpose of AIS packet feeding.

```
{
    "protocol": "jsonais",
    "command": "login",
    "username": "users.email@gmail.com",
    "password": "password"
}
```

Login ack packet format

```
{
    "protocol": "jsonais",
    "command": "login",
    "result": "ok",
    "description": "Login successful, welcome"
}
```

- **result**
  - String. Either "ok" or "fail".
- **description**
  - String. An english-language message describing the reason of a failed login ("internal error", "invalid username or password"), or a welcome message for a successful login. Please note, that saying "invalid username" or "invalid password" instead of "invalid username or password" decreases security - it helps an attacker with a dictionary-based attack by telling whether he guessed the username correctly.