NATS Message Headers

Metadata	Value
Date	2021-05-12
Author	@aricart, @scottf, @tbeets
Status	Implemented
Tags	server, client

Context

This document describes NATS Headers from the perspective of clients. NATS headers allow clients to specify additional meta-data in the form of headers. NATS headers are similar to HTTP Headers with some important differences.

As with HTTP headers:

- Each header field consists of a field name followed by a colon (`:`), optional leading whitespace, the field value, and optional trailing whitespace.
- No spaces are allowed between the header field name and colon.
- Field value may be preceded or followed by optional whitespace.
- The specification may allow any number of strange things like comments/tokens etc.
- The keys can repeat.

More specifically from rfc822 Section 3.1.2:

> Once a field has been unfolded, it may be viewed as being composed of a field-name followed by a colon ("😊, followed by a field-body, and terminated by a carriage-return/line-feed. The field-name must be composed of printable ASCII characters (i.e., characters that have values between 33. and 126., decimal, except colon). The field-body may be composed of any ASCII characters, except CR or LF. (While CR and/or LF may be present in the actual text, they are removed by the action of unfolding the field.)

Unique to NATS Headers

Version header

Instead of an HTTP method followed by a resource, and the HTTP version (`GET / HTTP/1.1`), NATS provides a string identifying the header version (`NATS/X.x`), currently 1.0, so it is rendered as `NATS/1.0␍␊`.

Case preserving

NATS treats application headers as a part of the message *payload* and is agnostic to the application use-case between publishers and subscribers; therefore, NATS headers are *case preserving*. The server will not change the case in message conveyance, the publisher's case will be preserved.

Any case sensitivity in header interpretation is the responsibility of the application and client participants.

> Note: This is *different* from HTTP headers which declare/define that web server and user-agent participants should ignore case.

With above caveats, please refer to the specification for information on how to encode/decode HTTP headers.

Enabling Message Headers

The server that is able to send and receive headers will specify so in it's `INFO` protocol message. The `headers` field if present, will have a boolean value. If the client wishes to send headers, it has to enable it must add a `headers` field with the `true` value in its `CONNECT` message:

```
"lang": "node",
"version": "1.2.3",
"protocol": 1,
"headers": true,
...
```

Publishing Messages With A Header

Messages that include a header have a `HPUB` protocol:

```
HPUB SUBJECT REPLY 23 30␍␊NATS/1.0␍␊Header: X␍␊␍␊PAYLOAD␍␊
HPUB SUBJECT REPLY 23 23␍␊NATS/1.0␍␊Header: X␍␊␍␊␍␊
HPUB SUBJECT REPLY 48 55␍␊NATS/1.0␍␊Header1: X␍␊Header1: Y␍␊Header2:
Z␍␊␍␊PAYLOAD␍␊
HPUB SUBJECT REPLY 48 48␍␊NATS/1.0␍␊Header1: X␍␊Header1: Y␍␊Header2: Z␍␊␍␊␍␊

HPUB <SUBJ> [REPLY] <HDR_LEN> <TOT_LEN>
<HEADER><PAYLOAD>
```

NOTES:

- `HDR_LEN` includes the entire serialized header, from the start of the version string (`NATS/1.0`) up to and including the ␍␊ before the payload
- `TOT_LEN` the payload length plus the HDR_LEN

MSG with Headers

Clients will see `HMSG` protocol lines for `MSG`s that contain headers

```
HMSG SUBJECT 1 REPLY 23 30␍␊NATS/1.0␍␊Header: X␍␊␍␊PAYLOAD␍␊
HMSG SUBJECT 1 REPLY 23 23␍␊NATS/1.0␍␊Header: X␍␊␍␊␍␊
HMSG SUBJECT 1 REPLY 48 55␍␊NATS/1.0␍␊Header1: X␍␊Header1: Y␍␊Header2:
Z␍␊␍␊PAYLOAD␍␊
HMSG SUBJECT 1 REPLY 48 48␍␊NATS/1.0␍␊Header1: X␍␊Header1: Y␍␊Header2: Z␍␊␍␊␍␊
HMSG <SUBJECT> <SID> [REPLY] <HDR_LEN> <TOT_LEN>
<PAYLOAD>
```

- `HDR_LEN` includes the entire serialized header, from the start of the version string (`NATS/1.0`) up to and including the ␍␊ before the payload
- `TOT_LEN` the payload length plus the HDR_LEN

Decision

Implemented and merged to master.

Consequences

Use of headers is possible.

Compatibility Across NATS Clients

The following is a list of features to insure compatibility across NATS clients that support headers. Because the feature in Go client and nats-server leverage the Go implementation as described above, the API used will determine how header names are serialized.

Case-sensitive Operations

In order to promote compatibility across clients, this section describes how clients should behave. All operations are *case-sensitive*. Application implementations should provide an option(s) to enable clients to work in a case-insensitive or format header names canonically.

Reading Values

`GET` and `VALUES` are case-sensitive operations.

- `GET` returns a `string` of the first value found matching the specified key in a case-sensitive lookup or an empty string.
- `VALUES` returns a list of all values that case-sensitive match the specified key or an empty/nil/null list.

Setting Values

- `APPEND` is a case-sensitive, and case-preserving operation. The header is set exactly as specified by the user.
- `SET` and `DELETE` are case-sensitive:
  - `DELETE` removes headers in case-sensitive operation
  - `SET` can be considered the result of a `DELETE` followed by an `APPEND`. This means only exact-match keys are deleted, and the specified value is added under the specified key.

Case-insensitive Option

The operations `GET`, `VALUES`, `SET`, `DELETE`, `APPEND` in the presence of a `case-insensitive` match requirement, will operate on equivalent matches.

This functionality is constrained as follows:

- `GET` returns the first matching header value in a case-insensitive match.
- `VALUES` returns the union of all headers that case-insensitive match. If the exact key is not found, an empty/nil/null list is returned.
- `DELETE` removes the all headers that case-insensitive match the specified key.
- `SET` is the combination of a case-insensitive `DELETE` followed by an `APPEND`.
- `APPEND` will use the first matching key found and add values. If no key is found, values are added to a key preserving the specified case.

> Note that case-insensitive operations are only suggested, and not required to be implemented by clients, specially if the implementation allows the user code to easily iterate over keys and values.

Multiple Header Values Serialization

When serializing, entries that have more than one value should be serialized one per line. While the http Header standard, prefers values to be a comma separated list, this introduces additional parsing requirements and ambiguity from client code. HTTP itself doesn't implement this requirement on headers such as `Set-Cookie`. Libraries, such as Go, do not interpret comma-separated values as lists.