httpmsgbus Documentation

Release 0.16

A. Heinloo

April 11, 2016

CHAPTER

ONE

INTRODUCTION

httpmsgbus (HMB) functions as a messaging service which runs over HTTP. It facilitates the transfer of objects, such as SeisComP 3 data model items, but also other content, between a server and a client. Messages sent by one client can be received by multiple clients connected to the same bus. JSON and BSON formats are used for communication.

A bus may have multiple *queues*. Order of messages within a queue is preserved. A queue may have multiple *topics*; topic name is simply an attribute of a message. A receiving client subscribes to one or more queues and tells which topics it is interested in.

Each message within a queue has a sequence number, so it is possible to resume connection without data loss, provided that the needed messages are still in the queue. A client can also select messages based on start- and end-time, and filter messages using a subset of MongoDB query language.

HMB supports out-of-order messages by letting a sending client specify the sequence number when sending messages. Messages are received in order; a receiving client may ignore out-of-order messages or wait for missing messages until a timeout.

httpmsgbus can be used as a standalone program or as an add-on to SeisComP 3.

CHAPTER

TWO

PROTOCOL DESCRIPTION

2.1 URL format

The URL has the following form:

{prefix}/{busname}/{method}[/{arg1}/{arg2}...]

prefix Arbitrary HTTP(S) prefix, such as http://localhost:8000

busname Arbitrary bus name (a single server can serve several busses at the same TCP port).

method See below.

arg1, arg2, ... Method-dependent arguments.

2.2 Function

Function defines the purpose of the server. The following functions are defined:

SC3MASTER SC3 master messaging server

WAVESERVER Waveform server

2.3 Capabilities

Capabilities define certain features and extensions to the core protocol. The following capabilities are defined:

JSON JSON format supported by /open, /send, /recv and /stream (cap:STREAM)

BSON BSON format supported (default)

INFO /info supported

STREAM /stream supported

WINDOW Time window requests supported

FILTER Filter supported

REGEX Filter supports the $regex operator (potential security risk with user-supplied regular expressions)

OOD Out-of-order messages supported

2.4 Data format

POST data can be in JSON (cap:JSON) or BSON (cap:BSON) format, the actual format is selected by the Content-Type header. The format of GET data is the same as the format used in /open (JSON or BSON). Sessionless methods (/features, /status, /info) use JSON format.

2.5 Mandatory methods

2.5.1 /features

Purpose: returns functions and capabilities supported by the server and optionally the name and version of the server software.

Arguments: none.

Response:

```
{
  "software": <string>,
  "functions": <list>,
  "capabilities": <list>
}
```

software Name and version of server software (optional).

functions List of functions.

capabilities List of capabilities.

2.5.2 /open

Purpose: opens a session, required by subsequent /send, /recv and /stream (cap:STREAM) methods.

Arguments: none

POST input:

```
{
  "cid": <string>,
  "heartbeat": <int>,
  "recv_limit": <int>,
  "queue": {
    <queue_name>: {
      "topics": <list of string>,
      "seq": <int>,
      "endseq": <int>,
      "starttime": <string>,
      "endtime": <string>,
      "filter": <doc>,
      "qlen": <int>,
      "oowait": <int>,
      "keep": <bool>
    },
    ...
  },
}
```

cid Requested client ID (optional).

heartbeat Heartbeat interval in seconds.

recv_limit Suggested maximum amount of kilobytes to return in one /recv call. The actual size can be slightly larger, depending on message size.

topics List of topics that the client is interested in. Wildcards ? and * are supported. A prefix '!' negates the pattern; message is delivered to the client if it matches any of the positive patterns and none of the negative patterns. None is equivalent to ["*"].

seq Starting sequence number. Negative numbers count from the end of the queue: -1 is the "next" message, -2 is the last message in the queue and so on. None is equivalent to -1.

endseq (cap:WINDOW) Ending sequence number. Can be used by clients to fill sequence gaps (seq..endseq).

starttime (cap:WINDOW) Request messages whose starttime..endtime overlaps with given starttime..endtime (time window request).

endtime (cap:WINDOW) Request messages whose starttime..endtime overlaps with given starttime..endtime (time window request).

filter (cap:FILTER) MongoDB style message filter. Operators $and, $or, $not, $nor, $eq, $gt, $gte, $lt, $lte, $ne, $in, $nin, $exists and $regex (cap:REGEX) are supported.

qlen (cap:OOD) Maximum queue length (last_sequence - current_sequence). When set, some messages can be discarded to make sure that the client does not fall too much behind real time.

oowait (cap:OOD) Maximum time to wait for out-of-order messages, in seconds.

keep Keep queue open after all data received (realtime mode).

Response: HTTP 400 with error message or

```
{
  "queue": {
    <queue_name>: {
      "seq": <int>,
      "error": <string>
    },
    ...
  },
  "sid": <string>,
  "cid": <string>
}
```

seq Actual sequence number (>= 0) or None if error.

error Error string (queue does not exist, invalid parameters, etc.). None if no error (seq must be set).

sid Session ID (required in subsequent /send, /recv and /stream methods).

cid Assigned client ID.

2.5.3 /status

Purpose: returns the status of connected clients (sessions).

Arguments: none.

Response:

```
{
  "session": {
    <sid>: {
      "cid": <string>,
      "address": <string>,
      "ctime": <string>,
      "sent": <int>,
      "received": <int>,
      "format": <string>,
      "heartbeat": <int>,
      "recv_limit": <int>,
      "queue": {
        <queue_name>: {
          "topics": <list of strings>,
          "seq": <int>,
          "endseq": <int>,
          "starttime": <string>,
          "endtime": <string>,
```

```
          "filter": <doc>,
          "qlen": <int>,
          "oowait": <int>,
          "keep": <bool>,
          "eof": <bool>
        },
        ...
      },
    },
    ...
  },
}
```

address Address of peer in ip:port format.

ctime Time when the session was created.

sent Number of bytes sent (client->server), not accounting HTTP headers and compression.

received Number of bytes received (server->client), not accounting HTTP headers and compression.

format JSON or BSON.

eof End of stream reached.

The remaining attributes have the same meaning as in /open above.

2.5.4 /send

Purpose: sends a message.

Arguments: /sid

sid The session ID received from /open.

POST input:

```
{
  "type": <string>,
  "queue": <string>,
  "topic": <string>,
  "seq": <int>,
  "starttime": <int>,
  "endtime": <int>,
  "data": <doc>
}
```

type Message type, eg., "SC3". Can be any string, except "HEARTBEAT" and "EOF".

queue Destination queue of the message, eg. "SC3MSG"

topic Optional topic/group of the message, eg., "PICK".

seq (cap:OOD) Optional sequence number of the message (if None or missing, the sequence number will be assigned by the server).

starttime (cap:WINDOW) Optional effective start time of the message.

endtime (cap:WINDOW) Optional effective end time of the message.

data Payload.

A heartbeat message can be sent to keep an idle session from expiring. The message is otherwise ignored by the server.

```
{
  "type": "HEARTBEAT"
}
```

JSON and BSON (cap:BSON) formats are supported. Multiple messages can be sent in one /send call; in case of BSON format, multiple messages must be concatenated. In case of JSON format, an array-style document must be sent (even if there is only a single message):

```
{
  "0": <msg>,
  "1": <msg>,
  ...
}
```

Response: HTTP 400 with error message or HTTP 204.

2.5.5 /recv

Purpose: receive a message.

Arguments: /sid[/queue/seq]

sid The session ID received from /open.

queue/seq Queue and sequence number of the last message received to ensure continuity in case of network errors (due to buffering, the server can otherwise not be sure that all messages have reached the client).

If sid is not known to server, HTTP 400 is returned and the client should proceed with /open to create a new session.

If queue/seq does not match queue/seq of last message sent, HTTP 400 is returned and the client should proceed with /open to create a new session. However, if queue/seq does match an earlier object, the server may roll back and continue.

Response: HTTP 400 with error message or

```
{
  "type": <string>,
  "queue": <string>,
  "topic": <string>,
  "sender": <string>,
  "seq": <int>,
  "starttime": <int>,
  "endtime": <int>,
  "data": <doc>
}
```

sender client ID of the sending client.

The remaining attributes have the same meaning as in /post.

Two special values are defined for type:

HEARTBEAT Heartbeat message.

EOF (cap:WINDOW) End of time window (or endseq) reached.

/recv blocks until at least one message (incl. HEARTBEAT) is available and then returns one or more messages. In case of JSON format, an array-style document is returned (even if the response only contains a single message):

```
{
  "0": <msg>,
  "1": <msg>,
  ...
}
```

2.6 Optional methods

2.6.1 /info (cap:INFO)

Purpose: returns the list of queues and topics and available data.

Arguments: none.

Response:

```
{
  "queue": {
    <queue_name>: {
      "startseq": <int>,
      "starttime": <string>,
      "endseq": <int>,
      "endtime": <string>,
      "topics": {
        <topic>: {
          "starttime": <string>,
          "endtime": <string>
        },
        ...
      },
    },
    ...
  },
}
```

startseq Sequence number of the first message in queue.

starttime Start time of the first message in queue if defined, otherwise null.

endseq Sequence number of the last message in queue + 1.

endtime End time of the last message in queue if defined, otherwise null.

topics Topics in the queue with optional starttime and endtime. The set of topics may not be exhaustive (a perfect implementation requires scanning the whole queue or using separate databases to keep track of available topics).

2.6.2 /stream (cap:STREAM)

Works like /recv, except that /stream sends an endless stream of messages and never returns. In case of JSON format, an array-style document is returned; since the document has no end, only a progressive JSON parser would be useful.

CHAPTER

THREE

SOFTWARE UTILITIES

3.1 httpmsgbus

3.1.1 Description

httpmsgbus is a server implementing the HMB protocol. It can use a database (-D) or RAM only. httpmsgbus usually runs behind a web server acting as a reverse proxy.

3.1.2 Database URL

Supported database types are mongodb and filedb. MongoDB is recommended in most scenarios. Filedb is suitable for binary messages with fixed size, such as waveform records. Database URL can take one of the following forms:

mongodb://server:port Use MongoDB server running at server:port. A repository will be created for each bus and a capped collection for each queue. The collection size is determined by the -q command-line parameter.

filedb:///directory?blocksPerFile=int&blocksize=int&bufsize=int&maxOpenFiles=int Use filedb in directory. A subdirectory will be created for each bus and a further subdirectory for each queue. Messages will be stored in (possibly sparse) files consisting of fixed size blocks. File name will be the sequence number of the first block in file. The total size of files in a directory is determined by the -q command-line parameter.

The following optional parameters are supported by filedb:

blocksPerFile (default 1024) Number of blocks per file.

blocksize (default 1024) Size of one block in bytes, maximum message size.

bufsize (default 65536) Read buffer size.

maxOpenFiles (default 800) Maximum number of file handles to keep open.

3.1.3 X-Forwarded-For

When HMB runs behind a reverse proxy, connections appear to originate from localhost or wherever the proxy is running. Using the X-Forwarded-For HTTP header, it is possible to determine the actual IP address of a client, which is then used for session counting (-c), etc.

3.1.4 Sequence difference into future

When using multiple servers, it may happen that a client requests a future message that has not reached the current server yet. Sequence numbers in future can be enabled with the -d command-line option. If the maximum difference is exceeded, the sequence number is considered invalid and the current seqence number is used.

3.1.5 Command-line

httpmsgbus [options]

-D string	Database URL
-F	Use X-Forwarded-For
-P int	TCP port (default 8000)
-V	Show program's version and exit
-b int	Buffer (RAM) size in messages per queue (default 100)
-c int	Connections (sessions) per IP (default 10)
-d int	Maximum sequence difference into future (default 0)
-h	Show help message
-p int	Maximum size of POST data in KB (default 10240)
-q int	Queue (MongoDB capped collection) size in MB (default 256)
-s	Log via syslog
-t int	Session timeout in seconds (default 120)

3.2 hmbseedlink

3.2.1 Description

hmbseedlink is a proxy implementing the SeedLink protocol on top of HMB. A SeedLink station maps to an HMB queue and a SeedLink stream maps to an HMB topic. The payload of an HMB message would be a Mini-SEED record.

3.2.2 Command-line

hmbseedlink [options]

-H string	Source HMB URL
-O string	Organization
-P int	TCP port (default 18000)
-V	Show program's version and exit
-c int	Connections per IP (default 10)
-h	Show help message
-q int	Limit backlog of records (queue length)
-s	Log via syslog
-t int	HMB timeout in seconds (default 120)
-w int	Wait for out-of-order data in seconds

3.3 wavefeed

3.3.1 Description

wavefeed is a program that runs a SeedLink plugin (normally chain_plugin) and sends the waveform data to HMB. Only log and Mini-SEED packets are supported.

3.3.2 Command-line

wavefeed [options]

-C string	Plugin command line
-H string	Destination HMB URL
-V	Show program's version and exit
-X string	Regex matching channels with unreliable timing
-b int	Maximum number of messages to buffer (default 1024)
-h	Show help message
-s	Log via syslog
-t int	HMB timeout in seconds (default 120)

CHAPTER

FOUR

INSTALLATION

4.1 Compiling and installing HMB

- Install Go from https://golang.org/dl/ and make sure that the "go" tool is in the path.
- Make sure that you have a working Internet connection and git is installed. Git will be used by the "go" tool to download some additional Go packages.
- Install PCRE development package for your Linux distribution (usually named pcre-devel or libpcre3-dev). If you are on a non-Linux platform or want to create a portable binary for Linux, it is possible to use the standard Go regexp package instead of PCRE by removing the "vendor" directory in HMB source. The standard Go regexp package may have a memory leak in its current version, so keep an eye on the memory consumption if you use it.
- Either copy HMB source code to SC3 source tree and use CMake or simply call the "install.sh" script included.

4.2 Configuring reverse proxy

HMB does not implement SSL, HTTP compression and authentication, so when providing the HMB service you usually want to have HMB behind a reverse proxy. Below is an example Apache configuration using mod_proxy and mod_deflate and implementing basic authentication. In some cases you may want to expose only a subset of methods or use different credentials for sending and receiving.

```
ProxyPass        /hmb/busname/ http://hmbserver:8000/busname/
ProxyPassReverse /hmb/busname/ http://hmbserver:8000/busname/

<Proxy http://hmbserver:8000/>
AuthType Basic
AuthName "HMB"
AuthUserFile /srv/www/hmbusers
Require valid-user
SetOutputFilter DEFLATE
SetInputFilter DEFLATE
</Proxy>
```

CHAPTER

FIVE

USE CASES

5.1 Sending and receiving simple JSON objects

Demo scripts send_json.py and receive_json.py are included in the distribution.

- Start httpmsgbus without arguments; in this case persistent storage is not used. Busses and queues are created on demand when first used.
- In another shell, call:

  ```
  $ python send_json.py notice "something happened"
  ```

- Now the bus and queue have been created, so you can subscribe to the queue by calling:

  ```
  $ python receive_json.py
  ```

- Any messages sent by send_json.py will now be received by receive_json.py.

JSON objects can be easily used in Javascript, for example a Javascript client may connect to HMB and display alerts on a web page.

When the httpmsgbus process in the above example is killed, all messages will be lost. It is possible to enable persistent storage using the -D command-line option, in which case the messages will be saved and available after a restart.

To save messages in files in a directory "filedb", use:

```
$ httpmsgbus -D filedb://filedb
```

To save messages in a MongoDB database, use:

```
$ httpmsgbus -D mongodb://localhost:27017
```

In the latter case, you can see your messages using the MongoDB shell:

```
$ mongo test
MongoDB shell version: 3.0.7
connecting to: test
> db.SYSTEM_ALERT.findOne()
{
        "_id" : ObjectId("5702e2e22ba8707f01375106"),
        "type" : "SYSTEM_ALERT",
        "queue" : "SYSTEM_ALERT",
        "sender" : "j5WknMw9h1wQUovw",
        "seq" : NumberLong(0),
        "data" : {
                "text" : "something happened",
                "level" : "notice"
        }
}
```

5.2 Sending and receiving binary objects

The BSON format can be used to embed binary data without space overhead. For example, the "eventpush" program (available in the "hmb-clients" repository) uses HMB to send messages containing compressed XML data. Such messages can be received using the "qmlreceiver" program.

5.3 Sending and receiving SC3 data model items

In SeisComP 3, HMB is disabled by default and can be enabled by adding the following options to ~/seiscomp3/etc/kernel.ini:

```
hmb.enable = true
hmb.port = 8000
```

Thereafter HMB can be configured with "scconfig" and started like any other SC3 module:

```
$ seiscomp start httpmsgbus
```

Modules like scimport can send messages to HMB instead of the Spread messaging server, for example:

```
$ echo "msggroups = AMPLITUDE,PICK,LOCATION,MAGNITUDE,EVENT,QC,INVENTORY,CONFIG" >scimport.cfg
$ seiscomp exec scimport --no-filter -o hmb://localhost:8000/test --console 1 -v
```

Likewise, messages can be received from HMB instead of Spread:

```
$ seiscomp exec scmm -H hmb://localhost:8000/test --console 1 -v
```

The "pick2hmb" program, included in the distribution, can be studied as a C++ example of sending SC3 objects to HMB.

5.4 Sending and receiving waveform data

The wavefeed SC3 module can be started to feed waveform data to HMB instead of a local SeedLink server.

```
$ seiscomp start wavefeed
```

Now an HMB recordstream can be used:

```
$ seiscomp exec scrttv -I hmb://localhost:8000/wave
```