SBBP: SIMPLE BULLETIN BOARD PROTOCOL

WILL DESTAFFAN

DECEMBER 2021

TABLE OF CONTENTS

1. INTRODUCTION

The objective of SBBP is to provide a simple, reliable, and easily-expandable protocol for facilitating shared communication through a bulletin-board. This is achieved through the TCP protocol, the specific details of which are described in Appendix A.

2. SBBP MODEL

The operation of SBBP is as follows:

2.1. SBBP CLIENT

1) The client establishes a TCP connection to the server on port 13037
2) The client sends an SBBP message to the server, consisting of an opcode followed by zero or more argument values.
3a) If the message was executed successfully, the SBBP server responds by echoing the opcode, followed by zero or more response values.
3b) If the message does not correspond to a successful response, the SBBP server responds with the error opcode “ERRORENC”, followed by a one-byte error-code.
4) Repeat from step 2 indefinitely until the connection is either closed, reset, or aborted.

2.2. SBBP SERVER

1) The server powers on, initializing a public board of id 0
2) The server accepts a TCP connection from port 13037, and dispatches a new thread to handle it. The main execution thread repeats from step 2 indefinitely until the server is closed, at which point it jumps to step 5
3) On receipt of a complete message from the client in the thread, the server responds with an appropriate response (see 2.1)
4) The thread returns to step 3 indefinitely until the connection is either closed, reset, or aborted, at which point the thread joins with the main execution.
5) All currently active threads close their connections, and the server terminates.

SBBP interprets a closure, reset, or abortion of the TCP connection (from either client or server) to be indicative

3. SBBP SPECIFICATION

3.1. DESCENDING TERMINATOR MODEL

The ‘Descending Terminator Model’ is a specification for a lossless ravel operation on an arbitrarily nested list-like data structure of unbounded length. This ‘data’ can be expressed by the following grammar:

```
      S ::= <list> terminator
<list> ::= <list> 1*{ separator <list> } | <atom>
<atom> ::= data | ε
```

Additionally to permit the lossless restructuring of ‘shape’ after unravelling, the separator terminal is incremented at each instance of direct left recursion. For example, the following data:

```
[1,2,[3,[4,5],[],6],7]
```

Would ravel to

```
1(SEP1)2(SEP1)3(SEP2)4(SEP3)5(SEP2)(SEP2)6(SEP1)7(TERMINATOR)
```

```
1(SEP1)2(SEP1)3(SEP2)4(SEP3)5(SEP2)(SEP2)6(SEP1)7(TERMINATOR)
1(SEP1)2(SEP1)3(SEP2)4(SEP3)5(SEP2)(SEP2)6(SEP1)7(TERMINATOR)
1(SEP1)2(SEP1)3(SEP2)4(SEP3)5(SEP2)(SEP2)6(SEP1)7(TERMINATOR)
                                  ^^
                                  ∟This 0-width space is also a terminal!
```

With such an encoding scheme, a trivial recursive-descent parser is easily able to reconstruct the separated, ravelled sentence back into an n-dimensional list structure.

By differentiating separators via a full-ordered scheme correlated with recursion depth, as opposed to a traditional method of nested ‘beginning’ and ‘ending’ separators (such as (), [], or {}), any legal SBBP sentence carries the following enviable property - a single pass through the sentence tokenized on the outermost separator can immediately detect the outermost dimension of the original sentence. This property is especially useful if the data is required to have a fixed count of possibly-<list> values, as a check for such would fail ‘fast’, and at an ‘obvious’ location!

3.2. SBBP IMPLEMENTATION OF DTM

SBBP's method of encoding, at its heart, a Descending Terminator Model with a few extra restrictions. Every communication via SBBP is encapsulated in 'frame', which consists of a Descending Terminator Model Sentence, with the first atom an 8-byte opcode.

3.2.1. SBBP CONTROL CHARACTERS

In the context of a Descending Terminator Model:

SBBP defines 'terminator' as 0xFF

SBBP defines the base separator as 0xFE. Every consecutive level of recursion ***decrements*** the separator by 0x01. Any separator less than or equal to 0xFB is automatically invalid.

An atom may contain any character except 0xFC, 0xFD, 0xFE, or 0xFF.

3.2.2. SBBP ATOMS

SBBP subdivides the DTM terminal 'data' into the following atom 'types':

byte: Exactly one byte of data.
string: One or more bytes of data.
integer: One or more bytes between 0x30 and 0x39, inclusive. These represent a stringly-typed decimal integer. For example, the byte sequence [0x35,0x31,0x32,0x34] may be interpreted as ASCII characters ['4','1','2','3'], corresponding to the integer 4123 (four thousand, one hundred, twenty-three). Finally, this integer is always non-negative, as there is no manner with which to express a negative value.
boolean: Exactly one byte of data, with the value 0x30 ('0') to mean False, or 0x31 ('1') to mean True.

2.2.3 SBBP ERROR CODES

A failed operation will, instead of echoing the calling opcode, instead return 'ERRORENC', followed by a **byte** atom as the next value. The value of this byte represents the following error, as described by the table:

0x00	Invalid Format	The message could not be reasonably interpreted as an SBBP sentence
0x01	Unknown Opcode	No valid SBBP opcode corresponds to the opcode given
0x02	Bad Argument Count	An incorrect number of atoms was supplied for the operation corresponding given opcode
0x03	Bad Argument Value	One or more of the provided atoms has an inappropriate value for the expected atom type for the operation corresponding to the given opcode
0x10	Board Missing	The specified board does not exist
0x11	Board Exists	The specified board already exists
0x12	Messages Missing	One or more of the specified messages do not exist on the specified board
0x20	No Permissions	The user does not have valid permissions to perform the given operation
0x30	Empty Result	The result returned a valid, yet empty response.

Conditions 0x00 through 0x03, inclusive, can be seen as ‘fault conditions’ - where an invalid response has been sent. (Indeed, a properly-designed client should catch all inputs which would trigger these conditions before even being sent!) As such, any and all ‘commands’ may return them if sufficiently mis-formatted by the client, and will not be listed as responses in section 3.

3. SBBP COMMANDS

The following commands are valid in SBBP - here, their calling conventions are listed unravelled in the form [opcode, *arguments]. For transmission, they must naturally be ravelled according to the SBBP's DTM - where, as described in section 2.2, the first atom is the byte-representation of the ASCII opcode.

CREATE_B(BOARD_ID, USER_ID)
Calling Convention: ["CREATE_B", integer, integer]
Description: Attempts to create a board on the SBBP server with the specified USER_ID
On Success: ["CREATE_B"]
On Failure:
Board Exists - The board with id BOARD_ID already exists!
["ERRORENC",0x11]

DELETE_B(BOARD_ID, USER_ID)
Calling Convention: ["DELETE_B", integer, integer]
Description: Attempts to delete a board on the SBBP server with the specified USER_ID. If the user specified by USER_ID did not create the board, the board is not deleted, and 0x20 (No Permissions) is returned
On Success: ["DELETE_B"]
On Failure:
Board Missing - No such board exists with id BOARD_ID on the server
["ERRORENC", 0x10]

No Permissions - The user specified by USER_ID is not the creator of the board specified.
["ERRORENC", 0x20]

GET_M_CT(BOARD_ID)
Calling Convention: ["GET_M_CT", integer]
Description: Attempts to return the total number of messages on the specified board.
On Success: ["GET_M_CT", integer] - The second atom is the number of messages on the board
On Failure:
Board Missing - No such board exists with id BOARD_ID on the server
["ERRORENC", 0x10]

GETNEWCT(BOARD_ID, USER_ID)
Calling Convention: [“GETNEWCT”, integer, integer]
Description: Attempts to return the total number of new messages for a user on the specified board. A new message is defined as a message that the user has not yet requested the message body for.
On Success: [“GETNEWCT”, integer] - The second atom is the number of messages on the board
On Failure:
Board Missing - No such board exists with id BOARD_ID on the server
[“ERRORENC”, 0x10]

POST_MSG(BOARD_ID, USER_ID, SUBJECT, TEXT)
Calling Convention: [“POST_MSG”, integer, integer, string, string]
Description: Attempts to post a message with subject SUBJECT and body TEXT to the board specified by BOARD_ID, as the user specified by USER_ID
On Success: [“POST_MSG”]
On Failure:
Board Missing - No such board exists with id BOARD_ID on the server
[“ERRORENC”, 0x10]

DELT_MSG(BOARD_ID, USER_ID, MESSAGE_ID)
Calling Convention: [“DELT_MSG”, integer, integer, integer]
Description: Attempts to delete a message with id MESSAGE_ID on the board specified by BOARD_ID as the user specified by USER_ID. If the message was not posted by the user referred to by USER_ID, the message is not deleted, and 0x20 (No Permissions) is returned.
On Success: [“DELT_MSG”]
On Failure:
Board Missing - No such board exists with id BOARD_ID on the server
[“ERRORENC”, 0x10]

Messages Missing - The message specified by MESSAGE_ID does not exist on the specified board
[“ERRORENC”, 0x12]

No Permissions - The user specified by USER_ID is not the creator of the message specified.
[“ERRORENC”, 0x20]

GET_MSGS(BOARD_ID, USER_ID, MESSAGE_IDS, SUBJECTS_ONLY, NEW_ONLY)
Calling Convention:
["GET_MSGS", integer, integer, [*integer[1]], boolean, boolean]
Description: Attempts to get all messages on the specified board as the specified user, pursuant to the filters specified by the last three parameters:
MESSAGE_IDS - If MESSAGE_IDS is not an empty list, only return messages on the board which ids are found within MESSAGE_IDS.
SUBJECTS_ONLY - Return each message with an implementation-defined body, as opposed to the message's actual body. The standard recommends, but does not enforce, the string "ignore". Additionally, messages read by a call with this flag enabled do not count as 'reading' the message for the purposes of new messages.
NEW_ONLY - Only return 'new' messages, specific to the given user. A new message is defined as a message that the user has not yet requested the message body for.

On Success: ["GET_MSGS",[*[integer, integer, integer, string, string][2]]]
The 5-tuple should be the following: [
Message's MESSAGE_ID
Message's creator's USER_ID
Floor of the Message's creation UNIX time (in seconds)
Message's subject
Message's body
]

On Failure:
Board Missing - No such board exists with id BOARD_ID on the server
["ERRORENC", 0x10]

Messages Missing - One or more messages specified in MESSAGE_IDS do not exist on the specified board
["ERRORENC", 0x12]

Result Empty - The query, after filtered, returned no messages.
["ERRORENC", 0x30]

---

[1] Zero or more integers
[2] Zero or more of the preceding 5-tuple - although, the protocol dictates that zero instances should instead return code 0x30

The SBBP standard defines these commands as required to be implemented - however, a server-owner may choose to add more commands and error-codes as they see fit, provided that the arguments and return-values are of SBBP terminal type **boolean, byte, integer,** or **string**.

5. SBBP STATE

5.1. SBBP STATE DEFINITIONS

A ‘board’ is defined as an object with the following information:
- A dynamic list of ‘messages’
- An **integer** ‘board’ ID
- An **integer** ‘user’ ID, corresponding to the user which created it

A ‘message’ is defined as an object with the following information
- A **unique, integer** ‘message’ ID
- An **integer** ‘user’ ID, corresponding to the user which created it
- An **integer** ‘timestamp’, the UNIX timestamp the message was created
- A **string** ‘subject’, the message’s subject
- A **string** ‘body’, the message’s body

5.2. SBBP STATE REQUIREMENTS

Each half-duplex of a given SBBP connection must maintain the following state:
- The TCP connection socket itself
- A message buffer, storing a best-effort ‘unbounded’ amount of bytes received from the TCP byte stream until encountering the SBBP terminator 0xFF.

5.2.1 SBBP CLIENT STATE

Aside from the state variables listed above, the SBBP Client need not store any additional state - although the standard recommends that the client store a USER_ID to facilitate natural operation from the point of view of a single, discrete user.

5.2.2 SBBP SERVER STATE

The SBBP server must store a copy of the state variables listed in section 5 for *each* connected client. Additionally, the SBBP server must store the following:

A dynamic list of ‘Boards’, initialized with a global board with id ‘0’

APPENDIX A: TCP

SBBP mandates the use of TCP, which permits a reliable ‘byte-stream’ complimenting the potentially-unbounded length of a message, as well as implicitly adding functionality for an intentional and incidental ‘close’ operation, the latter of which is handled by an underlying OS forcibly resetting the connection on application close.