The FloatHub Communications Protocol

Core Protocol Version: 1
Security Envelope Version: 2
This Document Version: 0.35

Bernhard Borges[*] and Thor Sigvaldason[†]

May 29, 2017

Abstract

We define a compact protocol by which a device equipped with sensors can communicate various readings (GPS location, battery voltages, etc.) to a data receiver over a simple socket connection. This specification includes a "core" protocol of how to markup data readings, an outer security envelope protocol, and a minimal handshake with the receiving server. This is the protocol used by the FloatHub vessel monitoring system.

[*] Modiot Labs (bb@modiot.com)
[†] Modiot Labs (thor@modiot.com)

1 Introduction

As part of designing a device to monitor marine vessel data, we needed a simple method to communicate this information. There were two main constraints on the protocol design: 1) the device is based around an 8-bit microcontroller architecture which has very limited storage and processing capabilities, and 2) it was assumed that bandwidth could be expensive. The first constraint precluded the uses of some relatively common data transmission approaches such as a json object exchanged over http. The second dictated that we avoid as much verbosity as was reasonably possible in the expression of the underlying data. The result is a very simple and compact approach to marking up marine related information and sending it to a receiving server.

Before turning to a more detailed description, we will briefly raise our thoughts on security. There are many potential concerns related to the transmission of boat data, particularly location information. Personal security while transiting dangerous areas, sensitive vessel statistics during racing events, and basic privacy all suggest that transmitting in the clear would be undesirable. Accordingly, we have baked a reasonably well understood and vetted encryption standard into the overall protocol as part of an outer envelope. This is not an optional feature.

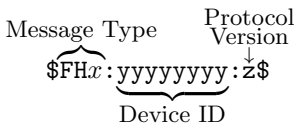
We used early drafts of this document internally during the development of the FloatHub system. It helped to make sure those working on the device side and those on the data receiver side had a concrete understanding of the correct format of the messages traveling from one to the other. We imagine the reader might use this specification to implement their own data receiving mechanism, perhaps for some other purpose than was originally intended by Modiot Labs and the FloatHub team. Others might be interested in building their own devices or modifying their existing FloatHubs to expand their use while still talking to our backend systems. We actively encourage both sorts of activities and are happy to receive feedback and questions.

2 Core Protocol

We use the term “core” protocol to distinguish it from the “security” protocol. The core specification explains how the data is marked up, while the security one explains how it is encrypted before transmission.

2.1 Message Structure

A message consists of a header and body. The header includes some identifying structure (i.e. `$` and `:` as element separators), a message type identifier, a device id, and a core protocol version number. The header looks like this:



To take a concrete example, if a FloatHub device with an id of `outofbox` where using core protocol version 1 to transmit an `FHA` message (see below), then the header would be:

```
$FHA:outofbox:1$
```

The body of the message then follows directly after the trailing $ and varies depending on the message type.

Note that as of this writing the core protocol specification is at version 1. That is, all transmissions which comply with the specification in this document should indicate 1 as their core protocol version. Also, note that the legal characters for the device id are any combination of letters and numbers and that these values are *case sensitive*. Blanks, punctuation, or other symbols are *not* permitted, meaning that it is an 8 character fixed field[1]. This gives $26 + 26 + 10 = 62$ characters for each of the eight positions, resulting in a namespace of 62^8 possible device ids (roughly 218 trillion).

2.2 Sample-Based Messages: `FHA`

The FloatHub device samples various sensors several times a second. At regular intervals it sends the current values of those samples to the receiving server using an `FHA` message. When the vessel is in motion these messages are communicated roughly twice a minute. When the vessel is stationary, they are sent approximately 10 minutes apart. The body of an `FHA` message consists of labeled data values that represent the sample values at the time they were measured. To take a concrete but simple example:

```
$FHA:outofbox:1$,U:11105809042014,T:68.80,P:29.6
```

Each piece of data starts with a comma (`,`), then a label (`U`, `T`, `P` etc.), followed by a colon (`:`) that separates the label from the actual value. In this example, we note that the sample time (`U`) was 11:10:58 am on April 9, 2014. Time is always formatted `hhmmssddMMYYYY` as detailed in Table 1 and should always be UTC (never adjusted for any local timezone).

If a time stamp is present, it is important to understand that this is the point in time where all data in this message were sampled. This could be hours,

[1] This is somewhat similar to the MAC address approach used by network adapters

Table 1: Time and Date Elements

Element	Description
`hh`	hours in 2 digit format
`mm`	minutes in 2 digit format
`ss`	seconds in 2 digit format
`dd`	day in 2 digit format
`MM`	month in 2 digit format
`YYYY`	year in 4 digit format

days, or conceivably weeks from the the time of transmission. The FloatHub device has a small amount of on-board EEPROM memory where it stores data it cannot successfully transmit. There is usually space for a few thousand such sets of data, so it is possible that after coming into range of a communications network a device might transmit many messages in rapid succession.

Conversely, if a time stamp is *not* present, it means that the device has some data to convey and has a communication uplink, but does not have a GPS fix or other reliable source of current UTC time. In this case, the best guess for the timestamp of the conveyed information is the time at which the message is received.

Table 2: `FHA` Data Labels and Types

Label	Type	Note
`U`	Timestamp	UTC Time in `hhmmssddMMYYYY` Format
`T`	Temperature	Degrees Fahrenheit
`P`	Pressure	Inches of Mercury (InHg)
`L`	Latitude	Degrees & Decimal Minutes (e.g. 43 15.3570N)
`O`	Longitude	Degrees & Decimal Minutes (e.g. 079 03.7733)
`A`	Altitude	Meters above Sea Level
`H`	HDOP	Horizontal Dilution of Precision of GPS Fix
`S`	SOG	Speed over Ground (in knots)
`B`	Bearing	Degrees True
`N`	Satellites	Number in View
`V`x	Battery	Voltage of Battery x
`C`x	Charger	Voltage of Charger x
`R`	STW	Speed through Water (in knots)
`D`	Depth	Depth Below Transducer (in meters)
`J`	Wind Speed	Knots
`K`	Wind Dir.	Degrees True
`Y`	Water Temp.	Degrees Fahrenheit

The full list of labels and data types is shown in Table 2. They are shown in the order likely to appear in current implementations, but this protocol makes

no guarantee about the order in which they will appear in the body of an FHA message.

2.3 Point-in-Time Messages: FHB

Where FHA messages are timer driven, reporting every so many number of seconds or minutes, FHB messages are event driven; something occurred *now*. The only current application of the FHB message is to pump events, but it is not difficult to imagine future extensions to other event driven information in a marine setting.

The structure is:

```
$FHB:outofbox:1$,U:03254015022015,P3:1
```

In this example, Pump 3 ($P3$) turned on (1) at 3:25:40 am on February 15, 2015. Unlike an FHA message, this does *not* mean that the pump happened to be on when the device sampled it. It means this is the timestamp of when the pump was turned on. When the pump turns off, it will send a similar message such as:

```
$FHB:outofbox:1$,U:04261315022015,P3:0
```

Indicating that the same pump turned off (0) at 04:26:13 am on the same day. This enables server side calculations of pump duration, frequency. etc.

Note that FHB analysis is complicated by the fact that communications can be intermittent and it is conceivable that the device will deliver out-of-order messages. Thus it is possible for a pump off message to arrive before a pump on message. It is also possible for FHB messages to be delivered without a timestamp (if there is no current GPS fix). As with the analogous case for FHA messages, the current time is the best guess for a timestamp.

In the most diabolical of cases, if the device sees a pump event but has no current time fix *and* no current communications link, it will not store the event for future transmission. Thus in the quite rare event that communications are going in and out while there is simultaneously no GPS fix, it is possible to receive a pump on message and never see the corresponding pump off message (or to see a pump go off when no message was received to indicate it had first turned on).

3 Outer Security Envelope: Base64/AES

As mentioned above, before an FHA or FHB message is sent, we encrypt it using AES. In slightly greater detail, we first encrypt using AES with cipher block chaining, the common AES-CBC approach. Each device has a 16 byte encryption key stored on it.[2] Because CBC also requires an initialization vector, we

[2]The key can be locally set or updated via the device's built-in web interface.

generate a pseudo-random one for each transmission but include it in the transmission itself along with the encrypted message. We then prepend an `FHS` header to the message in plain text so that the receiver knows whose transmission it is and thus how to decrypt it. The steps are:

1. Choose a pseudo-random `IV` (initialization vector)
2. Use the device's key and `IV` to encrypt the `FH`x message
3. Prepend the cyphertext with the `IV` in plain text
4. Base64 encode the `IV` and cyphertext concatenation
5. Prepend the above with an `FHS` header (including comma before body)
6. Transmit the above over a simple socket connection
7. Terminate the message transmission with \r (return) and \n (newline)

This results in a messages that looks like (shortened for readability, so not an actually decryptable message):

`$FHS:outofbox:2$,eGIfK/pryFff4gzPBpbI4ZFo9iO`\r\n

The protocol version in this example is 2, and note that this is the version of the security protocol we are describing here (as distinct from the core protocol). There was a security version 1 of this protocol that existed for some time during early development, but it is no longer supported.

Although AES-CBC is a reasonable encryption protocol in this context, it is important to point out that the device and its transmissions are still vulnerable to traffic analysis; a dedicated party could readily determine that transmissions from a given IP address are traveling to a given IP address, and use the unchanging textual elements `$FHS:yyyyyyyy:2$` as an identifier. Combined with IP location services, it is possible for such a third party to determine to close proximity where a set of transmissions from a given device are occurring. This is a risk of similar magnitude to tracking a cell phone based on it's IMEI number while on a cellular network or it's MAC/IP address while on a WiFi network.[3]

4 A Note on other `FH`x Message Types

If you connect a computer to the USB port on a FloatHub device, you may see messages that begin `$FH`x other than the `FHA`, `FHB`, or `FHS` types. If the code on a device has been compiled with one or more debugging flags turned on, there may be a *large* volume of these other message types. Strictly speaking, these are not part of the communications protocol; they are never sent out over the communications uplink. They were included to keep console output human readable but also make it readily parsable into different data types. Table 3 lists these other message types.

[3] More pessimistically put, this risk is in *addition* to cellular or WiFi tracking, since the device is susceptible to these risks as well.

Table 3: Other FHx Message Types

Header	Description
FHC	Console Message (similar to FHA, much more frequent)
FHD	Debugging Message
FHH	Help Message

5 Handshaking

The need for handshaking in this context is extremely limited. Because the devices generating and transmitting the data have very constrained computational and storage resources, they simply require an acknowledgment that the data was received *and* sensibly decoded. A failure to decode the Base64/AES envelope is equivalent to a traditional checksum error; somewhere in the creation, transmission, or decoding process, the message was corrupted.

Given the simple acknowledgment structure, we adopt the very simple convention that if the device receives back a message beginning:

```
$FHS$ OK
```

It is safe for it to consider the current message successfully stored on the remote data receiver. It can then delete that data and, optionally, send the next message whenever it is ready to do so.