# AYDP Communications Protocol

Autonomous Yacht Development Project
https://github.com/TimB-QNA/AYDP

April 27, 2016

# Contents

# Chapter 1

# Introduction

Chapter Text

# Chapter 2

# Message Construction

An AYDP message is constructed from three basic components; a header, a data section, and a checksum. The checksum at the end of each sentence is a single byte which is the XOR of all of the bytes in the sentence (excluding itself). This checksum byte is only intended to allow the recipient to ensure that the message is complete, not to re-build any missing data. All values are transmitted little-endian.

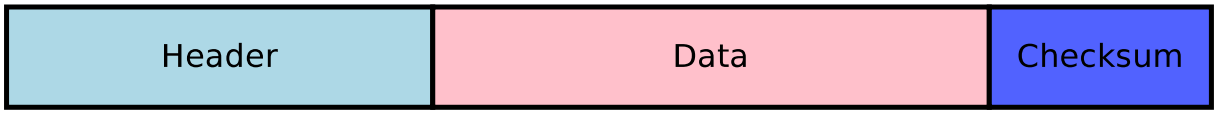


Figure 2.1: Basic protocol layout

The header is 27 bytes long and contains the message type and length as shown in table 2.1.

| Byte | Content | Data Type | Length (bytes) |
|---|---|---|---|
| 1 | Start (0xFF) | uint8 | 1 |
| 2 | Message Type | uint8 | 1 |
| 3 | Message Length | uint32 | 4 |
| 7 | Seconds from epoch | int64 | 8 |
| 19 | Milliseconds into current second | int64 | 8 |

Table 2.1: Header Contents

For message types $\geq 128$ (i.e. where the MSB of the message type is high) the timing data is omitted. This is termed an expedited message, and the header becomes only six bytes long. This is used to transfer high-speed data where the transmission time is secondary to the data rate. Any message can be expidited, simply by adding 128 to the message identifier.

## 2.1 Transmission limitations

The expidited message form is useful for serial links, but it should be noted that ethernet and WiFi links will hit packet rate limits before the bandwidth limits, except in the case of extremely large messages, where expedited transfer will make little difference anyway.

## 2.2 Protocol Robustness

The protocol robustness against random data is provided by a defined start byte (0xFF), the message type (which cannot be 0xFF), and a checksum. Occasionally, it is possible to improve

the robustness beyond these measures by checking the message length for known-length messages. This reduces the ease of updating the message format or data content, but will improve the error rejection to random noise.

# Chapter 3

# Message Types

The AYDP protocol supports a number of message types, given below.

| Message Type | Purpose | Link |
|---|---|---|
| 0 | Time sync | 3.1 |
| 1 | Wind speed and direction | 3.2 |
| 2 | Environmental conditions | 3.3 |
| 3 | Position | 3.4 |
| 4 | Position fix quality | 3.5 |
| 5 | Position fix detail | 3.6 |
| 6 | Vessel data | 3.7 |
| 7 | Direct output control | 3.8 |
| 8 | System value update | 3.9 |
| 9 | System control message | 3.10 |
| 100 - 126 | User defined message | 3.11 |
| 128 | Heartbeat | 3.12 |
| 255 | Reserved | 3.13 |

Table 3.1: Message Types

## 3.1 Time synchronisation

The time synchronisation message contains no data and is used to set the time on remote devices, by using the message time in the header. If this message is expidited, then it's meaning changes to that of a heartbeat see **??**.

## 3.2 Wind speed and direction

TBD

## 3.3 Environmental conditions

TBD

## 3.4 Position

Position

| Byte | Description | Type |
|---|---|---|
| 0-8 | Latitude | flt64 |
| 9-16 | Longitude | flt64 |
| 17-21 | Altitude | flt32 |

Table 3.2: Position information

## 3.5 Position fix quality

Fix Quality

| Byte | Description | Type |
|---|---|---|
| 0 | Fix Type | uint8 |
| 1-2 | Mean DOP | uflt16 |
| 3-4 | Horizontal DOP | uflt16 |
| 5-6 | Vertical DOP | uflt16 |
| 7 | Number of satellites | uint8 |

Table 3.3: Position fix quality information

## 3.6 Position fix detail

For the fix detail, bytes 1 to 7 are repeated for each Satellite. Thus a message for 10 satellites would be 71 bytes long.

| Byte | Description | Type |
|---|---|---|
| 0 | Number of satellites | uint8 |
| 1 | Satellite #n ID | uint8 |
| 2-3 | Satellite #n Azimuth | uflt16 |
| 4-5 | Satellite #n Elevation | uflt16 |
| 6-7 | Satellite #n Signal to Noise ratio | uflt16 |

Table 3.4: Position fix detail from GPS

## 3.7 Vessel Data

Time Sync

| Byte | Description | Type |
|---|---|---|
| 0 | Number of satellites | uint8 |
| 1 | Satellite #n ID | uint8 |
| 2-3 | Satellite #n Azimuth | uflt16 |
| 4-5 | Satellite #n Elevation | uflt16 |
| 6-7 | Satellite #n Signal to Noise ratio | uflt16 |

Table 3.5: Position fix detail from GPS

## 3.8 Direct Output Control

The ouput channels can be controlled directly, as long as direct control is enabled (table 3.9). This is typically reserved for testing and calibration, or for controlling the platform as one might a radio controlled boat.

The control system has a series of outputs, which may contain channels within them, consequently each address is made up of two integer values (device and channel) and a floating point value for the calibrated position. That is, to move a rudder +10 degrees on device 1, channel 2, the message content for that channel should read:

```
0x01 0x02 0x00 0x00 0x20 0x41
```

This message type supports setting an arbitrary number of channels, upto the addressable maximum (65,536) by specifying the device, channel and output data for each channel. If a channel is defined more than once, the last value is used as the active output.

| Byte | Description | Type |
|---|---|---|
| 0 | Number of channels | uint16 |
| 1 | Output device | uint8 |
| 2 | Output channel | uint8 |
| 3-7 | Calibrated output | flt32 |

Table 3.6: Direct output

## 3.9 System Value Update

The system value update message allows values within the software (such as control loop targets e.g. ordered heading or waypoints) to be changed on the fly. There are three modes in which a value may be updated, as specified in table 3.7.

| Update type | Value |
|---|---|
| Single Value | 1 |
| Entire Array | 2 |
| Element Within Array | 3 |

Table 3.7: Value update types

Each variable which can be changed must be registered at an address. This is typically hard-coded in the software, and therefore will vary dependant on the purpose of the software. The author of the software should give the address mapping for their code in their documentation. Some addresses are prescribed and are required for other messages in this protocol to provide the correct functionality. These are given in table 3.9.

The address space is 16 bits wide, giving 65,536 possible addresses which can be edited (some addresses are read-only). These addresses are not memory locations. They are in effect an array of pointers to other memory locations. This means that any address can contain any type of data, or an array of any type of data. The sending and recieving software must know what type of data is expected for each address, and therefore, type data is not included in the message. Only a single address can be updated in each message.

| Byte | Description | Type |
|---|---|---|
| 0 | Update type | uint8 |
| 1-2 | Address | uint16 |
| 3-6 | Array size or element | uint32 |
| 7- | Data | As required |

Table 3.8: System value update

Where a single value is edited, the “array size or element” entry is set to 0. If an entire array is being written, the array in this message over-writes the array in memory. This function

must be used with caution, as changing array length may cause the software to crash. The programmer should guard against this possibility.

When writing an array, the array size specifies the number of elements in the array, and the data entry is a contigious list of the array elements.

| Address | Description | Type | Access |
|---|---|---|---|
| 0x0000 | Enable direct control | uint8 (0/1) | read-write |
| 0x0001 | System Name | char[40] | read-only |
| 0x0002 | System ID | char[40] | read-only |
| 0x0003 | Hardware Version | char[40] | read-only |
| 0x0004 | Firmware Version | char[40] | read-only |

Table 3.9: Prescribed system addresses

For example, to enable direct control the following bytes would be sent:

```
0xFF 0x88 0x09 0x00 0x00 0x00 0x00 0x01 CS
```

## 3.10 System Control Message

The system control message allows the user to control the operation of the remote system, and request data. In some cases, such as with half-duplex communication, this may be preferable to a simple broadcast policy using system value update (3.9). The command message consists of a single byte indicating the command, then further bytes specifying the arguments to that command.

| Command | Value | Reference |
|---|---|---|
| Request data | 1 | 3.10.1 |
| Store configuration in non-volatile storage | 2 | 3.10.2 |

Table 3.10: System Control Message Commands

For example, to request the data at address 8, the following bytes would be sent:

```
0xFF 0x89 0x09 0x00 0x00 0x00 0x01 0x08 0x00 0x7F
```

While to save the configuration data, the following bytes would be sent:

```
0xFF 0x89 0x08 0x00 0x00 0x00 0x02 CS
```

### 3.10.1 Request Data

- Parameters

  Address

This command allows the user to request data stored at a particular address. The system responds to this command with a system value update (3.9) message. The svu message contains the required data, but does not give type information. If an address references an array, the whole array is returned.

### 3.10.2 Store configuration in non-volatile storage

- Parameters

  None

This command enables calibration or other configuration data to persist across system restarts. Typically, the remote system will store data in EEProm on microcontrollers or store to disk on larger systems. Issuing this command will save the current value of all non-volatile settings.

## 3.11 User defined messages

The message type range 100-126 is provided for user-defined messages. The message format should be cearly defined in the user's program documentation with reference to this protocol.

## 3.12 Heartbeat

The heartbeat message is the expedited form of the time synchronisation message (3.1). It serves to ensure that communication is still valid.

## 3.13 Reserved Message

The type 255 messages are provided to improve the error rejection properties of the protocol. If this message type is found, the first byte of the stream should be discarded, as the assumption is that the data stream has become a byte out of sync. This can happen in cases where the checksum of the previous message is 0xFF.

# Chapter 4

# Glossary

The following terms and abbreviations are used in this document.

## 4.1 Data Types

Data types are made up of two or three sections. The signedness, type and size. an unsigned integer capable of holding integer values from 0 to 255 would be described as uint8. That is an unsigned, integer of 8 bits in length. The signedness may be omitted for signed types, thus in the previous example, int8 would denote a type capable of storing integer values from -128 to +128. The abbreviations given in table 4.1 may be used to describe the data concerned.

| Abbreviation | Meaning |
|---|---|
| int | integer |
| flt | floating point |
| bool | unsigned integer containing flags |

Table 4.1: Message Types

# Chapter 5

# Bibliography