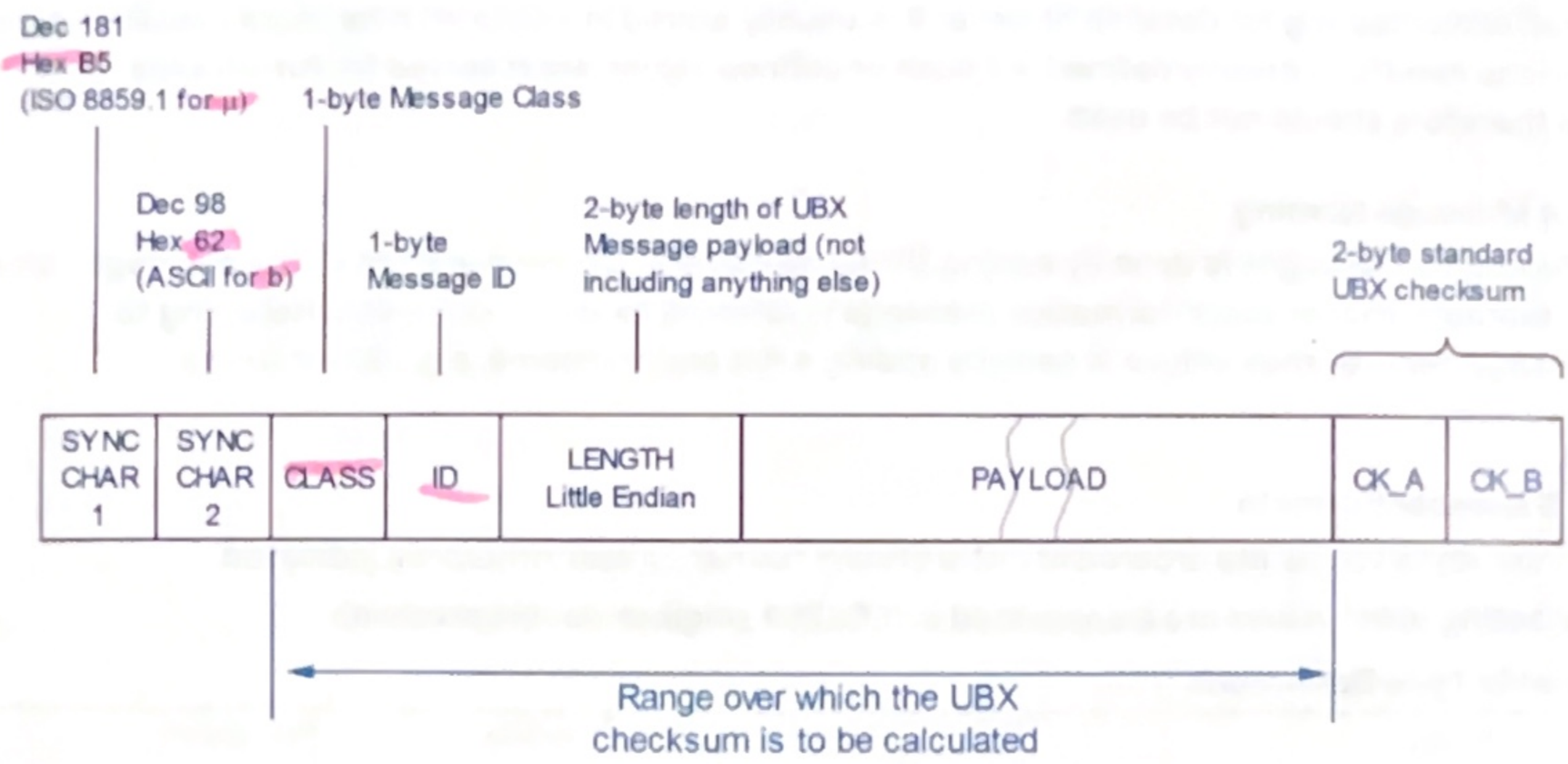
5.2 UBX Frame Structure

The structure of a basic UBX Frame is shown in the following diagram.



- Every **Frame** starts with a 2-byte Preamble consisting of two synchronization characters: 0xB5 0x62.
- A 1-byte Message **Class** field follows. A Class is a group of messages that are related to each other.
- A 1-byte Message **ID** field defines the message that is to follow.
- A 2-byte **Length** field follows. The length is defined as being that of the payload only. It does not include the Preamble, Message Class, Message ID, Length, or CRC fields. The number format of the length field is a Little-Endian unsigned 16-bit integer.
- The **Payload** field contains a variable number of bytes.
- The two 1-byte **CK_A** and **CK_B** fields hold a 16-bit checksum whose calculation is defined below. This concludes the Frame.

The checksum algorithm used is the 8-Bit Fletcher Algorithm, which is used in the TCP standard (RFC 1145). This algorithm works as follows:

- Buffer[N] contains the data over which the checksum is to be calculated.
- The two CK_ values are 8-Bit unsigned integers, only! If implementing with larger-sized integer values, make sure to mask both CK_A and CK_B with 0xFF after both operations in the loop.

```
CK_A = 0, CK_B = 0
For(I=0;I<N;I++)
{
    CK_A = CK_A + Buffer[I]
    CK_B = CK_B + CK_A
}
```

- After the loop, the two U1 values contain the checksum, transmitted after the Message, which conclude the Frame.

5.5.2 Polling Mechanism

All messages that are output by the receiver in a periodic manner (i.e. messages in classes MON, NAV and RXM) and Get/Set type messages, such as the messages in the CFG class, can also be polled.

The UBX protocol is designed so that messages can be polled by sending the message required to the receiver but without a payload (or with just a single parameter that identifies the poll request). The receiver then responds with the same message with the payload populated.

The UBX protocol can be read out over I2C. We will continually read data out of the 0xFF register. To determine how many bytes are available, we can read the 0xFD and 0xFE register. See page 87 for more details.

There are lots of messages we can poll over UBX. See the datasheet for the full list.

We care about NAV-PVT because it gives position velocity and time information.

Page	Mnemonic	Cls/ID	Length	Type	Description
154	**NAV-POSLLH**	0x01 0x02	28	Periodic/Polled	Geodetic Position Solution
155	**NAV-PVT**	0x01 0x07	92	Periodic/Polled	Navigation Position Velocity Time...
158	**NAV-RELPOSNED**	0x01 0x3C	64	Periodic/Polled	Relative Positioning Information in...
160	**NAV-RESETODO**	0x01 0x10	0	Command	Reset odometer
161	**NAV-SAT**	0x01 0x35	8 + 12*numSvs	Periodic/Polled	Satellite Information
163	**NAV-SIG**	0x01 0x43	8 + 16*numSi...	Periodic/Polled	Signal Information
165	**NAV-STATUS**	0x01 0x03	16	Periodic/Polled	Receiver Navigation Status
167	**NAV-SVIN**	0x01 0x3B	40	Periodic/Polled	Survey-in data
168	**NAV-TIMEBDS**	0x01 0x24	20	Periodic/Polled	BDS Time Solution
169	**NAV-TIMEGAL**	0x01 0x25	20	Periodic/Polled	Galileo Time Solution
170	**NAV-TIMEGLO**	0x01 0x23	20	Periodic/Polled	GLO Time Solution
171	**NAV-TIMEGPS**	0x01 0x20	16	Periodic/Polled	GPS Time Solution
172	**NAV-TIMELS**	0x01 0x26	24	Periodic/Polled	Leap second event information
174	**NAV-TIMEUTC**	0x01 0x21	20	Periodic/Polled	UTC Time Solution
176	**NAV-VELECEF**	0x01 0x11	20	Periodic/Polled	Velocity Solution in ECEF
176	**NAV-VELNED**	0x01 0x12	36	Periodic/Polled	Velocity Solution in NED

06/09/25 Wesley Cooke

Reading NAV-PVT on the ZEDF9P without a Library

also see page 41

GOAL

- [ ] Explore the UBX Protocol
- [ ] Request the NAV-PVT packet
- [ ] See the response of the NAV PVT packet

5.14.11 UBX-NAV-PVT (0x01 0x07)

5.14.11.1 Navigation Position Velocity Time Solution

Message	**UBX-NAV-PVT**
Description	**Navigation Position Velocity Time Solution**
Firmware	Supported on: • u-blox 9 with protocol version 27.11
Type	Periodic/Polled
Comment	Note that during a leap second there may be more or less than 60 seconds in a minute. See the section Leap seconds in Integration manual for details. This message combines position, velocity and time solution, including accuracy figures

Message Structure	Header	Class	ID	Length (Bytes)	Payload	Checksum
	0xB5 0x62	0x01	0x07	92	see below	CK_A CK_B

To request the NAV_PVT message, we have to send the above message with a length of 0 and no payload. (see page 90).

Example Code

- [ ] craft the packet
  - Compute the checksum (see page 89)
- [ ] Wait for GPSAvailable(); to have bytes ready (See Page 88)
- [ ] Read the data from the gps

Reading NAV-PVT on the ZEDF9P without a Library

06/09/25
Wesley Cook

```
21 void setup()
22 {
23   Wire.begin();        // Start the I2C bus
24   Serial.begin(9600); // Start serial comms
25 }
26
27 void loop()
28 {
29   // Message to send to request NAV_PVT.
30   // 0x01 is the Class
31   // 0x07 is the ID
32   // 0x00 and 0x00 is the length
33   // Per the data sheet: we send an empty packet to request the data.
34   uint8_t nav_pvt[4] = { 0x01, 0x07, 0x00, 0x00};
35
36   // Checksums
37   byte CK_A = 0;
38   byte CK_B = 0;
39
40   // Compute Checksum
41   for (int ii=0; ii<4; ii++)
42   {
43     CK_A = CK_A + nav_pvt[ii];
44     CK_B = CK_B + CK_A;
45   }
46
47   // Send the packet
48   Wire.beginTransmission(gps_add); ☐ Craft the packet
49   Wire.write(0xB5); // Sync Char1
50   Wire.write(0x62); // Sync Char2
51   Wire.write(0x01); // Class
52   Wire.write(0x07); // ID
53   Wire.write(0x00); // Length in Little Endian Order!
54   Wire.write(0x00);
55   Wire.write(CK_A); // Check sum!
56   Wire.write(CK_B);
57   Wire.endTransmission();
58
59   // Wait for the GPS to tell us it has a response
60   while(GPSAvailable() < 0)
61   {}
62
63   uint16_t availableBytes = GPSAvailable();
64
65   if (availableBytes == 100)
66   {
67     Serial.println("Got 100 Bytes!");
68     // We expect 100 bytes because there are 2 sync bytes, a class, an id, 2 length bytes, and 92
69     // 2 + 1 + 1 + 2 + 92 + 2 = 92 + 8 = 100!  // payload bytes and 2 checksum bytes.
70   }
71   else
72   {
73     Serial.println("Weird number of bytes...");
74   }
75
76   // Read the bytes available from the GPS and print them in HEX ☐ Read the data
77   Wire.requestFrom(gps_add, availableBytes);
78   for (int ii=0; ii<availableBytes; ii++)
79   {
80     Serial.print(Wire.read(), HEX); Serial.print(" ");
81   }
82   // new line between readings.
83   Serial.println();
84   delay(1500);
85 }
```

Example output