MAVLink Messages

MAVLink documentation can be found at: http://qgroundcontrol.org/mavlink/start

Physical Interface

Communication to a uAvionix transponder is accomplished through a full duplex asynchronous serial interface. The interface should use 3.3v logic levels. The baud rate on the NAV interface is 115200bps and on the HOST interface 57600bps. Both interfaces, no parity, 1 stop bits and 8 data bits. Any multi-byte data is formatted and transmitted as little-endian.

Message ID	Description	I/O	Length	CRC Extra
66	DataStream Request	Out	6	148
246	Traffic Report	Out	38	184
203	Status	Out	1	85
202	Ownship	Out	42	7
202	Dynamic	In	42	7
201	Static	In	19	126

MAVLink Messages

Interface Priority

The two interfaces on the ping share a priority system based upon their intended purpose. This system provides prioritization and redundancy since the same information could be provided over either interface. Here is a list of required messages and timeout's for each interface:

Messages Required	Description	Timeout	Preferred/Backup Interface
MAVLink Static (ID: 201)	0.1Hz	30s	Host/NAV
MAVLink Dynamic (ID: 202)	5Hz	5s	NAV/Host

Interface Priority

When a particular message has not been received through the preferred interface for the specified timeout, the data being received from that message switches from the preferred interface to the backup interface.

DataStream Request Message

Message ID	66
Payload Length	6
CRC_EXTRA	148

This is a legacy MAVLink message to request telemetry position messages from Ardupilot. Prior to v3.7(arduplane) and 3.4(arducopter)

Traffic Report Message

Message ID	246
Payload Length	38
CRC_EXTRA	184

Position	Field	Type	Description
0	ICAO_address	uint32_t	ICAO Address
4	lat	int32_t	The reported latitude in degrees * 1E7
8	lon	int32_t	The reported longitude in degrees * 1E7
12	altitude	int32_t	Altitude in Meters * 1E3 (up is +ve) - Check ALT_TYPE for reference datum
16	heading	uint16_t	Course over ground in degrees * 10^2
18	hor_velocity	uint16_t	The horizontal velocity in (m/s * 1E2)
20	ver_velocity	int16_t	The vertical velocity in (m/s * 1E2)
22	validFlags	uint16_t	Valid data fields
24	squawk	uint16_t	Mode A Squawk code (0xFFFF = no code)
26	altitude_type	uint8_t	Altitude Type
27	callsign	char[9]	The callsign
36	emitter_type	uint8_t	Emitter Category
37	tslc	uint8_t	Time since last communication in seconds

alitutdeType	
0x00: PRESSURE_ALTITUDE (AMSL, QNH) 0x01: GEOMETRIC (GNSS, WGS84)	
emitterType	
0x00: NO_TYPE_INFO 0x01: LIGHT_TYPE 0x02: SMALL_TYPE 0x03: LARGE_TYPE 0x04: HIGH_VORTEX_LARGE_TYPE 0x05: HEAVY_TYPE 0x06: HIGHLY_MANUV_TYPE 0x07: ROTOCRAFT_TYPE 0x08: UNASSIGNED_TYPE 0x09: GLIDER_TYPE	0x0A: LIGHTER_AIR_TYPE 0x0B: PARACHUTE_TYPE 0x0C: ULTRA_LIGHT_TYPE 0x0D: UNASSIGNED2_TYPE 0x0E: UAV_TYPE 0x0F: SPACE_TYPE 0x10: UNASSGINED3_TYPE 0x11: EMERGENCY_SURFACE_TYPE 0x12: SERVICE_SURFACE_TYPE 0x13: POINT_OBSTACLE_TYPE
flags	
0x0001: LATLON_VALID 0x0002: ALTITUDE_VALID 0x0004: HEADING_VALID 0x0008: VELOCITY_VALID 0x0010: CALLSIGN_VALID	0x0020: IDENT_VALID 0x0040 SIMULATED_REPORT 0x0080 VERTICAL_VELOCITY_VALID 0x0100 BARO_VALID 0x8000: SOURCE UAT

Status Message

Message ID	203
Payload Length	1
CRC_EXTRA	85

Position	Field	Type	Description
0	status	uint8_t	Self test status.

status
0x00: INITIALIZING 0x01: OK 0x02: TX_FAIL_1090ES 0x04: RX_FAIL_1090ES 0x08: TX_FAIL_UAT 0x10: RX_FAIL_UAT

Dynamic / Ownship Message

Message ID	202
Payload Length	42
CRC_EXTRA	7

Position	Field	Type	Description
0	utcTime	uint32_t	UTC time in since GPS epoch (in s since Jan 6, 1980). If unknown set to UINT32_MAX.
4	latitude	int32_t	Latitude WGS84 (deg * 1E7). If unknown set to INT32_MAX
8	longitude	int32_t	Longitude WGS84 (deg * 1E7). If unknown set to INT32_MAX.
12	altPres	int32_t	Barometric pressure altitude relative to a standard atmosphere of 1013.2 mBar and NOT bar corrected altitude (meters * 1E3) UP +ve. If unknown set to INT32_MAX.
16	altGNSS	int32_t	Altitude (meters * 1E3). (up +ve). WGS84 altitude. If unknown set to INT32_MAX.
20	accHoriz	uint32_t	Horizontal accuracy(HFOM) (mm). If unknown set to UINT32_MAX.
24	accVert	uint16_t	Vertical accuracy(VFOM) (cm). If unknown set to UINT16_MAX.
26	accVel	uint16_t	Velocity accuracy (m/s * 1E3). If unknown set to UINT16_MAX.
28	velVert	int16_t	GPS vertical speed (m/s * 1E2). If unknown set to INT16_MAX.
30	nsVog	int16_t	North-South velocity over ground (m/s * 1E2) North +ve. If unknown set to INT16_MAX.
32	ewVog	int16_t	East-West velocity over ground (m/s * 1E2) East +ve. If unknown set to INT16_MAX.
34	state	uint16_t	ADS-B input flags.
36	squawk	uint16_t	Mode A code (typically 1200 [0x04B0] for VFR).
38	fixType	uint8_t	GPS Fix.
39	numSats	uint8_t	Number of satellites visible. If unknown set to UINT8_MAX.
40	emStatus	uint8_t	Emergency status (table 2-78 of DO-260B).
41	control	uint8_t	ADS-B transponder dynamic input control flags.

state	
0x01: INTENT_CHANGE 0x02: AUTOPILOT_ENABLED 0x04: NICBARO_CROSSCHECKED 0x08: ON_GROUND 0x10: IDENT	
control	
0x00: STANDBY 0x01: RECEIVE ONLY 0x02: TX_ENABLE_1090 0x04: TX_ENABLE_UAT	0x08: MODE A ENABLED 0x10: MODE C ENABLED 0x20: MODE S ENABLED
fixType	
0x00: GPS_NO_FIX_0 0x01: GPS_NO_FIX_1 0x02: 2D_FIX 0x03: 3D_FIX 0x04: GPS_DGPS 0x05: GPS_RTK	

Example Packet

fe2a590000ca9574854523131f1653d945c800000000ddbb0500273201006e000f270000d4fe82000800b004030500000d57

Payload Length = 42	Packet Sequence = 0x59
System ID = 0	Component = 0
Message ID = 0xCA	utcTime = 1166374037
Latitude = 371135267	Longitude = -934946477
altPres = 0	altGNSS = 375773
accHoriz = 78375	accVert = 110
accVel = 9999	velVert = 0
nsVog = -300	ewVog = 130
State = 8	Squawk = 1200
fixType = 3	numSats = 5
emStatus = 0	Control = 0x06
CKA = 0x0D	CKB = 0x57

Static Message

Message ID	201
Payload Length	19
CRC_EXTRA	126

Position	Field	Type	Description
0	ICAO	uint8_t[3]	Vehicle address (24 bits). Byte[2] = msByte
3	integrity	uint8_t	System Integrity and Design Assurance
4	stallSpeed	uint16_t	Aircraft stall speed in cm/s.
6	callsign	char[8]	Vehicle identifier (8 characters, valid characters are A-Z, 0-9, " " only).
14	capability	uint8_t	Max Aircraft Speed and ADS-B in capability
15	emitter	uint8_t	Transmitting vehicle type.
16	alwEncode	uint8_t	Aircraft length and width encoding (table 2-35 of DO-282B). Upper Bound
17	gpsLatOffs	uint8_t	GPS antenna lateral offset (table 2-36 of DO-282B).
18	gpsLonOffs	uint8_t	GPS antenna longitudinal offset from nose [if non-zero, take position (in meters) divide by 2 and add one with max 60m] (table 2-37 DO-282B).

integrity	
0x00: SDA = 0 0x01: SDA = 1 0x02: SDA = 2 0x03: SDA = 3 0x00: SIL = 0 0x04: SIL = 1 0x08: SIL = 2 0x0C: SIL = 3	0x10: CSID 0x20: Force use of GNSS Altitude Data
capability	
0x00: Max Aircraft Speed Not Available 0x01: S ≤ 75 kts 0x02: 75 < S ≤ 150 kts 0x03: 150 < S ≤ 300 kts 0x04: 300 < S ≤ 600 kts 0x05: 600 < S ≤ 1200 kts 0x06: S > 1200 kts	0x00: No ADS-B in Capability 0x10: 1090MHz ADS-B in Capability 0x20: 978MHz ADS-B in Capability
alwEncode	
0x00: NO_DATA 0x01: AIRCRAFT_SIZE_L15M_W23M 0x02: AIRCRAFT_SIZE_L25M_W28P5M 0x03: SIZE_L25_W34M 0x04: SIZE_L35_W33M 0x05: SIZE_L35_W38M 0x06: SIZE_L45_W39P5M 0x07: SIZE_L45_W45M	0x08: SIZE_L55_W45M 0x09: SIZE_L55_W52M 0x0A: SIZE_L65_W59P5M 0x0B: SIZE_L65_W67M 0x0C: SIZE_L75_W72P5M 0x0D: SIZE_L75_W80M 0x0E: SIZE_L85_W80M 0x0F: SIZE_L85_W90M
gpsLatOffs	
0x00: GPS_LAT_OFFSET_NO_DATA	0x04: GPS_LAT_OFFSET_RIGHT_0M

0x01: GPS_LAT_OFFSET_LEFT_2M 0x02: GPS_LAT_OFFSET_LEFT_4M 0x03: GPS_LAT_OFFSET_LEFT_6M	0x05: GPS_LAT_OFFSET_RIGHT_2M 0x06: GPS_LAT_OFFSET_RIGHT_4M 0x07: GPS_LAT_OFFSET_RIGHT_6M
gpsLonOffs	
0: GPS_LON_OFFSET_NO_DATA 1: GPS_LON_OFFSET_APPLIED_BY_SENSOR 2-31: Compute: (meters/2 + 1)	

Example Packet

fe132f0000c93412a025000050494e4732303230001201040111fa

Payload Length = 19	Packet Sequence = 0x2F
System ID = 0	Component = 0
Message ID = 0xC9	ICAO = A01234
integrity = 0x25	stallSpeed = 0
emitter = 0x12	alwEncode = 0
Callsign = “PING2020”	Capability = 0
gpsLatOffs = 0x04	gpsLonOffs = 0x01
CKA = 0x11	CKB = 0xFA

MAVLink Protocol Frame Format

Each message will be formatted into a frame for transmission across the physical interface. Each frame must be transmitted in its entirety with the start of the frame being inferred by the beginning of any transfer initiated after the end of a previous frame. The end of a Frame will be delineated with an idle condition on the interface.

The uAvionix transponder is 'stateless' MAVLink packets do not have to be coordinated.

Ping OEM Frame 8 - 263 bytes

Byte Index	Content	Value	Description
0	Packet Start Flag	0xFE	Indicates the start of a new packet
1	Payload Length	0 - 255	Indicates the length of the payload
2	Packet Sequence	0 - 255	Send sequence. Allows to detect packet loss
3	System ID	0 - 255	ID of the Sending system. Allows to differentiate different PINGs on the same network
4	Component	0 - 255	ID of the Sending Component.
5	Message ID	0 - 255	ID of the Message - the 'id' defines what the payload "means" and how it should be correctly decoded.
6 to (n+6)	Data	0 - 255 Bytes	Message Data
(n+7) to (n+8)	Checksum (low Byte, high Byte)	ITU X.25/SAE AS-4 hash, **excluding packet start flag, so bytes 1..(n+6)** Note: The checksum also includes CRC_EXTRA	

CRC Code:
Code to validate the packet CRC.

```
#define X25_INIT_CRC 0xffff
#define X25_VALIDATE_CRC 0xf0b8

/**
 * @brief Accumulate the X.25 CRC by adding one char at a time.
 *
 * The checksum function adds the hash of one char at a time to the
 * 16 bit checksum (uint16_t).
 *
 * @param data - New char to hash
 * @param crcAccum - Already accumulated checksum
 **/
void crc_accumulate(uint8_t data, uint16_t *crcAccum)
{
   // Accumulate one byte of data into the CRC
   uint8_ttmp;

   tmp = data ^ (uint8_t)(*crcAccum&0xff);
   tmp ^= (tmp<<4);
   *crcAccum = (*crcAccum>>8) ^ (tmp<<8) ^ (tmp<<3) ^ (tmp>>4);
}
#endif


/**
 * @brief Initialize the buffer for the X.25 CRC
 *
 * @param crcAccum - 16 bit X.25 CRC
 */
void crc_init(uint16_t *crcAccum)
{
   *crcAccum = X25_INIT_CRC;
}

/**
 * @brief Calculates the X.25 checksum on a byte buffer
 *
 * @param  pBuffer - buffer containing the byte array to hash
 * @param  length - length of the byte array
 * @return the checksum over the buffer bytes
 **/
uint16_t crc_calculate(const uint8_t *pBuffer, uint16_t length)
{
   uint16_tcrcTmp;
   crc_init(&crcTmp);
   while (length--) crc_accumulate(*pBuffer++, &crcTmp);
   return crcTmp;
}

/**
 * @brief Accumulate the X.25 CRC by adding an array of bytes
 *
 * The checksum function adds the hash of one char at a time to the
 * 16 bit checksum (uint16_t).
 *
 * @param data - New bytes to hash
 * @param crcAccum - Already accumulated checksum
```

```
 **/
void crc_accumulate_buffer(uint16_t *crcAccum, const char *pBuffer, uint16_t length)
{
   const uint8_t *p = (const uint8_t *)pBuffer;
   while (length--) crc_accumulate(*p++, crcAccum);
}


// Note CRC_EXTRA is defined for each individual packet in the document.
crc_accumulate_buffer(&msg->checksum, _PAYLOAD(msg), msg->len);
crc_accumulate(CRC_EXTRA, &msg->checksum);
ck_a(msg) = (uint8_t)(msg->checksum & 0xFF);
ck_b(msg) = (uint8_t)(msg->checksum >> 8);
```