MavLink Tutorial for Absolute Dummies (Part –I)

What the hell is **MavLink**? It is a protocol for communication. People have been scared of it since this concept came out. This tutorial will force you to get it into your head and demystify on what it is, how it is and most importantly how the hell it works!! I will try to explain how the Mission Planner communicates with APM/ PX4 and vice-versa. This will help you with extensions and make your programmer genius come out from within you, i.e. if you not already are!!

This tutorial assumes:

1) You are a dummy ☹ I was also once upon a time, but no more!
2) You have at least limited programming skills in C (i.e. have programmed simple switch cases in C/C++/C#/Java). If you already are a pro, then return 0;
3) You are serious to learn, because you are going to lose some sleep!

But anyhow, have the willingness to learn, I promise you will never forget it ☺ Shall I begin?

I AM COMING FOR YOU, MAVLINK

The Mavlink message (we call it '*msg'*) is basically a stream of bytes that has been encoded by Mission Planner (MP) aka. GCS (Ground Station Control) and is sent to the APM via USB serial OR Telemetry (both cannot be used at the same time. If they are plugged in at the same time, USB is given preference and Telemetry is ignored). What *encoding* means is nothing special here but just to put the packet into a data structure and send it via the channel in bytes adding some error correction alongside.

STRUCTURE OF THE MAVLINK MESSAGE:

Each MavLink packet has a length of 17 bytes and here is the structure:

message length = 17 (6 bytes header + 9 bytes payload + 2 bytes checksum)

6 bytes header
0. message header, always 0xFE
1. message length (9)
2. sequence number -- rolls around from 255 to 0 (0x4e, previous was 0x4d)
3. **System ID** - what system is sending this message (1)
4. **Component ID**- what component of the system is sending the message (1)
5. **Message ID** (e.g. 0 = heartbeat and many more! Don't be shy, you can add too..)

Variable Sized Payload (specified in octet 1, range 0..255)
**** Payload** (the actual data we are interested in)

Checksum: For error detection.

What the software does is to check that it is a valid message (by checking the checksum and it is not corrupted, if so discards the message). That is one reason, why the baud rate for telemetry is set to 57,600 and not 115,200 bps. The lower it is the less errors it is prone to although slower it will be in updating to Ground station. If you want to get a greater distance with MavLink, it may be a good idea to reduce further the baud rate. However, note that the tested baud rate 57,600 bps would give, in theory, around a mile of radius coverage with 3DR telemetry radio. Remember Signal to noise ratio (SNIR) concept from high school☺?

Now, from the above, what we are interested in is:

a) **System ID** (aka. source of message): This is the source (i.e. Mission planner) sending a message to the APM via wireless Telemetry OR USB port. The software does a regular check so as to know that this message is for itself.

b) **Component ID** (aka. subsystem within the system): Any subsystem within the main system. Currently, there are no subsystems and we don't really make use of it.

c) **Message ID:** What is this message about. In this tutorial, we call it the 'main message'.

d) **Payload** (the actual data) – This is the meat!! This is what you want!!

HOW MAVLINK REALLY WORKS

MavLink is nothing but a message. Micro Aerial Vehicle Link (MavLink) as the name suggests, isn't quite a true name. It can be used with ground robots as well. It was named this way, because it started with copters (if my conjecture is correct).

The 'message' is a data bundle that contains a 'constant number of bytes' (i.e. 17, as already described). APM gets streaming bytes (from the air), gets it to the hardware interface (e.g. via UART serial OR Telemetry) and decodes the message in software. Note that the message contains the payload, which we would extract.

We are interested in the **payload**, but hey, along with the payload is the **message ID** (see above) to know what it represents. Before all this, these are a few steps on how the code interprets any MavLink message:

1) We have a method called handlemessage (msg). This is what you need to learn and know! Go and hunt for it in GCS_MavLink.pde (in Arducopter/ ArduPlane).

   It basically asks the packet: Hey, who are you packet? Are you meant for me or trying to hack into my system? Let me read the **System ID and Component ID** of yours first, before I allow you. Any system using MavLink has a System ID and Component ID. E.g. the MP and Quadcopter you are flying will have the same System ID to begin with. The Component ID is for a 'sub system' attached to the APM/PX4.

   *Note: Currently, the System ID and Component ID are **hardcoded** to be the same.*

So, you have one telemetry and one Copter with APM, that's it, go and fly happy – don't think of more! These will be helpful for swarm copters (in future). Different pairs of System ID's – That's the future ☺

2) The payload data is extracted from the message and put into a **packet**. A packet is a data structure based on a 'type' of information'. We no longer will use the term 'message'. That is over and done with. We are interested in packet, which is basically the packed 'raw data'.

3) The packet is put into an 'appropriate data structure'. There are plenty of data structures e.g. for Attitude (pitch, roll, yaw orientation), GPS, RC channels, etc. that groups similar things together, to be it modular, understandable. These data structures are **'perfectly 100% alike'** at the sending and receiving ends (i.e. at MP and APM side). If not, your copter would have come crashing at odd times!

   Also, this is where the MavLink GUI generator comes to the rescue. To generate these data structures, we do not need to code a single line!! Well, not quite, but only a little...

I hope it is easy so far, if not re-read it again!! It is OK, to be a dummy ☺

Ok, now for the real thingy. We send bi-directional messages with MavLink.

➔ From Ground station control (GCS) to APM/PX4 OR from APM/PX4 to Ground station control (GCS). Note, when I mean GCS, I mean it as Mission Planner(MP) or QGroundControl(QGC), DroidPlanner (DP) or your own custom tool you are working with to communicate with the copter.

GROUND STATION CONTROL(GCS) TO QUADCOPTER:

So far, we know that each message (we called a packet, which has the useful information for us), has a **message ID** and **payload** put into a data structure of 'appropriate type'. We switch on the 'main message' (or MAVLINK_MSG_ID_) type and once that message is detected, we do magic like storing the information received to permanent memory (called EEPROM), or whatever we wish to do with it.

As of Nov' 2013, in the latest 3.0.1 RC5 (release candidate) of Arducopter, these are the parameters you would find. I have tried to list the 'main message' ID for all possible MavLink message.

Note that within each *'main message'* category (like the ones in bold below), you would find *'sub messages'* which belong to that category which basically are closely related to the payload information (the real meat) and how it deals with it. Just like the 'Bike' category has Yamaha, Suzuki, Harley Davidson, etc. I am listing all main message categories but only specify some of the *sub category*. You can look up the details yourself ☺ because if you understand what I meant so far, you are no more a DUMMY. Get my point?

MAVLINK_MSG_ID (Block of main messages):

1) MAVLINK_MSG_ID_HEARTBEAT: //0
    a. This is the **most important message**. The GCS keeps sending a message to APM/PX4 to find out whether it is connected to it (every 1 second). This is to make sure the MP is in sync with APM when you update some parameters. If a number of heartbeats are missed, a failsafe (can be) is triggered and copter lands, continues the mission or Returns to launch (also called, RTL). The failsafe option can be enabled/ disabled in MP under Configure/Setup Failsafe options. But you can't stop heartbeats, can you? The name is very justified!!
2) MAVLINK_MSG_ID_REQUEST_DATA_STREAM: //66
    a. Sensors, RC channels, GPS position, status, Extra 1/2/3
3) MAVLINK_MSG_ID_COMMAND_LONG: // 76
    a. Loiter unlimited, RTL, Land, Mission start, Arm/Disarm, Reboot
4) SET_MODE: //11
    a. E.g. set_mode(packet.custom_mode);
5) MAVLINK_MSG_ID_MISSION_REQUEST_LIST: //43
    a. Total waypoints: **command_total** variable of parameters. This stores the total number of waypoints that are present (except home location, for multi-copters)
6) MAVLINK_MSG_ID_MISSION_REQUEST: //40
    a. Set of `MAV_CMD` value enum members, such as: (MAV_CMD_)CHANGE_ALT, SET_HOME, CONDITION_YAW, TAKE_OFF, NAV_LOITER_TIME
7) MAVLINK_MSG_ID_MISSION_ACK: //47
    a. // turn off waypoint send
8) MAVLINK_MSG_ID_PARAM_REQUEST_LIST: //21
    a. count_parameters (Count the total parameters)
9) MAVLINK_MSG_ID_PARAM_REQUEST_READ: //20
    a. Receive and decode parameters (Make sense of Param name and Id)

10) MAVLINK_MSG_ID_MISSION_CLEAR_ALL: //45

    a. When you use mission planner flight data screen and say, Clear Mission with the mouse menu, this is where it goes. It clears the EEPROM memory from the APM/PX4.

11) MAVLINK_MSG_ID_MISSION_SET_CURRENT: //41

    a. This is used to change active command during mid mission. E.g. when you click on MP google map screen and click 'Fly To Here', as an example.

12) MAVLINK_MSG_ID_MISSION_COUNT: // 44

    a. Save the total number of waypoints (excluding home location) -> for Multicopters.

13) MAVLINK_MSG_ID_MISSION_WRITE_PARTIAL_LIST: //

    a. Just keeping a global variable stating that APM is *receiving commands now*. This is to avoid other MavLink actions while important parameters are being set.

14) MAVLINK_MSG_ID_SET_MAG_OFFSETS: //151

    a. Set the **mag_ofs_x, mag_ofs_y, mag_ofs_z**, say after compass calibration to EEPROM of APM/PX4. Mission Planner (MP) automatically does this or you can do this too by going to Full Parameters List under SOFTWARE CONFIGURATION.

15) MAVLINK_MSG_ID_MISSION_ITEM: //39

    This is an interesting part. This message contains sub messages for taking real-time action. Like setting waypoints and advanced features... This is how it works, as below:

    a. Receive a Waypoint (WP) from GCS and store in EEPROM of APM/PX4.

    b. Sends 4 **params** (e.g. Delay, HitRad, -, and Yaw Angle) for LOITER_TIME (as **ID**) + (**Lat, Long, Alt**: Defines the 3D position of object in space). These parameters are defined as an Enum in code + Options (1 = Store Altitude (Alt) relative to home altitude). Each Command (or ID) may have different parameters of interest. Mission Planner shows 'blank' header for column, because there is no parameter defined for this ID.

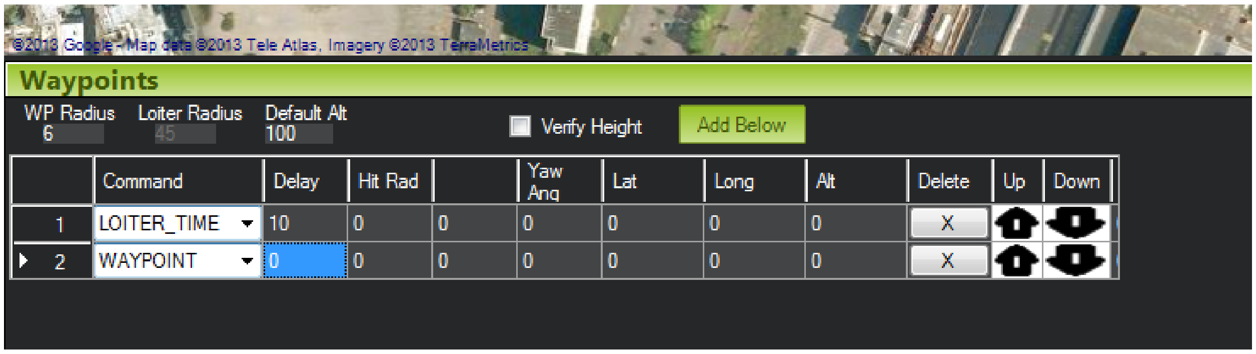
       As a summary, the interesting parameters sent with every action are:

       4 params + ID (of action) + (Lat, Long, Alt) defines 3D position of copter. Note the 4 params can be some sort of custom action as for camera setting, camera trigger, Loiter time, etc.

    c. As in figure below from Mission Planner, each ID defines a waypoint (AFAIK). LOITER_TIME, LOITER_UNLIMITED, WAYPOINT all are waypoints that send along with other parameters (LATITUDE, LONGITUDE and ALTITUDE) because each is saved as a waypoint in APM/PX4. Period.

d. Keep in mind: ALTITUDE is always relative to home altitude (Always!) with the current design.

e. You can define 'these actions' in Common.xml and use Python GUI generator to generate the code that APM/PX4 will use. Let me cover this later or ask me in forum on how to. You can add your own interesting parameters for the (4 params), I mentioned.


©2013 Google - Map data ©2013 Tele Atlas, Imagery ©2013 TerraMetrics
Waypoints
Verify Height
Add Below

WP Radius	Loiter Radius	Default Alt
6	45	100

	Command	Delay	Hit Rad		Yaw Ang	Lat	Long	Alt	Delete	Up	Down
1	LOITER_TIME	10	0	0	0	0	0	0	X		
2	WAYPOINT	0	0	0	0	0	0	0	X		

f. When APM receives this 'main' command (MAVLINK_MSG_ID_MISSION_ITEM), it reads the ID from the MavLink packet and performs switch (case) on the ID. E.g.

i. Loiter turns, Set home, Loiter time, Repeat servo, Set servo, etc.

16) MAVLINK_MSG_ID_PARAM_SET: //23

a. Set the parameter. Remember, we can set a value to a parameter in the Mission Planner (say, 'Full Parameter List'); this is where it goes when we do this. APM sends the data value set for confirmation. Have you seen MP, saying '**value set failed**'? Also, APM/PX4 logs simultaneously, this value, for our analyses offline.

17) MAVLINK_MSG_ID_RC_CHANNELS_OVERRIDE: //70

a. Override RC channel values for HIL (Hardware-In-Loop-Simulation) OR complete GCS control of the switch position via GUI (I have not tried this, though)!

18) MAVLINK_MSG_ID_HIL_STATE: //90

a. Used for HIL simulation. It is a virtual reality with your copter/ plane.

19) MAVLINK_MSG_ID_DIGICAM_CONFIGURE: //

20) MAVLINK_MSG_ID_MOUNT_CONFIGURE: //

21) MAVLINK_MSG_ID_MOUNT_CONTROL: //

22) MAVLINK_MSG_ID_MOUNT_STATUS://

   a. So far, as the name suggests, configures the appropriate command settings as set by the user.

23) MAVLINK_MSG_ID_RADIO, MAVLINK_MSG_ID_RADIO_STATUS: //

   a. Studies the packet rate of the telemetry/ USB and auto adjusts the delay between sending receiving packets if the signal strength is lower than expected or errors are getting higher. It is like an adaptive software flow control. Look at `__mavlink_radio_t` in C++ (APM code) OR `mavlink_radio_t` (C#, Mission Planner) code. Both are defined as structs.

QUADCOPTER TO GROUND STATION CONTROL (GCS) TO QUADCOPTER:

Ok, I admit. This gets more interesting. But this is far easier. The fact is that GCS is only mediator between you and your copter. It gets the data from copter in return, displays on the GCS.

If you go to Arducopter.pde file, look at this part of the code:

```
static const AP_Scheduler::Task scheduler_tasks[] PROGMEM = {
. . .
. . .
{ gcs_send_heartbeat,  100,     150 },
{ update_notify,         2,     100 },
{ one_hz_loop,         100,     420 },
{ gcs_check_input,       2,      550 },
{ gcs_send_heartbeat,  100,     150 },
{ gcs_send_deferred,     2,     720 },
{ gcs_data_stream_send,  2,     950 },
. . .
. . .
. . .
```

Don’t be scared. This is easier than even breathing. This is where **real-time systems** concept plays a role. We want certain tasks to take certain time, and if they are not finished by then, then we don’t proceed with them.

The first parameter is the function name,
The second is the ‘time it is supposed to take’ (in 10ms units, i.e., 2 means, 20ms, i.e. 50Hz, i.e. this function runs 50 times a second).
The third parameter is the ‘max time beyond which the function should not run’.

I think this is quite straightforward! Each function that you see there, its future is destined, it runs exactly for that much time. This is why it is safe to use Real-time systems for these nasty machines, make it predictable not unpredictable!!

All these functions have been handpicked for you to know that these are related to GCS updates. Simply, go to each of the function definition and these will invariably take you to GCS_Mavlink.pde where the real action of GCS communication occurs!!

The most interesting (and important) is this:

```
/*
 *  send data streams in the given rate range on both links
 */
static void gcs_data_stream_send(void)
{
    gcs0.data_stream_send();
    if (gcs3.initialised) {
        gcs3.data_stream_send();
    }
}
```

What this does is to send data down a link (gcs0 is via USB and gcs3 is via Telemetry). If you go down further/deeper, you would know that we send that the data structures back to GCS for display.
E.g. what happens when you move your copter with your hands and see Mission Planner’s HUD screen? You see copter moving on the screen. We are getting the Attitude data (Pitch, Roll and Yaw) every time unit. Likewise, we have IMU data, GPS data, Battery data, etc.

-- -- -- -- -- -- -- --

So, if you have any questions shoot me a message on forums and I will try to answer it with my knowledge. With the possibilities of these kinds of protocols, it makes a new beginning with no ending. Keeps growing with a new idea!!

-- -- -- -- -- -- -- --

Shyam Balasubramanian
(Embedded UAV Researcher and Developer),
Netherlands
~~Shyambs85@gmail.com~~