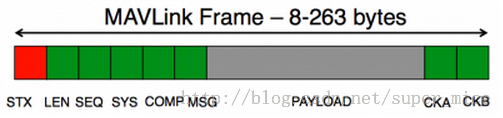
先简单介绍下mavlink协议。Mavlink协议最早由 苏黎世联邦理工学院 计算机视觉与几何实验组 的 [Lorenz Meier](http://www.inf.ethz.ch/personal/lomeier/)于2009年发布，并遵循LGPL开源协议。（数据链路层）Mavlink协议是在串口通讯基础上的一种更高层的开源通讯协议，主要应用在微型飞行器（micro aerial vehicle）的通讯上。Mavlink是为小型飞行器和地面站（或者其他飞行器）通讯时常常用到的那些数据制定一种发送和接收的规则并加入了校验（checksum）功能。

【1】下面开始说介绍mavlink所发送的数据结构。Mavlink传输时的基本单位是消息帧。



如图所示，每个消息帧都是上述的结构，除了灰色外，其他的格子都代表了一个字节的数据。

**红色的是起始标志位（stx），在v1.0版本中以“FE”作为起始标志。这个标志位在mavlink消息帧接收端进行消息解码时有用处。**

**第二个格子代表的是灰色部分（payload，称作有效载荷，要用的数据在有效载荷里面）的字节长度（len），范围从0到255之间。在mavlink消息帧接收端可以用它和实际收到的有效载荷的长度比较，以验证有效载荷的长度是否正确。**

**第三个格子代表的是本次消息帧的序号（seq），每次发完一个消息，这个字节的内容会加1，加到255后会从0重新开始。这个序号用于mavlink消息帧接收端计算消息丢失比例用的，相当于是信号强度。**

**第四个格子代表了发送本条消息帧的设备的系统编号（sys），使用PIXHAWK刷PX4固件时默认的系统编号为1，用于mavlink消息帧接收端识别是哪个设备发来的消息。**

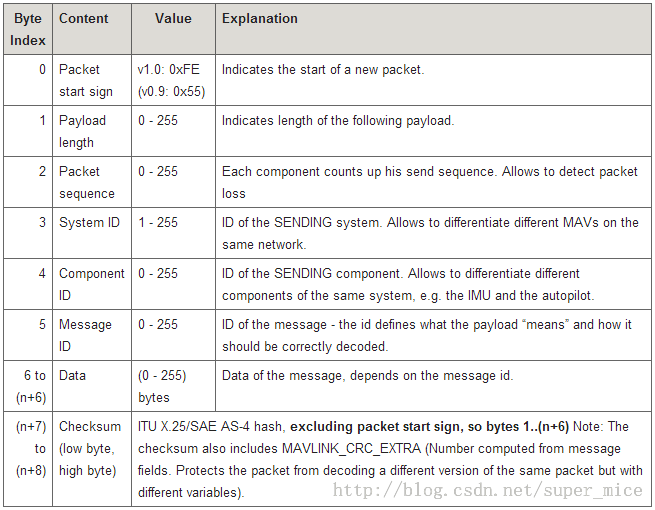
**第五个格子代表了发送本条消息帧的设备的单元编号（comp），使用PIXHAWK刷PX4固件时默认的单元编号为50，用于mavlink消息帧接收端识别是设备的哪个单元发来的消息（暂时没什么用） 。**

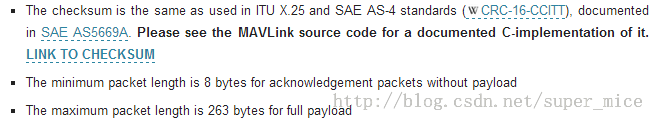
**第六个格子代表了有效载荷中消息包的编号（msg），注意它和序号是不同的，这个字节很重要，mavlink消息帧接收端要根据这个编号来确定有效载荷里到底放了什么消息包并根据编号选择对应的方式来处理有效载荷里的信息包。**

**最后两个字节是16位校验位，ckb是高八位，cka是低八位。校验码由crc16算法得到，算法将整个消息（从起始位开始到有效载荷结束，还要额外加上个**MAVLINK\_CRC\_EXTRA字节）进行crc16计算，得出一个16位的校验码。之前提到的每种有效载荷里信息包（由消息包编号来表明是哪种消息包）会对应一个MAVLINK\_CRC\_EXTRA，这个 MAVLINK\_CRC\_EXTRA 是由生成mavlink代码的xml文件生成的，加入这个额外的东西是为了当飞行器和地面站使用不同版本的mavlink协议时，双方计算得到的校验码会不同，这样不同版本间的mavlink协议就不会在一起正常工作，避免了由于不同版本间通讯时带来的重大潜在问题。

为了方便叙述，消息包将称作包，包所代表的信息称作消息。上图中的sys将称为sysid，comp将称为compid，msg将称为msgid。

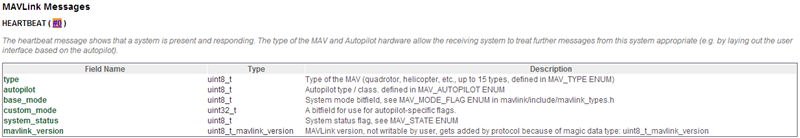
官方的介绍如下图：





上文中已经提到了在mavlink消息帧里最重要的两个东西，一个是msgid；一个是payload，前者是payload中内容的编号，后者则存放了消息。消息有许多种类型，在官网的网页中中以蓝色的“#”加数字的方式来表示消息的编号如 “#0”（这样的表示方法应该是为了方便在网页中查找相应编号消息的定义）。在官网介绍网页里往下拉，大概拉到二分之一的位置处，开始出现“MAVLink Messages”的介绍，往下看是各种消息的数据组成说明。下面将以几个消息为例，讲解mavlink消息。

先以 #0 消息为例，这个消息叫心跳包（heartbeat）。它一般用来表明发出该消息的设备是活跃的，飞行器和地面站都会发出这个信号（一般以1Hz发送），地面站和飞行器会根据是否及时收到了心跳包来判断是否和飞行器或地面站失去了联系。



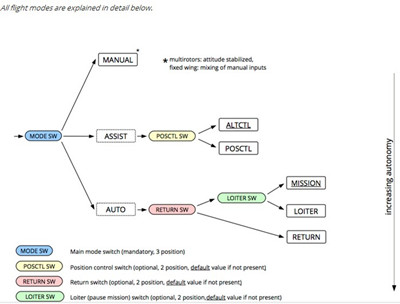
从图上可以看出，心跳包由6个数据组成，第一个是占一个字节的飞行器类型数据（type），这个数据表示了当前发消息的是什么飞行器，比如四旋翼，固定翼等等。type的取值如何与飞行器类型对应，这要在官方的mavlink消息介绍网页上找，位于网页开始出的数据枚举中。

这里只是一部分的类型，第一个是通用飞行器，对应的type数值是0；第二个是固定翼类型，对应的数值是1；第三个对应的是四旋翼，对应的数值是2.这个飞行器类型，其实对于发心跳包的地面站来说可能没什么意义（不同飞控对该消息的处理方法不同，至少刷了PX4固件的Pixhawk飞控对地面站发来的心跳包里的这个参数并不关心，如无特殊说明，之后所说的Pixhawk飞控都是指刷PX4固件的飞控），对于飞行器端来说代表了当前飞行器的类型，地面站可以根据这个参数来判断飞行器的类型并作出其他的反应。

第二个参数是自驾仪（即通常所说的飞控）类型，比如apm，ppz，Pixhawk等飞控，具体定义查找和之前查找飞行器类型时的方法一样。同样的，对于发送心跳包的飞行器来说代表了自己的飞控类性，对地面站发出的心跳包来说意义不大。

第三个参数是基本模式（base mode），是指飞控现在处在哪个基本模式，对于发心跳包的地面站来说没有意义，对于发送心跳包的飞控来说是有意义的。这个参数要看各个飞控自己的定义方式，mavlink介绍网页并不会给出具体的模式。在Pixhawk中基本模式可以分为使用用户模式（custom mode）还是基本模式（这里有点绕，其实是就是是否使用用户模式）。使用用户模式将在讲下个参数时说明，使用基本模式又会分为自动模式（auto），位置控制模式（posctl）和手动模式（manual）。一般情况下都会使用用户模式，普通用户不用关心这个参数。开发者在使用mavlink修改飞行器模式时需要注意基本模式的设置，具体请看PX4代码，下载地址<https://pixhawk.org/firmware/source_code>。

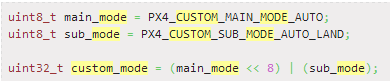
另外，Pixhawk的模式和apm的有很大的不同，具体请看官网介绍<https://pixhawk.org/users/system_modes>，里面还有关于遥控器如何设置模式的教程链接<https://pixhawk.org/users/system_modes/mode_switch_config>。用QGroundControl地面站（以后简称QGC，下载地址<http://qgroundcontrol.org/downloads>）的图形界面来设置飞行模式的功能很鸡肋，建议直接在QGC中读取飞控参数值，并对遥控器的设置参数进行修改，记得改变参数后只是改变了飞控ram参数，要把参数写入到rom中才可以。



第四个参数是用户模式（custom mode），大概说一下Pixhawk的用户模式。以多轴为例。它分为主模式（main mode）和子模式（sub mode），两种模式组合在一起成为最终的模式，主模式分为3种，手动（manual），辅助（assist），自动（auto）。手动模式类似apm的姿态模式。在辅助模式中，又分为高度控制模式（altctl）和位置控制模式（posctl）两个子模式，高度控制模式就类似apm的定高模式，油门对应到飞行器高度控制上。位置模式控制飞行器相对地面的速度，油门和高度控制模式一样，yaw轴控制和手动模式一样。自动模式里又分为3个子模式，任务模式（mission），留待模式（loiter），返航模式（return），任务模式就是执行设定好的航点任务，留待模式就是gps悬停模式，返航模式就是直线返回home点并自动降落。在apm里这个参数貌似是没有用的，注意这个数据占了4个字节，在Pixhawk中，前两个字节（低位）是保留的，没有用，第三个字节是主模式，第四个字节是子模式。普通用户请无视，开发者请注意：官网给出的通过程序设置模式的代码是错误的。如图，最后一行代码有误，应该为：

1. uint32\_t custom\_mode = (main\_mode<< 16) | (sub\_mode << 24);

[color=rgb(34, 34, 34) !important]复制代码



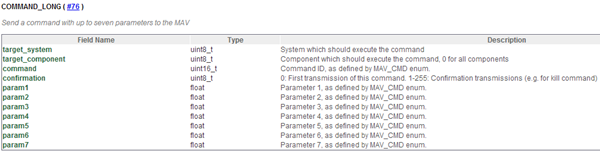
第五个是系统状态（system status），查定义就好了，其中的standby状态在Pixhawk里就是还没解锁的状态，active状态就是已经解锁，准备起飞的状态。

第六个是mavlink版本（mavlink version），现在是“3”版本。

其余的消息也是类似的结构，各个数据的定义可以查看mavlink官方网页的说明，这些说明一般在网页的前面部分。具体说明以飞控为准，mavlink仅提供基本的定义。

**有几个相对特殊和容易混淆的消息再特别说明下**：

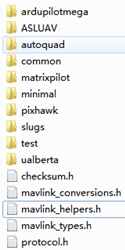
#76消息（command long），该消息是发送长命令，一般是地面站发送给飞控命令用的。该消息组成如下图。目标系统（命令的接收方，就是目标系统编号sysid），目标单元（命令的接收单元，就是目标单元编号compid）。command数据是这条命令的编号，用于区别不同的命令。confirmation数据，笔者还不是很明白，大概是是否需要收到命令后回复确认信号的意思。接下去有七个参数，这些参数是执行这条命令所需要告诉飞控的，许多命令都用不到七个参数，多余的参数清0就可以了。



Pixhawk支持的命令有许多种（但不是所有mavlink命令都支持）。要看mavlink提供了哪些命令请在介绍mavlink的官网查询mav\_cmd，在网页的中上部分。比如：第176号命令 MAV\_CMD\_DO\_SET\_MODE。这条命令用于改变飞行器的飞行模式，第一个参数就是设置飞控的base\_mode，第二个是设置custom\_mode。想要通过这条命令正确设置pixhawk的模式需要查看PX4代码，mavlink对参数的描述不够具体。

现在应该对介绍mavlink官网的布局有所了解了吧。网页前面主要讲了各类数据的取值和含义，比如飞控类型（mav\_autopilot）,飞行器类型（mav\_type）等，其中mav\_cmd是比较特殊和重要的一种数据。网页的后半部分主要讲了mavlink消息的种类和数据组成，这里会用到各种数据，具体数据定义的可以回到前半部分去找。但是mavlink是个通用的通讯协议，不同的飞控支对mavlink支持方式不一样，一般都只支持一部分mavlink消息，还会自己扩展一些mavlink协议所没有定义的消息（pixhawk和apm都是如此），具体都以飞控代码为准。

大概说说地面站和飞控的通讯流程，由于没看过地面站的代码，所以很可能有误，还望发评论指正！一般飞控在连接上地面站后都会主动向地面站发送心跳包，飞行器姿态，系统状态，遥控器信号等组成的数据流。各个数据都会以一定的频率发送，比如心跳包一般是1Hz，姿态信息会快些，pixhawk用数传连接QGC时的姿态数据发送频率在7-8Hz左右。一般地面站会在刚连接上飞控时发送命令，请求飞控传回所有参数（QGC就是这样），飞控根据自己的情况判断是否接受地面站的请求，并根据不同的命令执行相应的操作（有些命令需要飞控回复地面站确认信号）。之后地面站根据用户的操作会发送相应的mavlink消息给飞控，比如设置航点，改写飞控参数等。据说数传是半双工的（在同一时刻只能选择发送或者选择接受数据，不能同时收发数据），地面站和飞控之间如何避免数据冲突（即双方同时向对方发送消息）的机制笔者并不清楚，希望能抛砖引玉。



可以看到，里面有多个文件夹和几个头文件。pixhawk，ardupilotmega（apm），matrixpilot这类的文件夹里都是各个飞控自己定义的mavlink消息类型，原始的mavlink消息放在common文件夹里面（大部分消息都在common文件夹中）。checksum.h中存放的是计算校验码的代码。 mavlink\_helper.h里面是将各个消息包补充完整（调用checksum.h中的函数计算校验码并补上消息帧的头，比如sysid和compid等）成为mavlink消息帧再发送。最主要的功能集中在这两个文件夹中。mavlink\_conversions.h里是dcm，欧拉角，四元数三种姿态表示方法之间的转换代码。

**下面以发送心跳包（heartbeat）为例，说明下如何使用mavlink头文件来发送心跳包**。首先打开common文件夹中的 mavlink\_msg\_heartbeat.h 头文件。这个头文件可以分为两部分，一部分用来打包、发送heartbeat消息，另一部分用来接收到heartbeat消息时解码消息。heartbeat.h定义了heartbeat消息对应的数据类型：

1. typedef struct \_\_mavlink\_heartbeat\_t
2. {
3. uint32\_t custom\_mode; ///< A bitfield for use for autopilot-specific flags.
4. uint8\_t type; ///< Type of the MAV (quadrotor, helicopter, etc., up to 15 types, defined in MAV\_TYPE ENUM)
5. uint8\_t autopilot; ///< Autopilot type / class. defined in MAV\_AUTOPILOT ENUM
6. uint8\_t base\_mode; ///< System mode bitfield, see MAV\_MODE\_FLAG ENUM in mavlink/include/mavlink\_types.h
7. uint8\_t system\_status; ///< System status flag, see MAV\_STATE ENUM
8. uint8\_t mavlink\_version; ///< MAVLink version, not writable by user, gets added by protocol because of magic data type: uint8\_t\_mavlink\_version
9. } mavlink\_heartbeat\_t;

[color=rgb(34, 34, 34) !important]复制代码

如果mavlink的发送方式可以使用（串口发送，函数接口也兼容），则可以调用 【1】

1. static inline void mavlink\_msg\_heartbeat\_send(mavlink\_channel\_t chan, uint8\_t type, uint8\_t autopilot, uint8\_t base\_mode, uint32\_t custom\_mode, uint8\_t system\_status)

[color=rgb(34, 34, 34) !important]复制代码

其中的chan是channel的缩写，用于选择发送的串口或者usb口。type就是飞行器类型，其余参数不明的可以看看本博客的第一篇文章。

该函数功能是将传入的各个参数按照对应的格式放到heartbeat消息包中（即打包）

这个函数内部有一句预处理：

1. #if MAVLINK\_CRC\_EXTRA

[color=rgb(34, 34, 34) !important]复制代码

是说是否使用额外的crc校验字符（默认使用），详情请看第一篇博客中对于两个校验字节的说明。

函数中会调用函数【2】

1. \_mav\_finalize\_message\_chan\_send(chan, MAVLINK\_MSG\_ID\_HEARTBEAT, buf, MAVLINK\_MSG\_ID\_HEARTBEAT\_LEN, MAVLINK\_MSG\_ID\_HEARTBEAT\_CRC);  
   [color=rgb(34, 34, 34) !important]
2. MAVLINK\_MSG\_ID\_HEARTBEAT//这个是心跳包消息对应的编号 这里=0  
   [color=rgb(34, 34, 34) !important]复制代码
3. MAVLINK\_MSG\_ID\_HEARTBEAT\_LEN//这个是心跳包的长度 注意这个长度仅仅是payload的长度，不包括帧的头尾。

[color=rgb(34, 34, 34) !important]

1. MAVLINK\_MSG\_ID\_HEARTBEAT\_CRC//这个是heartbeat消息对应的额外的crc校验码 这里=50

[color=rgb(34, 34, 34) !important]

这个函数位于mavlink\_helper.h中，用于更新消息帧的编号（seq 每发送一帧加1）并将消息帧的头和计算校验码，使得成为完整的一个mavlink消息帧。最后调用串口发送函数进行消息帧的发送。

如果只是想将对应的心跳包参数按照心跳包的格式存放好，则可以只调用

1. static inline uint16\_t mavlink\_msg\_heartbeat\_pack(uint8\_t system\_id, uint8\_t component\_id, mavlink\_message\_t\* msg,
2. uint8\_t type, uint8\_t autopilot, uint8\_t base\_mode, uint32\_t custom\_mode, uint8\_t system\_status)

[color=rgb(34, 34, 34) !important]复制代码

将参数打包为heartbeat消息帧，待之后使用。

解码消息帧时可以调用mavlink\_helper.h中的

1. MAVLINK\_HELPER uint8\_t mavlink\_parse\_char(uint8\_t chan, uint8\_t c, mavlink\_message\_t\* r\_message, mavlink\_status\_t\* r\_mavlink\_status)

[color=rgb(34, 34, 34) !important]复制代码

它会将收到的字符一个个进行解码，会检验收到的校验码是否正确；有效载荷的长度小于最大长度并且和该消息的长度一致。如果一切顺利，将会得到解码到的消息，放在解码得到的消息帧类型中

1. typedef struct \_\_mavlink\_message {
2. uint16\_t checksum; ///< sent at end of packet
3. uint8\_t magic; ///< protocol magic marker
4. uint8\_t len; ///< Length of payload
5. uint8\_t seq; ///< Sequence of packet
6. uint8\_t sysid; ///< ID of message sender system/aircraft
7. uint8\_t compid; ///< ID of the message sender component
8. uint8\_t msgid; ///< ID of message in payload
9. uint64\_t payload64[(MAVLINK\_MAX\_PAYLOAD\_LEN+MAVLINK\_NUM\_CHECKSUM\_BYTES+7)/8];
10. } mavlink\_message\_t;

[color=rgb(34, 34, 34) !important]复制代码

其中的magic是一帧的起始标志（FE=254），就是mavlink\_stx的值。

其余的mavlink消息也是类似的，旧的mavlink代码中有些类型的消息类型可能会找不到，使用时要注意接受和发送方使用的mavlink版本是否兼容。common文件夹中的common.h里面包含了要用到的数据类型和所有消息的头文件，使用时直接包含进来即可。