8、logger，参数同步；

9、空闲。

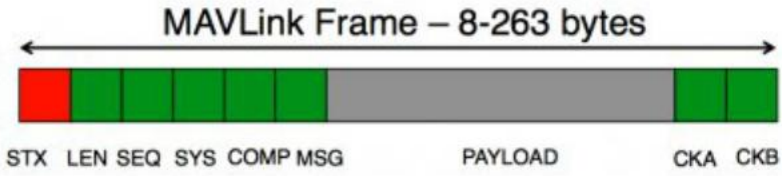
Mavlink

参考链接：http://qgroundcontrol.org/mavlink/start

MAVLink 是一个轻量级的、只由头文件组成的信息集编库，常被用在微型飞行器上。它能将 C 语言结构的串行数据高效低打包并发送给地面控制软件。它已经被大量使用在如下系统中：PX4、Pixhawk、APM 和 Parrot AR.Drone 等几种飞控中，同时，也用在微控制器（MCU）和惯性测量装置（IMU）间的通信，此外，也用在 Linux 进程与地面链路间的通信。

帧结构：

Mavlink 的帧结构参考了 CAN 和 SAE AS-4 标准，其帧结构如下：



字节序号	内容	值	备注
0	帧头	V1.0：0xFE V0.9：0x55	指示帧的起始(一位足够，官网有解释)
1	帧长	0- 255	表示帧数据长度
2	帧序号	0 - 255	每发一帧，自动增一，溢出归零。用来判断是否丢帧。
3	系统 ID (System ID)	1 - 255	发送方系统 ID。允许同一网络中有不同的系统。

4	组件 ID (Component ID)	0 - 255	发送组件 ID。允许同一系统中有多个组件，如 IMU 和飞控。
5	信息 ID (Message ID)	0 -255	信息 ID：指示帧的定义
6 to (n+6)	数据（Data）	(0 - 255)bytes	数据，根据 Message ID 进行编码
(n+7) to (n+8)	Checksum(低字节、高字节)	ITU X.25/SAE AS-4 hash，不包含帧头，即 1~（n+6）	

- 最短的帧长度为 8 个字节（命令状态返回帧）；
- 最长的帧长度为 263 个字节。

支持的数据类型

Mavlink 支持固定大小整形数据，IEEE 754 单精度浮点数据，这些数据组成的数组（如：char[10]），已经特殊的 mavlink_version 域，被协议自动添加。如下类型是被支持的：

- **char** - Characters / strings
- **uint8_t** - Unsigned 8 bit
- **int8_t** - Signed 8 bit
- **uint16_t** - Unsigned 16 bit
- **int16_t** - Signed 16 bit
- **uint32_t** - Unsigned 32 bit
- **int32_t** - Signed 32 bit
- **uint64_t** - Unsigned 64 bit
- **int64_t** - Signed 64 bit
- **float** - IEEE 754 single precision floating point number
- **double** - IEEE 754 double precision floating point number

- **uint8_t_mavlink_version** –8 位无符号数，被自动添加，用于指示 Mavlink 版本；无法被修改，只能被作为一个普通的 uint8_t 数被读出。

性能

这个协议注重两个特性：传输速度和安全性。它允许检测信息内容，也允许检测丢失的信息，但依然只需要在每帧数据前加上 6 个字节的头部（即帧头、各种 ID 等等）。

发送示例：

数据链速率	硬件	更新速率	有效帧长	等效 float 数个数
115200	XBee Pro 2.4 GHz	50 Hz	224 bytes	56
115200	XBee Pro 2.4 GHz	100 Hz	109 bytes	27
57600	XBee Pro 2.4 GHz	100 Hz	51 bytes	12
9600	XBee Pro XSC 900	50 Hz	13 bytes	3
9600	XBee Pro XSC 900	20 Hz	42 bytes	10

Mavlink 板级集成教程（MAVLink Onboard Integration Tutorial）

参考链接：

http://qgroundcontrol.org/dev/mavlink_onboard_integration_tutorial

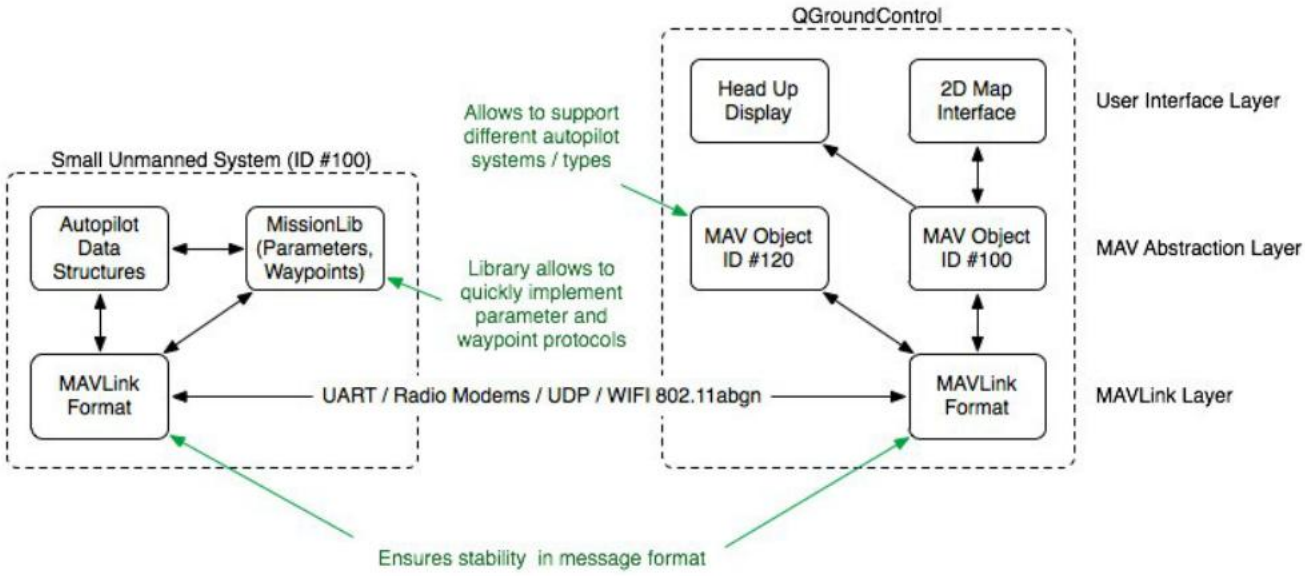
Mavlink 是一个只由头文件组成的库，这意味着你在 MCU 编程时不需要编译它。你只需要将 Mavlink/include 添加到程序列表中即可。（注意：你可以创建自定义的信息帧，然后使用 Mavlink 官方提供的工具转换得到 C 语言程序）

Mavlink 是 stateless 的，但是地面站软件 QGroundControl 会通过心跳信息（heartbeat message）来确定系统是否处于活跃状态。因此需要确保每个 60、30、10 或者 1 秒（1Hz 是推荐值，但是非必须）发送一次心跳信息，只有这样这个系

统才会被认为是活跃的。

集成结构

如下图所示，集成 Mavlink 是非入侵式的。它不需要成为板子架构的核心部分。Mavlink 提供的 missionlib 掌管参数和任务以及路径点的传输，飞控只需要从特性的结构体中读出其值即可。



快速集成：发送数据

下面的示例方法会需要很多行（与实际应用相比），但是能够帮助你快速入门。

```
    /* The default UART header for your MCU */
#include "uart.h"
#include <mavlink/v1.0/common/mavlink.h>

mavlink_system_t mavlink_system;

mavlink_system.sysid=20;///< ID 20 for this airplane
mavlink_system.compid= MAV_COMP_ID_IMU;///< The component sending the message is the IMU, it could be also a Linux process
mavlink_system.type= MAV_TYPE_FIXED_WING;///< This system is an airplane / fixed wing

// Define the system type, in this case an airplane
uint8_t system_type = MAV_TYPE_FIXED_WING;
```

```
uint8_t autopilot_type = MAV_AUTOPILOT_GENERIC;

uint8_t system_mode = MAV_MODE_PREFLIGHT;///< Booting up
uint32_t custom_mode =0;///< Custom mode, can be defined by user/adopter
uint8_t system_state = MAV_STATE_STANDBY;///< System ready for flight

// Initialize the required buffers
mavlink_message_t msg;
uint8_t buf[MAVLINK_MAX_PACKET_LEN];

// Pack the message
mavlink_msg_heartbeat_pack(mavlink_system.sysid, mavlink_system.compid,&msg,
system_type, autopilot_type, system_mode, custom_mode, system_state);

// Copy the message to the send buffer
uint16_t len = mavlink_msg_to_send_buffer(buf,&msg);

// Send the message with the standard UART send function
// uart0_send might be named differently depending on
// the individual microcontroller / library in use.
    uart0_send(buf, len);
```

接收数据

上述例子讲述了如何使用不太方便的函数发送数据（这种函数需要更多的行）。下面的程序讲述如何接收数据。这里为了示例，直接把接收放在主循环中了，这样会尽可能快地清空串口接收缓存。

```
#include <mavlink/v1.0/common/mavlink.h>
// Example variable, by declaring them static they're persistent
// and will thus track the system state
staticint packet_drops =0;
staticint mode = MAV_MODE_UNINIT;/* Defined in mavlink_types.h, which is included by
mavlink.h */
/**
* @brief Receive communication packets and handle them
*
* This function decodes packets on the protocol level and also handles
* their value by calling the appropriate functions.
```

```
*/
staticvoid communication_receive(void)
{
	mavlink_message_t msg;
	mavlink_status_t status;

	// COMMUNICATION THROUGH EXTERNAL UART PORT (XBee serial)

	while(uart0_char_available())
	{
		uint8_t c = uart0_get_char();
		// Try to get a new message
		if(mavlink_parse_char(MAVLINK_COMM_0, c,&msg,&status)){
			// Handle message

			switch(msg.msgid)
			{
			case MAVLINK_MSG_ID_HEARTBEAT:
			{
				// E.g. read GCS heartbeat and go into
// comm lost mode if timer times out
			}
			break;
			case MAVLINK_MSG_ID_COMMAND_LONG:
				// EXECUTE ACTION
				break;
			default:
				//Do nothing
				break;
			}
		}

		// And get the next one
	}

	// Update global packet drops counter
	packet_drops += status.packet_rx_drop_count;

	// COMMUNICATION THROUGH SECOND UART
```

```
        while(uart1_char_available())
        {
                uint8_t c = uart1_get_char();
                // Try to get a new message
                if(mavlink_parse_char(MAVLINK_COMM_1, c,&msg,&status))
                {
                        // Handle message the same way like in for UART0
// you can also consider to write a handle function like
// handle_mavlink(mavlink_channel_t chan, mavlink_message_t* msg)
// Which handles the messages for both or more UARTS
                }

                // And get the next one
        }

        // Update global packet drops counter
        packet_drops += status.packet_rx_drop_count;
    }
```

使用便捷的函数

Mavlink 提供了方便发送帧的便捷函数。如果你想使用这些函数，你需要在你的代码中添加一些特殊的函数、这种方法在发送数据时需要更少的代码，但是需要你自己去写适配头文件。下面的例子中将会讲到如何使用这些函数（注意：我们建议初学者从非便捷函数用起，能成功发送消息到地面站软件后，如果需要，再去添加适配头文件）。

```
#include "your_mavlink_bridge_header.h"
/* You have to #define MAVLINK_USE_CONVENIENCE_FUNCTIONS in your_mavlink_bridge_header,
   and you have to declare: mavlink_system_t mavlink_system;
   these two variables will be used internally by the mavlink_msg_xx_send() functions.
   Please see the section below for an example of such a bridge header. */
#include <mavlink.h>

// Define the system type, in this case an airplane
int system_type = MAV_FIXED_WING;
// Send a heartbeat over UART0 including the system type
```