

# Pixhawk 源码分析 (WalkAnt 版)

By @WalkAnt

30175224@qq.com

2016/3/16 Monday

第一次编辑时间：2014 年 12 月

第二次整理时间：2016 年 3 月 16 日

源代码: <https://github.com/diydrones/ardupilot>

英文资料: <http://dev.ardupilot.com/>

<http://liung.github.io/blog/apm/2014-08-29-APM-开发人员参考手册目录列表.html>

## 1 概述

### 支持的硬件版

当前 ArduPilot/APM 支持的主板包括：

- [Pixhawk](#)：下一代 PX4
- [PX4 FMU](#)：32 位 ARM，基于 [NuttX](#) 实时操作系统
- [VRBrain](#)
- [FlyMaple](#)
- BeagleBone Black：正在开发，基于 Linux 系统。
- [APM2](#)：AVR2560 8 位 CPU。

ArduPilot/APM 源代码基于 [AP-HAL](#) 硬件抽象层，便于移植到更多不同种类的板子上，详细了解见：[blog post](#)。

### 所有开源项目列表

- APM:Plane ([wiki](#), [code](#)) – autopilot for planes，固定翼
- APM:Copter ([wiki](#), [code](#)) – autopilot for multicopters and traditional helicopters，多旋翼
- APM:Rover ([wiki](#), [code](#)) – autopilot for ground vehicles，无人车
- Mission Planner ([wiki](#), [code](#)) – the most commonly used ground station written in C# for windows but also runs on Linux and MacOS via mono，地面站 MissionPlanner
- APM Planner 2.0 ([wiki](#), [code](#)) is a ground station specifically for APM written in C++ using the Qt libraries，地面站，APM Planner
- MAVProxy ([wiki](#)) – command line oriented and scriptable ground station (mostly used by developers)，
- MinimOSD ([wiki](#), [code](#)) – on-screen display of flight data，
- AndroPilot ([user guide](#), [code](#), [google play](#)) – android ground station，地面站，安卓版，AndroPilot
- DroneAPI ([tutorial](#), [droneshare](#)) – A developer API for drone coprocessors and web applications.
- DroidPlanner ([wiki](#), [code](#), [google play](#)) – android ground station，地面站，安卓版，DroidPlanner
- [QGroundControl](#) is an alternative ground station written in C++ using the Qt libraries，地面站，QGroundControl
- PX4 ([wiki](#)) – designers of the PX4FMU and owners of the underlying libraries upon which APM:Plane/Copter/Rover use when running on the PX4FMU，
- MAVLink ([wiki](#)) – the protocol for communication between the ground station, flight controller and some peripherals including the OSD. A “Dummy’s Guide” to working with MAVLink is [here](#). MAVLink 通讯协议

## 入门参考目录

- [Where to get the code](#) , 如何获取代码
- [Learning the ArduPilot codebase](#) 基础知识
- [Understanding the ArduCopter code](#) , 源代码预览
  - ◇ [Library description](#)
  - ◇ [Attitude Control](#)
  - ◇ [How to add Parameters](#)
  - ◇ [How to create a new flight mode](#)
  - ◇ [How to run your new code intermittently](#)
  - ◇ [How to add a new MAVLink message](#)
- [Building the code](#) 编译代码
- Loading the code onto the board
  - ◇ [Load the code onto APM2.x](#)
  - ◇ [Load the code onto a BeagleBone Black](#)
- [Editing the code with NotePad++, Eclipse or MS Visual Studio](#)
- [Git and GitHub guide](#)
- [Testing](#)
  - ◇ [Setting up SITL on Windows](#)
  - ◇ [Setting up SITL on Linux](#)
  - ◇ [Using Replay to rerun the flight with the latest code](#)
  - ◇ [Hardware in the Loop with X-Plane \(Arduplane only\)](#)
- [Debugging with GDB](#)
- [Submitting Patches back to master](#)
- [EZ Developer Walk Throughs](#) Install and set up a development system, get the ArduPilot code and compile and load it to your controller.
  - ◇ [EZ Install the Ardupilot Source Code on Your Computer Using Zip](#)
  - ◇ [EZ Get Arduino and Initialize it to Work With Ardupilot](#)
  - ◇ [EZ Arduino Compile and Upload the Ardupilot Firmware to Your APM](#)
- [Communicating with Raspberry Pi or ODroid U3 via Mavlink](#)

## 2 如何获取代码

英文：<http://dev.ardupilot.com/wiki/where-to-get-the-code/>

中文：<http://liung.github.io/blog/apm/2014-08-29-APM-获取源码.html>

整个 APM 工程的源码都采用 [git](#) (译者注：一款自由和开源的分布式版本控制系统) 进行代码管理，并且托管在 [github](#) 网站上。目前 APM 的源码以开源的形式托管在 <https://github.com/diydrones/ardupilot> 上。由于历史原因，另一个老版本的 Google code 仓库仍然保持在线可访问状态，但是除非你对老版本 (APM 1.x) 有特殊的需求，一般建议您不要使用该仓库中的代码。

固定翼飞机 (Plane)，旋翼飞行器 (Copter)，无人车 (Rover) 模块全部包含在 [diydrones/ardupilot](#) 代码库中，天线跟踪模块虽

然也包含在该代码库中，但在文件目录 `Tools/` 下。

地面站 MissionPlanner 包含在 [diydrone/MissionPlanner](#) 代码库中。

## 2.1 预备知识

Git 在绝大多数操作平台上都是可以使用的，并且还存在各种工具使得开始使用 Git 更加简单。首先，你需要[下载和安装对应你使用的操作平台的客户端程序](#)，如果你刚开始接触到这种源代码控制系统，那么，[windows 版的 GitHub](#) 或 [Mac 版的 GitHub](#) 客户端都有一个非常好的使用文档，并且和 GitHub 有着很好的集成，可以从这里开始学习如何使用 Git。本次说明将同时使用 Windows 版的 Github 用户界面和 OSX/Linux 终端下的命令行界面进行操作。

如果你想要直接向 APM 官网代码库中提交代码，那么你需要[在 Github 上免费注册一个账号](#)。

如何安装 git，请进入以下链接：

<http://liung.github.io/blog/apm/2014-08-29-APM-获取源码.html>

## 2 基础知识

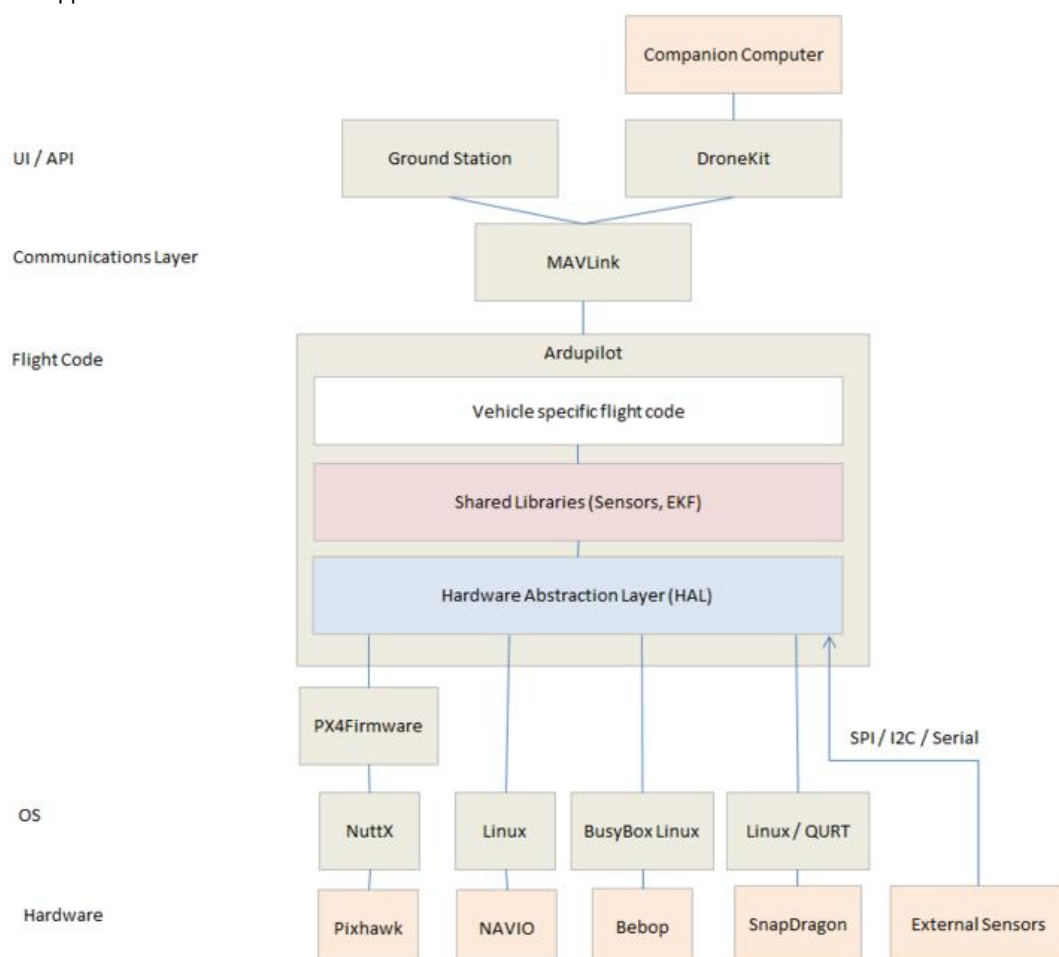
详细参考：<http://dev.ardupilot.com/wiki/learning-the-ardupilot-codebase/>

### 2.1 介绍

详细参考：<http://dev.ardupilot.com/wiki/learning-ardupilot-introduction/>

ArduPilot 代码分为 5 个主要部分，基本结构分类如下：

- vehicle directories
- AP\_HAL
- libraries
- tools directories
- external support code



## 1、vehicle directories 模型类型

当前共有 4 种模型：ArduPlane, ArduCopter, APMrover2 and AntennaTracker。都是.pde 文件，就是为了兼容 arduino 平台，以后可能会放弃。

## 2、AP\_HAL 硬件抽象层

硬件抽象层，使得在不同硬件平台上的移植变得简单。

其中 AP\_HAL 目录定义了一个通用的接口。其他的目录 AP\_HAL\_XXX 针对不同硬件平台进行详细的定义。例如 AP\_HAL\_AVR 目录对于 AVR 平台，**AP\_HAL\_PX4 对应 PX4 平台**，AP\_HAL\_Linux 对应 Linux 平台。

## 3、tools directories 工具目录

主要提供支持。For examples, tools/autotest provides the autotest infrastructure behind the [autotest.diydrones.com](http://autotest.diydrones.com) site and tools/Replay provides our log replay utility。主要用于测试。

## 4、external support code 外部支持代码

对于其他平台，需要外部支持代码。例如 Pixhawk、PX4 的支持代码如下：

- [PX4NuttX](#) – 板载实时系统。the core NuttX RTOS used on PX4 boards
- [PX4Firmware](#) – PX4 固件。the base PX4 middleware and drivers used on PX4 boards
- [uavcan](#) – 飞行器 CAN 通信协议。the uavcan CANBUS implementation used in ArduPilot
- [mavlink](#) – Mavlink 通信协议。the mavlink protocol and code generator

## 5、系统编译

针对不同的硬件板，编译可以采用“make TARGET”的形式。

- make apm1 // the APM1 board
- make apm2 // the APM2 board
- make px4-v1 // the PX4v1
- **make px4-v2 // the Pixhawk**

如果要移植到新的硬件，可以在 [mk/targets.mk](#) 文件中添加。

比如：**make apm2-octa-j8**

采用 8 通道并行编译方式，针对 APM 硬件板（AVR），编译八旋翼代码。

编译环境为 px4toolchain，请自行下载。

关于编译环境搭建，参考以下链接：

官方 1：[http://www.pixhawk.com/dev/toolchain\\_installation\\_win](http://www.pixhawk.com/dev/toolchain_installation_win)

官方 2：<http://dev.px4.io/starting-installing-windows.html>

网友文章：

[http://wenku.baidu.com/link?url=67s7oM4YZoPFTTrRYcnEDpdadyVCscYEQ6W8\\_rYW5vBoo0caS8ipEEDggnjEcHkZXmLEVvwDt4h4Jw9h9KO\\_Fcn9nNeQS52jL2DpOgx\\_pACMO](http://wenku.baidu.com/link?url=67s7oM4YZoPFTTrRYcnEDpdadyVCscYEQ6W8_rYW5vBoo0caS8ipEEDggnjEcHkZXmLEVvwDt4h4Jw9h9KO_Fcn9nNeQS52jL2DpOgx_pACMO)

## 2.2 学习 sketch 例程代码

详细参考：<http://dev.ardupilot.com/wiki/learning-ardupilot-the-example-sketches/>

sketch，是指使用 .pde 文件编写的主程序。

开始之前，你可以试着阅读、编译并运行下面的 sketches

- libraries/AP\_GPS/examples/GPS\_AUTO\_test
- libraries/AP\_InertialSensor/examples/INS\_generic
- libraries/AP\_Compass/examples/AP\_Compass\_test

- libraries/AP\_Baro/examples/BARO\_generic
- libraries/AP\_AHRS/examples/AHRS\_Test

例如，下面的编译方法，将在 Pixhawk 上安装 AP\_GPS 例程 sketch。

```
cd libraries/AP_GPS/examples/GPS_AUTO_test
make px4-clean
make px4-v2
make px4-v2-upload
```

正确理解 sketch 例程代码，我们以 GPS\_AUTO\_test.pde 代码为例（目录 ardupilot\libraries\AP\_GPS\examples\GPS\_AUTO\_test），主要几个特点：

- 1、pde 文件包含很多 includes；
- 2、定义了 hal 引用声明；
- 3、代码非常简单；用来测试 GPS 数据。通过串口输出 GPS 经纬度，状态，时间等信息。
- 4、setup() 和 loop()函数

代码基本结构：

```
#include <stdlib.h>
#include <AP_Common.h>
#include <AP_Progmem.h>
...
#include <AP_NavEKF.h>

const AP_HAL::HAL& hal = AP_HAL_BOARD_DRIVER;

void setup()
{
    hal.console->println("GPS AUTO library test");
}

void loop()
{
    static uint32_t last_msg_ms;
    gps.update();
    if (last_msg_ms != gps.last_message_time_ms()) {
        last_msg_ms = gps.last_message_time_ms();
        const Location &loc = gps.location();// 获取 GPS 位置数据
        hal.console->print("Lat: ");// 输出纬度
        print_latlon(hal.console, loc.lat);
        hal.console->print(" Lon: ");//输出经度
        print_latlon(hal.console, loc.lng);
        hal.console->printf(" Alt: %.2fm GSP: %.2fm/s CoG: %d SAT: %d TIM: %u/%lu STATUS: %u\n",
                           loc.alt * 0.01f,
                           gps.ground_speed(),
                           (int)gps.ground_course_cd() / 100,
                           gps.num_sats(),
                           gps.time_week(),
                           gps.time_week_ms(),
                           gps.status());
    }
    hal.scheduler->delay(10);
}
```

```
}  
  
AP_HAL_MAIN();
```

## 1、include 文件

pde 文件转变为 C++ 文件后，提供必要的库引用支持。

## 2、hal 引用声明

定义如下：

```
const AP_HAL::HAL& hal = AP_HAL_BOARD_DRIVER;// pixhawk 的定义为 AP_HAL_PX4
```

该定义，方便访问飞控器硬件接口，比如 console 终端、定时器、I2C、SPI 接口等。

实际的定义是在 HAL\_PX4\_Class.cpp 中定义，如下：

```
const HAL_PX4 AP_HAL_PX4;
```

hal 是针对 AP\_HAL\_PX4 的引用。

经常使用的方法如下：

- 终端字符输出。 [hal.console->printf\(\)](#) and [hal.console->printf\\_P\(\)](#) to print strings (use the \_P to use less memory on AVR)
- 获取当前运行时间。 [hal.scheduler->millis\(\)](#) and [hal.scheduler->micros\(\)](#) to get the time since boot
- 延时。 [hal.scheduler->delay\(\)](#) and [hal.scheduler->delay\\_microseconds\(\)](#) to sleep for a short time
- IO 输入输出。 [hal.gpio->pinMode\(\)](#), [hal.gpio->read\(\)](#) and [hal.gpio->write\(\)](#) for accessing GPIO pins
- I2C 操作， [hal.i2c](#)
- SPI 操作， [hal.spi](#)

## 3、setup()和 loop()

每个 sketch 都有一个 setup()和 loop()函数。板子启动时，setup()被调用。这些调用都来自 HAL 代码中的 main()函数调用（HAL\_PX4\_Class.cpp 文件 main\_loop()）。setup()函数只调用一次，用于初始化所有 libraries。

Loop()循环被调用，执行主任务。

## 4、AP\_HAL\_MAIN()宏指令

每一个 sketch(.pde 文件)最底部，都有一个“AP\_HAL\_MAIN();”指令，它是一个 HAL 宏，用于定义一个 C++ main 函数，整个程序的入口。它真正的定义在 AP\_HAL\_PX4\_Main.h 中。

```
#define AP_HAL_MAIN() \  
    extern "C" __EXPORT int SKETCH_MAIN(int argc, char * const argv[]); \  
    int SKETCH_MAIN(int argc, char * const argv[]) {    \  
        hal.init(argc, argv); \  
        return OK; \  
    }
```

作为程序的起点，在 AP\_HAL\_MAIN()里，就正式调用了 hal.init()初始化代码。

程序的执行过程就是：[程序起点 AP\\_HAL\\_MAIN\(\)](#) → [hal.init\(\)](#) → [hal.main\\_loop\(\)](#) → sketch 中的 [setup\(\)](#)和 [loop\(\)](#)。

## 2.3 APM 线程

详细参考：<http://dev.ardupilot.com/wiki/learning-ardupilot-threading/>

对于 APM1、APM2 硬件板，不支持多线程，所以只能通过简单的定时器加回调函数来实现。类似 PX4 和 Linux 硬件板支持 Posix 标准的多线程。线程一般是指基于多任务操作系统的并行任务，我们首先要明白的几个概念如下：

- 1、定时回调
- 2、HAL 专属线程
- 3、驱动专属线程

- 4、 APM 驱动与板级驱动
- 5、 板级专属线程、任务
- 6、 AP\_Scheduler 任务调度系统
- 7、 信号灯（任务队列互锁用）
- 8、 lockless data structures

## 1、定时回调 The timer callbacks

每个飞控平台都提供一个 1kHz 的定时器（见 AP\_HAL），通过“注册”一个定时器函数来获取 1kHz 定时功能。所有注册的定时器将被顺序调用。调用形式如下：

```
hal.scheduler->register_timer_process(AP_HAL_MEMBERPROC(&AP_Baro_MS5611::_update));
```

上面代码是以 MS5611 气压计驱动为例，其中 AP\_HAL\_MEMBERPROC() 宏，主要作用是**将一个 C++成员函数包装起来，作为一个回调参数**。其定义在 AP\_HAL\_Namespace.h 文件中，如下：

```
// macro to hide the details of AP_HAL::MemberProc
#define AP_HAL_MEMBERPROC(func) fastdelegate::MakeDelegate(this, func)
```

使用 hal.scheduler->millis() 和 hal.scheduler->micros() 可以记录时间。

好了，你可以试着自己编一个简单的 sketch，在 setup() 和 loop() 函数中练习一下 1 秒钟向 USB 终端输出一个时间或字符。

## 2、HAL 专属线程（有待测试）

以 PX4 为例，HAL 专属线程有：

- 1、 UART 线程，用于读、写串行接口数据（包括 USB）；
- 2、 定时器线程，支持 1kHz 定时功能；
- 3、 IO 线程，支持写 microSD、EEPROM、FRAM 等。

对于 Pixhawk，请准备一条调试电缆，[连接到 nsh console（serial 5 端口）](#)，波特率 57600。如果已经连接，试下“ps”命令，你会得到如下信息：

PID	PRI	SCHD	TYPE	NP	STATE	NAME
0	0	FIFO	TASK	READY	Idle	Task()
1	192	FIFO	KTHREAD	WAITSIG	hpwork()	
2	50	FIFO	KTHREAD	WAITSIG	lpwork()	
3	100	FIFO	TASK	RUNNING	init()	
37	180	FIFO	TASK	WAITSEM	AHRS_Test()	AHRS 线程
38	181	FIFO	PTHREAD	WAITSEM	<pthread>(20005400)	定时器线程
39	60	FIFO	PTHREAD	READY	<pthread>(20005400)	UART 线程
40	59	FIFO	PTHREAD	WAITSEM	<pthread>(20005400)	IO 线程
10	240	FIFO	TASK	WAITSEM	px4io()	
13	100	FIFO	TASK	WAITSEM	fmuservo()	
30	240	FIFO	TASK	WAITSEM	uavcan()	

上面的线程为定时器线程（优先级 181），UART 线程（60），IO 线程（59），以及其他线程诸如：px4io, fmuservo, uavcan, lpwork, hpwork and idle tasks

线程的主要目的是在不干扰主进程的情况下，在后台处理一些低优先级任务。例如 AP\_Terrain library，需要向 microSD 卡写地形文件，它的实现方式如下：

```
hal.scheduler->register_io_process(AP_HAL_MEMBERPROC(&AP_Terrain::io_timer));
```

注意：IO 线程优先级 59，相比定时器 181 优先级慢了很多。

## 3、Driver 专属线程

没什么好说的，请参考英文原版，需要提的一点是，我们可以利用 register\_io\_process() 和 register\_timer\_process() 来处理驱动的访问。

## 4、APM 驱动与板级（原生）驱动

我们可以看到 MPU6000 驱动有两个版本：一个是 APM 版本，在 `libraries/AP_InertialSensor/AP_InertialSensor_MPU6000.cpp`，另一个为原生代码版本，在 `PX4Firmware/src/drivers/mpu6000`。

注意，对于 Pixhawk，APM 代码使用的是 Pixhawk 原生驱动，因为原生驱动已经做得很好了。`libraries/AP_InertialSensor/AP_InertialSensor_PX4.cpp` 中可以查看详情。

在非 PX4 平台上，我们使用 `AP_InertialSensor_MPU6000.cpp` 驱动，在 PX4 平台上，我们就用 PX4 原生驱动 `AP_InertialSensor_PX4.cpp`

## 5、板级专属线程、任务

在上面第 2 节“HAL 专属线程”讲到“ps”命令显示的线程。很多都不是 `AP_HAL_PX4 Schedule` 启动的线程，这些线程列举如下：

- idle task – called when there is nothing else to run
- init – used to start up the system
- px4io – handle the communication with the PX4IO co-processor
- hpwork – PX4 高优先级驱动线程。handle thread based PX4 drivers (mainly I2C drivers)
- lpwork – PX4 低优先级驱动线程。handle thread based low priority work (eg. IO)
- fmuservo – AUX 输出。handle talking to the auxillary PWM outputs on the FMU
- uavcan – handle the uavcan CANBUS protocol

这些任务的启动，由 `rc.APM` 脚本文件(`ardupilot\mk\PX4\ROMFS\init.d\rc.APM`)指定。PX4 启动时，会读取该文件。`rc.APM` 属于 `nsh` 类型脚本。作为练习，你可以修改 `rc.APM` 脚本文件，增加一些 `sleep` 和 `echo` 命令，那么当 PX4 启动时，通过 `debug console`（也就是 `serial 5`）可以显示出来。

`rc.APM` 很重要，它同时也是 APM 在 PX4 板级上的启动脚本。

另外，[ardupilot/mk/PX4/ROMFS/init.d/rcS](#) 脚本，在 `rc.APM` 前面被调用。

更多内容，可以参考英文原版。

原生线程的启动代码如下：

```
hrt_call_every(&_call, 1000, _call_interval, (hrt_callout)&MPU6000::measure_trampoline, this);
```

等同于 `AP_HAL` 中的 `hal.scheduler->register_timer_process()`。上述代码的意思是，HRT（high resolution timer）高精度定时器，以 1000 微秒的周期调用 `MPU6000::measure_trampoline` 函数。这些操作是禁止中断的，最多占用数十微秒的时间。

上面的优先级非常高。下面的方法，是稍低优先级。

```
work_queue(HPWORK, &_work, (worker_t)&HMC5883::cycle_trampoline, this, 1);
```

用于处理 I2C 设备。大概花几百微秒的操作时间。是可以被中断的任务。如果是最低优先级，那么参数改为 `LPWORK`，这样的任务一般需要花费更长的时间。

## 6、AP\_Scheduler 任务调度系统

用于飞行器主线程，提供了简单的机制控制每个操作花费了多少时间。例如：1、等待一个新 IMU 采样；2、在每一个 IMU 采样周期之间调用一系列其他任务。

每一个飞行器都有一个 `AP_Scheduler::Task table` 任务列表，参考代码（`ardupilot\libraries\AP_Scheduler\Scheduler_test.pde`）类似如下：

```
static const AP_Scheduler::Task scheduler_tasks[] PROGMEM = {  
    { ins_update, 1, 1000 },  
    { one_hz_print, 50, 1000 },  
    { five_second_call, 250, 1800 },  
};
```

结构体第 1 列，循环调用的任务函数。第 2 列，调用频率（也叫 tick，一个 tick，就是一个最小时间单元，pixhawk 最小时间单位为 2.5ms，也就是 400Hz）。第 3 列为最大可能占用的操作时间（微秒），`scheduler.run()` 会传递当前可用的时间（微秒），如果时间不够，那么这个任务就 pass 掉了，不执行。

注意，`AP_Scheduler::Task table` 列表必须满足以下要求：

- 1、他们不能被阻塞。



- 2、在飞行时，他们不能调用 sleep function。
- 3、他们必须有可预估的最坏的运行时间。

你可以修改 Scheduler\_test.pde，加入自己的代码来读取气压计、罗盘、GPS、更新 AHRS 输出 roll/pitch。

补充：

关于最小时间单位的定义为 #define MAIN\_LOOP\_RATE 400 // pixhawk 采用

static const AP\_InertialSensor::Sample\_rate ins\_sample\_rate = AP\_InertialSensor::RATE\_400HZ;

## 7、信号灯

有 3 种方法可以避免多线程访问冲突：1、信号量；2、lockless data；3、PX4 ORB。

例如：I2C 驱动可以通过信号量，确保同一时间，只有一个 I2C 设备被使用。可以查看 ardupilot\libraries\AP\_Compass\AP\_Compass\_HMC5843.cpp 了解：

获得信号量：\_i2c\_sem->take(1);

释放信号量：\_i2c\_sem->give();

```
if(!_i2c_sem->take(1)) {  
    // the bus is busy - try again later  
    return;  
}  
  
bool result = read_raw();  
_i2c_sem->give();
```

## 8、Lockless Data Structures

Lockless Data Structures 比信号量要方便，例子见：

- the \_shared\_data structure in libraries/AP\_InertialSensor/AP\_InertialSensor\_MPU9250.cpp
- the ring buffers used in numerous places. A good example is libraries/DataFlash/DataFlash\_File.cpp

Go and have a look at these two examples, and prove to yourself that they are safe for concurrent access. For DataFlash\_File look at the use of the \_writebuf\_head and \_writebuf\_tail variables.

## 9、PX4 ORB

ORB(Object Request Broker)是 PX4 的互斥机制。

另外两种 PX4 驱动通信机制，列举如下：

- ioctl calls (see the examples in AP\_HAL\_PX4/RCOutput.cpp)
- /dev/xxx read/write calls (see \_timer\_tick in AP\_HAL\_PX4/RCOutput.cpp)

## 2.4 串行接口 UART 和 Console

详细参考：<http://dev.ardupilot.com/wiki/learning-ardupilot-uarts-and-the-console/>

UART 很重要，用于调试输出，数传、GPS 模块等。

### 1、5 个 UART

目前共定义了 5 个 UART，他们的用途分别是：

- uartA – 串行终端，通常是 Micro USB 接口，运行 MAVLink 协议。
- uartB – GPS1 模块。
- uartC – 主数传接口，也就是 Pixhawk telem1 接口。
- uartD – 次数传接口，也就是 telem2 接口。
- uartE – GPS2 模块。

有些 UART 具备双重角色，比如通过修改 SERIAL2\_PROTOCOL 参数，可以将 uartD 的 Mavlink telemetry 数传更改为 Frsky telemetry 数传（中国江苏产数传）。

测试 libraries/AP\_HAL/examples/UART\_test 目录下的 example sketch，分别对 5 个 UART 都输出 hello 消息。使用 USB 转串口工具，可以测试。

## 2、调试终端 Debug console

作为 5 个 UART 的补充，有些平台额外的还有一个 debug console 调试终端。你可以通过检查 HAL\_OS\_POSIX\_IO 宏定义来判断，诸如：

```
#if HAL_OS_POSIX_IO
    ::printf("hello console\n");
#endif
```

如果定义了 HAL\_OS\_POSIX\_IO，可以试着查看 AP\_HAL/AP\_HAL\_Boards.h 代码。

## 3、UART 函数

每个 UART 都一系列基本操作函数，主要有：

1. printf – formatted print
2. printf\_P – formatted print with progmem string (saves memory on AVR boards)
3. println – print and line feed
4. write – write a bunch of bytes
5. read – read some bytes
6. available – check if any bytes are waiting
7. txspace – check how much outgoing buffer space is available
8. get\_flow\_control – check if the UART has flow control capabilities

可以到 AP\_HAL 中查看他们的定义，并使用 UART\_TEST 进行测试。

## 4、UART 接口说明

众多 UART 接口，众多名称，他们的对应关系，我总结如下，如有问题，[欢迎发邮件至 30175224@qq.com](mailto:30175224@qq.com) 新浪@WalkAnt，欢迎指正。

代码定义	飞控板接口	PCB 电路表述	Serial 标号	说明
APM 代码中的表述	Pixhawk 外壳上的标识	STM32F4 芯片外设	PCB 硬件标号	
uartA	USB	Micro USB	USB	接 USB，支持 MAVLink 协议
uartB	GPS	UART4	Serial 3	接 GPS 模块，另 CAN2 接口
uartC	Telem1	UART2	Serial 1	接第 1 数传模块
uartD	Telem2	UART3	Serial 2	接第 2 数传模块
uartE	SERIAL4/5	UART8	Serial 4	一般接 GPS2 模块
Debug Console	SERIAL4/5	UART7	Serial 5	Debug Console 用于程序调试

## 2.5 学习 RC Input and Output

参考：<http://dev.ardupilot.com/wiki/learning-ardupilot-rc-input-output/>

RC Input，也就是遥控输入，用于控制飞行方向、改变飞行模式、控制摄像头等外围装置。ArduPilot 支持集中不同 RC input（取决于具体的硬件飞控板）：

1. **PPMSum** – on PX4, Pixhawk, Linux and APM2
2. **SBUS** – on PX4, Pixhawk and Linux
3. **Spektrum/DSM** – on PX4, Pixhawk and Linux
4. **PWM** – on APM1 and APM2
5. **RC Override (MAVLink)** – all boards

其中 SBUS 和 Spektrum/DSM 是串行协议，SBUS 为 100kbps 反 UART 协议，Spektrum/DSM 为 115200bps UART 协议。对于 PX4，这些协议是通过硬件 UARTs 实现的，而有些 Linux 系统是通过软件 UARTs 实现的。（原文：Some boards implement these using hardware UARTs (such as on PX4) and some implement them as bit-banged software UARTs (on Linux).）

RC Output，是指飞控接受到 RC 输入后，再将其处理后，输出到伺服和电机上。RC Output 默认 50Hz PWM 信号。对于 ArduCopter 多轴飞行器和直升机，输出频率为 400Hz。

## 1、RCInput 对象 (AP\_HAL)

RCInput 对象声明：

```
AP_HAL::RCInput* rcin;
```

相关例程：[libraries/AP\\_HAL/examples/RCInput/RCInput.pde](#)，试着动遥控器手柄，看看输出是否符合预期。

## 2、RCOutput 对象 (AP\_HAL)

RCOutput 对象声明：

```
AP_HAL::RCOutput* rcout;
```

不同的飞控，代码实现有所不同，可能包含了片上定时器、I2C、经由协处理器(PX4IO)输出等程序。

相关例程：[libraries/AP\\_HAL/examples/RCOutput/RCOutput.pde](#) 这段程序从 1 通道到 14 通道，控制电机从最小转速到最大转速逐级变化。

## 3、RC\_Channel 对象

hal.rcin 和 hal.rcout 对象，为低级调用。最常用的是使用更高级的 RC\_Channel 对象来实现 RC input 和 output。它允许用户对参数进行配置，例如每个通道 min/max/trim 值，同时支持辅助 AUX 通道函数，还可对 input output 进行比例缩放处理等。

相关例程：[libraries/RC\\_Channel/examples/RC\\_Channel/RC\\_Channel.pde](#) 例程教你如何 setup、read、copy input to output。

## 4、RC\_Channel 奇怪的 input/output 设置

看代码时，有些地方程序会让你感到奇怪，有一些是由于程序代码的不完善产生的，有一些则不是。

例如，很多变量作用在 input 和 output 上：

```
radio_out = (pwm_out * _reverse) + radio_trim;
```

上述代码中的 radio\_trim，是一个 trim 叠加，用来修正遥控器的值。

又例如，对于固定翼飞行器，roll (横滚) 输入，成为了 steer (转向 yaw)。对于 ArduCopter 中的多轴飞行器，在处于 Drift 模式 (漂移模式) 时，我们看到，pitch 用于前飞，roll 用于转向 (而不是传统 yaw 用于转向)。以后，APM 团队会将其纠正过来，将这两个概念分开。大家知道这么回事就 OK 了。

## 5、RC\_Channel\_aux 对象

另一个非常重要的类：RC\_Channel\_aux class，它是 RC\_Channel 的子类。它有很多特点可供用户使用。这个会有一点比较难以理解，举个例子：

用户想要使用通道 6(Channel 6)对航拍设备的横向稳定进行控制，那么他可以将 **FUNCTION** 设置为 21，枚举变量类型为“k\_rudder” (偏航，偏转，转向的意思)。如下：

```
AP_GROUPINFO("FUNCTION", 1, RC_Channel_aux, function, 21),
AP_GROUPEND
```

如果程序中调用此代码，RC\_Channel\_aux::set\_servo\_out(RC\_Channel\_aux::k\_rudder, 4500);，那么所有 FUNCTION 设为 21(k\_rudder) 的通道(channel)都将输出满偏(4500 就是满偏最大值)。

在相应的 update\_aux\_servo\_function()代码中，

```
case RC_Channel_aux::k_rudder:
    _aux_channels[i]->set_angle(4500); // 设置最大角度。
    break;
```

注意这是一对多的设置。就我的理解，其实也就是我们常说的混控输出。比如在辅助通道 6 中，我们可以将其他通道设置为使用 function = 21。那么其他使用了 21 的通道，将会被通道 6 混控。(这个很复杂，我也没太明白，对这个有更好理解的，请一定告诉我，相互学习：[30175224@qq.com](mailto:30175224@qq.com)。当然如果日后我能有更好的理解，我会更新本博客。)

// FUNCTION 为 1-27 , function 参数。

```
typedef enum
{
    k_none                = 0,          ///< disabled
    k_manual               = 1,          ///< manual, just pass-thru the RC in signal
    k_flap                 = 2,          ///< flap
    k_flap_auto            = 3,          ///< flap automated
    k_aileron              = 4,          ///< aileron
    k_unused1              = 5,          ///< unused function
    k_mount_pan            = 6,          ///< mount yaw (pan)
    k_mount_tilt           = 7,          ///< mount pitch (tilt)
    k_mount_roll           = 8,          ///< mount roll
    k_mount_open           = 9,          ///< mount open (deploy) / close (retract)
    k_cam_trigger          = 10,         ///< camera trigger
    k_egg_drop             = 11,         ///< egg drop
    k_mount2_pan           = 12,         ///< mount2 yaw (pan)
    k_mount2_tilt          = 13,         ///< mount2 pitch (tilt)
    k_mount2_roll          = 14,         ///< mount2 roll
    k_mount2_open          = 15,         ///< mount2 open (deploy) / close (retract)
    k_dspoiler1            = 16,         ///< differential spoiler 1 (left wing)
    k_dspoiler2            = 17,         ///< differential spoiler 2 (right wing)
    k_aileron_with_input   = 18,         ///< aileron, with rc input
    k_elevator             = 19,         ///< elevator
    k_elevator_with_input  = 20,         ///< elevator, with rc input
    k_rudder               = 21,         ///< secondary rudder channel
    k_sprayer_pump         = 22,         ///< crop sprayer pump channel
    k_sprayer_spinner      = 23,         ///< crop sprayer spinner channel
    k_flaperon1            = 24,         ///< flaperon, left wing
    k_flaperon2            = 25,         ///< flaperon, right wing
    k_steering             = 26,         ///< ground steering, used to separate from rudder
    k_parachute_release    = 27,         ///< parachute release
    k_epm                  = 28,         ///< epm gripper
    k_nr_aux_servo_functions ///< This must be the last enum value (only add new values _before_ this one)
} Aux_servo_function_t;
```

```
AP_Int8      function;          ///< see Aux_servo_function_t enum
```

## 2.6 存储与 EEPROM 管理

详细参考：<http://dev.ardupilot.com/wiki/learning-ardupilot-storage-and-EEPROM-management/>

用户参数、航点、集结点、地图数据以及其他有用的信息需要存储。ArduPilot 提供 4 个基本存储接口：

- 1、**AP\_HAL::Storage** 对象：hal.storage；
- 2、**StorageManager 库**，是 hal.storage 更高级别的封装；
- 3、**DataFlash** 用于日志存储；
- 4、**Posix IO 函数**，是传统文件系统读写函数。

其他用于永久存储信息的函数库，都是基于以上 4 种实现。例如：AP\_Param library（用于处理用户可配置参数）是建立在 StorageManager 库之上的，而 StorageManager 库则是基于 AP\_HAL::Storage 之上。AP\_Terrain library（用于处理地形数据）则是建立在 Posix IO functions 之上，用于操作地形数据库。

## 1、AP\_HAL::Storage library

AP\_HAL::Storage 对象适用于所有 ArduPilot 硬件平台。最小支持 4096 字节空间的存储，一些类似 PX4v1 的板子有 8K EEPROM，[Pixhawk](#) 有 16K FRAM。所有这些都封装在 AP\_HAL::Storage API 中。

hal.storage API，非常简单，仅 3 个函数：

- 1、init()，初始化存储系统；
- 2、read\_block()，读块数据；
- 3、write\_block()，写块数据。

之所以这么简单，是因为 APM 团队鼓励开发者使用 StorageManager API，而不是 hal.storage。只有在代码移植或调试时，使用 hal.storage 会比较方便（原文：You should only be delving into hal.storage when doing bringup of a new board, or when debugging.）。

存储空间的大小，在 [AP\\_HAL/AP\\_HAL\\_Boards.h](#) 文件中的 HAL\_STORAGE\_SIZE 宏中定义，如下：

```
#define CONFIG_HAL_BOARD_SUBTYPE HAL_BOARD_SUBTYPE_PX4_V2
#define HAL_STORAGE_SIZE          16384          // 存储空间 16KB
#endif
```

也就是说，我们不支持动态存储空间的定义。如果希望使用动态存储空间，可以使用 Posix IO。

## 2、StorageManager library

在将 ArduPilot 代码移植到一个新的硬件板上时，hal.storage API 非常简单，但是在操作存储区时就不那么好使了。我们会采用 StorageManager。StorageManager library 提供对存储区域“伪连续块”（一般用作不同的功能和目的）的访问。正因此我们将存储区域分配了不同的功能：

- 1、参数区；
- 2、飞行区域限制点数据区；
- 3、航点数据区；
- 4、集结点数据区。

参见：[libraries/StorageManager/StorageManager.cpp](#)，我们可以看到存储区域的划分：

```
const StorageManager::StorageArea StorageManager::layout_copter[STORAGE_NUM_AREAS] PROGMEM = {
    // ----- 0-4096 分配给了 AVR 版本的 APM
    { StorageParam, 0, 1536}, // 0x600 param bytes
    { StorageMission, 1536, 2422},
    { StorageRally, 3958, 90}, // 6 rally points
    { StorageFence, 4048, 48}, // 6 fence points
    #if STORAGE_NUM_AREAS >= 8
        // ----- 4096-8192 分配给了 PX4 版本
        { StorageParam, 4096, 1280},
        { StorageRally, 5376, 300},
        { StorageFence, 5676, 256},
        { StorageMission, 5932, 2132}, // leave 128 byte gap for
                                         // expansion and PX4 sentinel
    #endif
    #if STORAGE_NUM_AREAS >= 12
        // ----- 8192-16384 分配给了 Pixhawk 版本
        { StorageParam, 8192, 1280}, // 类型 偏移量 长度
        { StorageRally, 9472, 300},
        { StorageFence, 9772, 256},
        { StorageMission, 10028, 6228}, // leave 128 byte gap for expansion
    #endif
};
```

对于上面的存储分布，我们可以观察到 AVR 版本用到存储地址是 0-4095，而 PX4 用到地址是 4096-8191，Pixhawk 用到的地址是

8192-16383。这样的结构，是为了更好的与之前的版本兼容。这样一来，用户在更新最新的固件时，所有之前配置的参数将不会改变，将继续起作用。

StorageManager API 也提供对类似整型数的读写访问，AP\_Mission 中就会利用这个 API 来存储和恢复航点数据。  
相关例程 ( [libraries/StorageManager/examples/StorageTest.pde](#) ) 对 StorageManager layer 和 AP\_HAL::Storage object 进行了测试。它对随机的偏移量、随机的长度进行了随机的 IO 操作。这也就意味可能会出现跨边界访问。这个例程非常有用，它用于对 StorageManager API 进行严苛测试，同样对于移植 ArduPilot 到新硬件平台也是极为有用的，因为它对 EEPROM 的访问函数进行了很严格的测试。

注意 StorageTest 是一个毁坏性的测试，它将会删除你之前存储的参数和航点。一定要记得测试之前，备份你的配置。  
存储对象的声明，一般如下：

```
StorageAccess AP_Param::_storage(StorageManager::StorageParam);  
又或者  
StorageAccess AP_Rally::_storage(StorageManager::StorageRally);  
StorageAccess AP_Mission::_storage(StorageManager::StorageMission);  
StorageAccess AP_Limit_Geofence::_storage(StorageManager::StorageFence);
```

### 3、DataFlash library

另一类存储，就是飞行日志存储，这个基于 DataFlash library。这个库的名字看上去有些怪怪的，实际上这个库最开始是为 APM1 的 DataFlash 芯片设计的，它原本是一个硬件驱动库，后来慢慢演变为一个通用日志系统，这个可以在代码中找到蛛丝马迹（这些都是以前的痕迹，不是最好的代码实现方式）。

现在 DataFlash API 主要用于实现日志存储。它允许你自定义日志消息的数据结构。例如 GPS 消息，用于记录 GPS 传感器的日志数据。它能够非常有效存储这些数据，它同时也对其他库提供相应的 APIs，用来进行日志回传、下载。  
LOG 数据结构是自定义的，其结构可以查看日志文件的 FMT 消息。FMT 消息对应的其他数据的存储格式。  
相关例程 [libraries/DataFlash/examples/DataFlash\\_test/DataFlash\\_test.pde](#)。这里描述了数据的存储结构和数据格式。简单列举如下：

**第一点**，在 .log 文件中，我们可以看到如下格式的表达：

```
FMT, 128, 89, FMT, BBnNZ, Type, Length, Name, Format, Columns  
FMT, 129, 23, PARM, Nf, Name, Value  
FMT, 130, 45, GPS, BIHBcLLeeEefl, Status, TimeMS, Week, NSats, HDop, Lat, Lng, RelAlt, Alt, Spd, GCrs, VZ, T  
FMT, 131, 31, IMU, Iffffff, TimeMS, GyrX, GyrY, GyrZ, AccX, AccY, AccZ  
FMT, 132, 67, MSG, Z, Message
```

**第二点**，上述格式，对应的代码（参见 DataFlash.h）：

```
#define LOG_BASE_STRUCTURES \  
    { LOG_FORMAT_MSG, sizeof(log_Format), \  
      "FMT", "BBnNZ",      "Type, Length, Name, Format, Columns" },    \  
    { LOG_PARAMETER_MSG, sizeof(log_Parameter), \  
      "PARM", "Nf",      "Name, Value" },    \  
    { LOG_GPS_MSG, sizeof(log_GPS), \  
      "GPS",  "BIHBcLLeeEefl", "Status, TimeMS, Week, NSats, HDop, Lat, Lng, RelAlt, Alt, Spd, GCrs, VZ, T" }, \  
    { LOG_IMU_MSG, sizeof(log_IMU), \  
      "IMU",  "Iffffff",      "TimeMS, GyrX, GyrY, GyrZ, AccX, AccY, AccZ" }, \  

```

上述结构，以 LOG\_IMU\_MSG 为例讲解：

信息类型 ID	数据大小	信 息 名称	数 据 类型	数据 1	数据 2	数据 3	数据 4	数据 5	数据 6	数据 7
LOG_IMU_MSG	sizeof(log_IMU)	IMU	Iffffff	TimeMS	GyrX	GyrY	GyrZ	AccX	AccY	AccZ
131	31(字节)	IMU	l:整型; f:浮点	整型 46481	0.000703	-0.000190	-0.000359	-0.133995	0.034236	-9.748702

**第三点**，日志文件(.log)的一条数据如下：

IMU, 46481, 0.000703, -0.000190, -0.000359, -0.133995, 0.034236, -9.748702

**第四点**，消息类型的定义：

```
// message types for common messages
// 消息类型,,, 对应 FMT 中的消息类型,,, 见日志文件 .log 文件。
#define LOG_FORMAT_MSG    128
#define LOG_PARAMETER_MSG 129
#define LOG_GPS_MSG       130
#define LOG_IMU_MSG       131
#define LOG_MESSAGE_MSG   132
#define LOG_RCIN_MSG      133
#define LOG_RCOUT_MSG     134
#define LOG_IMU2_MSG      135
...
```

**第五点**, log\_IMU 的结构, 共占用  $3 + 4 + 12 + 12 = 31$  字节。

```
struct PACKED log_IMU {
    LOG_PACKET_HEADER;          // 3
    uint32_t timestamp;         // 4
    float gyro_x, gyro_y, gyro_z; // 4*3 = 12
    float accel_x, accel_y, accel_z; // 4*3 = 12
};
```

**第六点**：如果要增加自定义的数据结构, 那么可以像以下代码一样增加。

```
#define LOG_TEST_MSG 1

struct PACKED log_Test {
    LOG_PACKET_HEADER;
    uint16_t v1, v2, v3, v4;
    int32_t l1, l2;
};

static const struct LogStructure log_structure[] PROGMEM = {
    LOG_COMMON_STRUCTURES,
    { LOG_TEST_MSG, sizeof(log_Test),          // 增加自定义格式数据
      "TEST", "HHHHii",          "V1,V2,V3,V4,L1,L2" } // 增加自定义格式数据
};
```

**第七点**：具体的数据结构操作

```
DataFlash.Init(log_structure, sizeof(log_structure)/sizeof(log_structure[0]));
log_num = DataFlash.StartNewLog();
DataFlash.WriteBlock(&pkt, sizeof(pkt));
```

DataFlash API 隐藏了底层如何存储 log 文件的细节。另外, 对于 Pixhawk or Linux 这样的支持 Posix IO 的系统, 日志文件是存储在 microSD 卡的 “LOGS” 目录中的。用户可以直接抽出 SD 卡, 直接拷贝到电脑中。

## 4、Posix IO

有些板子是带操作系统的, 支持类似 Posix API, 如 Linux 和 NuttX。AP\_Terrain library 就是一个典型的例子。地形数据对于 EEPROM



是非常的大，经常要随机的存储。DataFlash API 就不够灵活了，同时有了 Posix IO 支持，也就没必要再用 DataFlash 了。

查看 [AP\\_HAL\\_Boards.h](#) 文件，确认 HAL\_OS\_POSIX\_IO 宏已定义，如下：

```
#define HAL_OS_POSIX_IO 1 // 带文件系统，has posix-like filesystem IO
```

下面给出了 LOG 和 TERRAIN 文件存放路径：

```
#define HAL_BOARD_LOG_DIRECTORY "/fs/microsd/APM/LOGS" // LOG 日志文件地址
```

```
#define HAL_BOARD_TERRAIN_DIRECTORY "/fs/microsd/APM/TERRAIN" // 地表、地形文件地址
```

有上述信息，就表示支持 Posix IO 功能，另外需要说明的是：

- 1、Posix IO 函数，只能通过 IO timer 定时器，或者其他低优先级线程调用。IO 线程优先级 59。
- 2、不要通过其他 API 直接调用，哪怕是简单 stat() 函数，都不可以，除非你长得太帅。
- 3、尽量少存储，存储数据长度小，尽量少用 seek（搜寻）功能。

很简单，一个原则，不要太耗时，影响飞控代码执行。一个简单的针对 SD 卡的 IO 操作有可能花上一秒钟，这段时间足够让你的飞行器翻转，垂直掉落，直接炸鸡了。Pixhawk SD 卡读写操作一般几毫秒，偶尔花费的时间会很长。现在在你知道怎么做了？

相关例程 [libraries/AP\\_Terrain/TerrainIO.cpp](#)，我们会发现处理 IO 的状态机都是通过 AP\_Terrain::io\_timer 调用的。

### 3 源代码预览

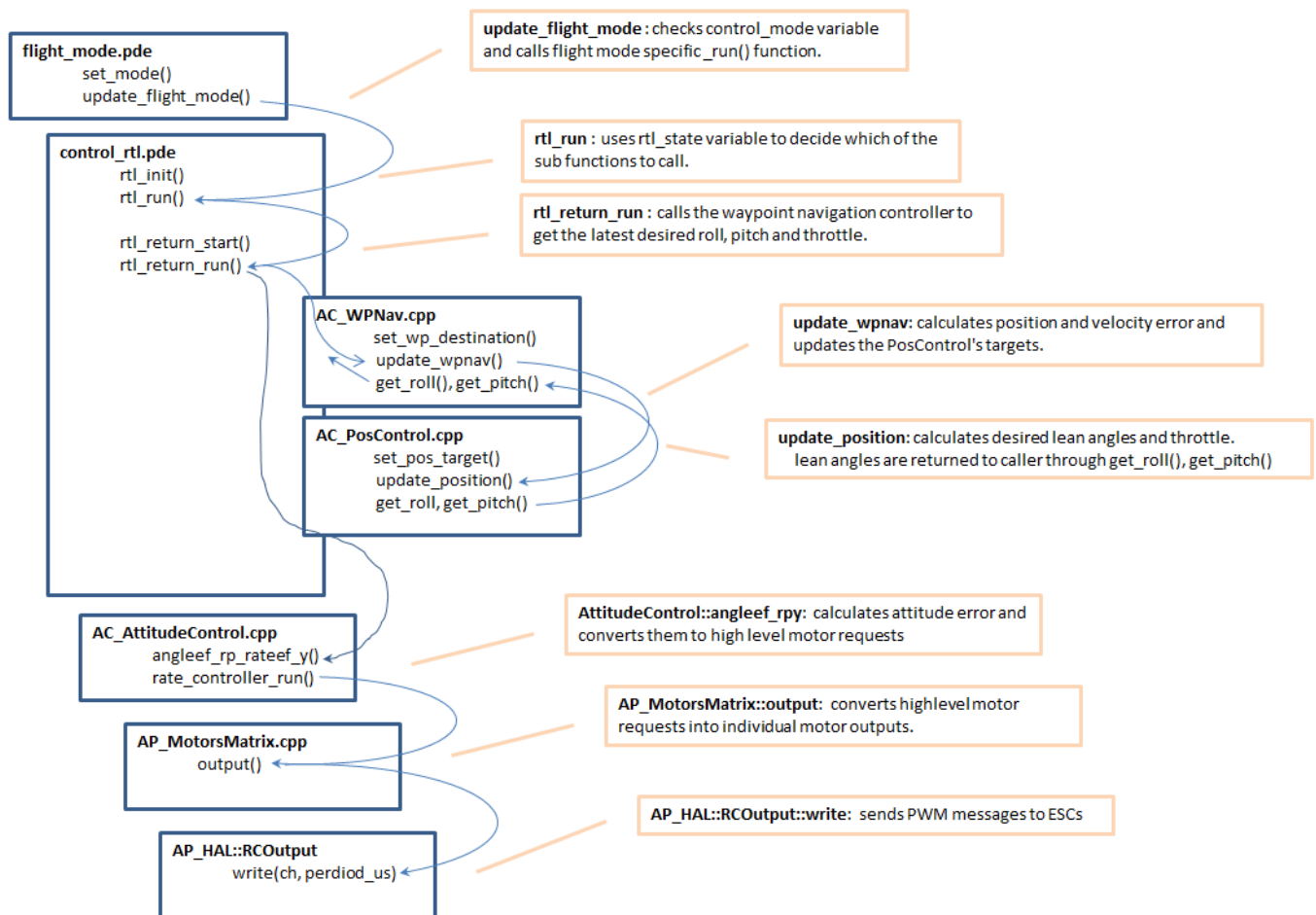
英文：<http://dev.ardupilot.com/wiki/apmcopter-code-overview/>

中文：<http://liung.github.io/blog/apm/2014-08-30-APM-Arducopter-代码预览.html>

APM::Copter 代码主要放在 ArduCopter 文件夹中，并且和 ArduPlane 和 ArduRover 使用同样的库文件。

下面这张图展示了从飞行模式到电机输出的高级别代码概要：

AutoPilot flight modes such as AltHold, RTL, Auto





## 3.1 APM:Copter 相关库

<http://dev.ardupilot.com/wiki/apmcopter-programming-libraries/>

[http://liung.github.io/blog/apm/2014-08-30-APM-Arducopter 相关库介绍.html](http://liung.github.io/blog/apm/2014-08-30-APM-Arducopter-相关库介绍.html)

这些库文件也同样被 ArduPlane 和 ArduRover 所使用。下面将列出一系列高层次的库的说明和它们的函数说明。

### 1、核心库

[AP\\_AHRS](#)：采用 DCM（方向余弦矩阵方法）或 EKF（扩展卡尔曼滤波方法）预估飞行器姿态。

[AP\\_Common](#)：所有执行文件（sketch 格式，arduino IDE 的文件）和其他库都需要的基础核心库。

[AP\\_Math](#)：包含了许多数学函数，特别对于矢量运算

[AC\\_PID](#)：PID 控制器库

[AP\\_InertialNav](#)：扩展带有 gps 和气压计数据的惯性导航库

[AC\\_AttitudeControl](#)：姿态控制相关库

[AP\\_WPNav](#)：航点相关的导航库

[AP\\_Motors](#)：多旋翼和传统直升机混合的电机库

[RC\\_Channel](#)：更多的关于从 APM\_RC 的 PWM 输入/输出数据转换到内部通用单位的库，比如角度

[AP\\_HAL](#)，[AP\\_HAL\\_AVR](#)，[AP\\_HAL\\_PX4](#)：硬件抽象层库，提供给其他高级控制代码一致的接口，而不必担心底层不同的硬件。

### 2、传感器相关库

[AP\\_InertialSensor](#)：读取陀螺仪和加速度计数据，并向主程序执行标准程序和提供标准单位数据（deg/s，m/s）。

[AP\\_RangerFinder](#)：声呐和红外测距传感器的交互库

[AP\\_Baro](#)：气压计相关库

[AP\\_GPS](#)：GPS 相关库

[AP\\_Compass](#)：三轴罗盘相关库

[AP\\_OpticalFlow](#)：光流传感器相关库

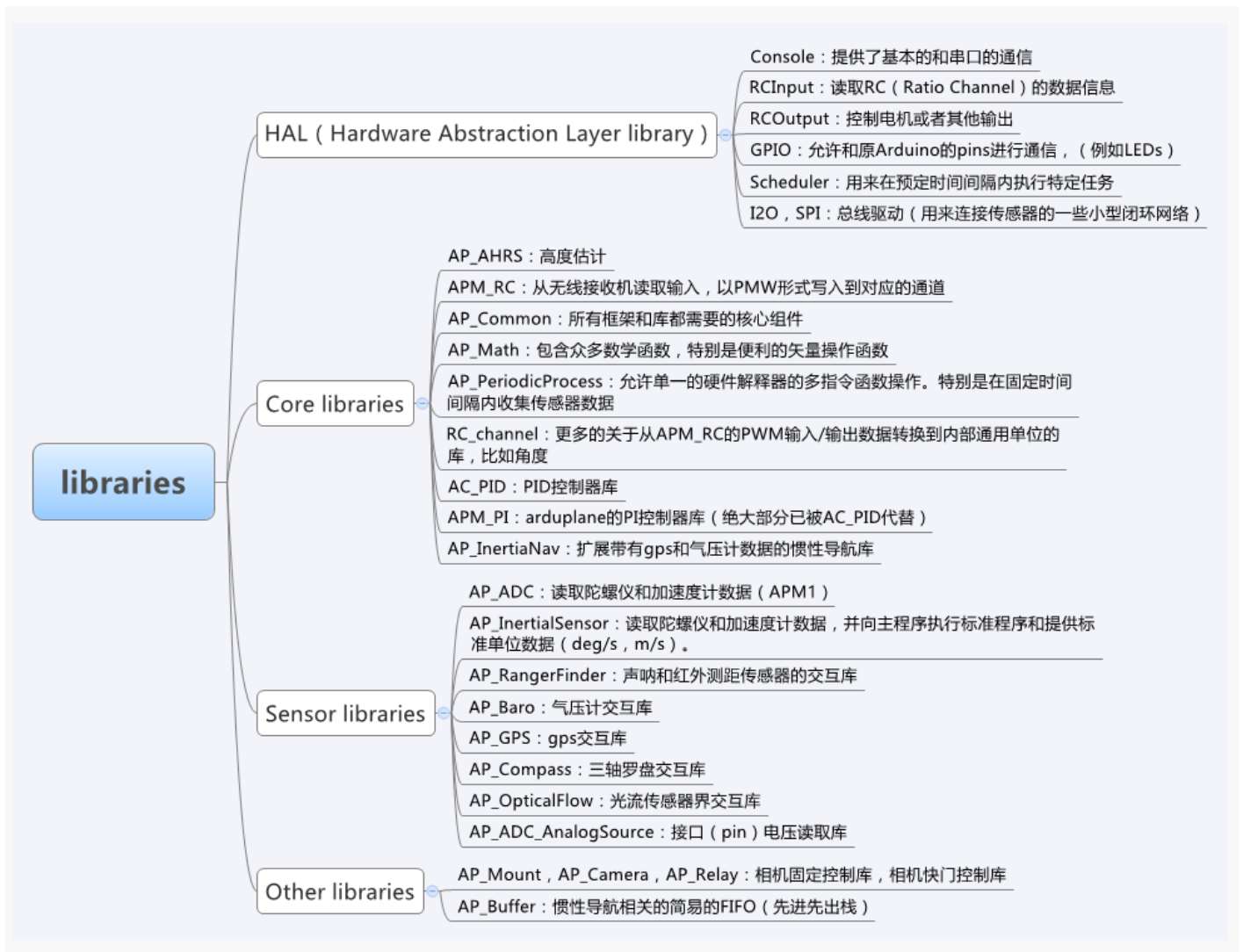
### 3、其他库

[AP\\_Mount](#)，[AP\\_Camera](#)，[AP\\_Relay](#)：相机安装控制库，相机快门控制库

[AP\\_Mission](#)：从 eeprom（电可擦只读存储器）存储/读取飞行指令相关库

[AP\\_Buffer](#)：惯性导航时所用到的一个简单的堆栈（FIFO，先进先出）缓冲区

关于库的导航图，如下：



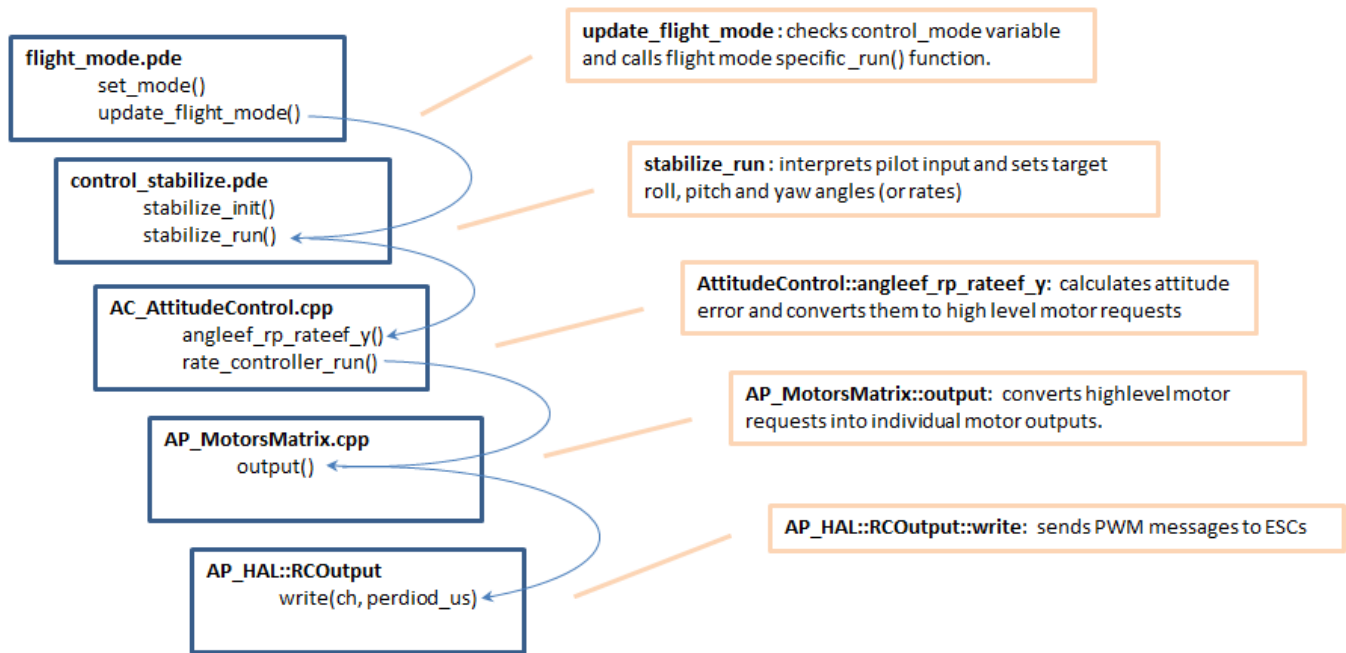
## 3.2 姿态控制预览

英文参考 : <http://dev.ardupilot.com/wiki/apmcopter-programming-attitude-control-2/>

本节源自 : [http://liung.github.io/blog/apm/2014-08-31-APM-ArduCopter\\_姿态控制概览.html](http://liung.github.io/blog/apm/2014-08-31-APM-ArduCopter_姿态控制概览.html)

手动飞行模式, 诸如自稳模式 ( Stabilize Mode )、特技模式 ( Acro Mode )、飘逸模式 ( Drift Mode ), 其程序结构如下图 :

## Manual flight modes such as Stabilize, Acro, Drift



在主循环执行过程中（比如 Pixhawk 的任务调度周期 2.5ms，400Hz；APM2.x 为 10ms，100Hz），每一个周期，程序会按下述步骤执行：

- 首先，高层次文件 **flight\_mode.pde** 中的 **update\_flight\_mode()** 函数被调用。通过检查 **control\_mode** 变量，飞行器的飞行模式（使用变量），然后执行相应飞行模式下的 **<flight mode>\_run()** 函数（如自稳模式的 **stabilize\_run**，返航模式（RTL）的 **rtl\_run** 等）。执行 **<flight mode>\_run** 的结果是，系统将会找到与飞行模式相对应的命名为 **control\_<flight mode>.pde** 飞行控制文件（比如：**control\_stabilize.pde**，**control\_rtl.pde** 等）。
- **<flight mode>\_run** 函数负责将用户的输入（从 **g.rc\_1.control\_in**，**g.rc\_2.control\_in** 等读入）转换为此时飞行模式下的倾斜角（lean angle）、滚转速率（rotation rate）、爬升率（climb rate）等（也就是设置目标值 **roll\pitch\yaw\throttle**）。举个例子：**AltHold**（定高，altitude hold）模式中将用户的滚转和俯仰输入转换为倾斜角（单位：角度/°），将偏航输入转换为滚转速率（单位：°/s），将油门输入转换为爬升率（单位：cm/s）。
- **<flight mode>\_run** 函数最后还必须要完成的就是将**预期角度、速率等参数**传送给姿态控制和/或方位控制库（它们都放在 **AC\_AttitudeControl** 文件夹内）。
- **AC\_AttitudeControl** 库提供了 5 种可能的方法来调整飞行器的姿态，下面来说明最通用的三种方法：
  - ◇ 1) **angle\_of\_roll\_pitch\_rate\_of\_yaw()**: 该函数需要一个地轴系坐标下滚转和偏航角度，一个地轴系坐标下的偏航速率。例如：传递给该函数三个参数分别为，**roll = -1000**，**pitch = -1500**，**yaw = 500** 代表飞行器此时向左倾斜 10°，低头 15°，向右偏航速率为 5°/s。
  - ◇ 2) **angle\_of\_roll\_pitch\_yaw()**: 该函数接受地轴系下的滚转、俯仰和偏航角。和上面的函数类似，不过参数 **yaw = 500** 代表飞行器北偏东 5°。
  - ◇ 3) **rate\_of\_roll\_pitch\_yaw()**: 该函数接受一个体轴系下的滚转、俯仰和偏航角速率（°/s）。例如：传递给该函数三个参数：**roll = -1000**，**pitch = -1500**，**yaw = 500** 代表飞行器此时左倾速率 10°/s，低头速率 15°/s，绕 Z 轴速率为 5°/s。
- 当上述这些函数调用之后，就会接着调用 **AC\_AttitudeControl::rate\_controller\_run()** 函数，将上面所列举的函数的输出转化为滚转、偏航和俯仰输入，并使用 **set\_roll**、**set\_pitch**、**set\_yaw** 和 **set\_throttle** 方法将这些输入发送给 **AP\_Motors** 库。

另外，

- **AC\_PosControl** 库用来控制飞行器的 3D 方位。不过通常只用来调整比较简单的 Z 轴方向（如姿态控制），这是因为许多需要复杂 3D 方位调整的飞行模式（例如悬停 Loiter）使用的是“**AC\_WPNav** 库”。总之，AC\_PosControl 库中常用的方法有：
  - ◇ 1) **set\_alt\_target\_from\_climb\_rate()**: 将爬升率（cm/s）作为参数，用来更新一个需要调整的相对高度目标。

- ✧ 2) `set_pos_target()`:接受一个以系统中的 `home` 位置作为参考点的 3D 位置矢量 ( 单位 : cm )。
- 如果调用了 `AC_PosControl` 中的任何一个方法,那么在该飞行模式下就必须调用函数 `AC_PosControl::update_z_controller()`。这样的话,就可以启用 z 轴的方位控制 PID 循环,并向 `AP_Motors` 库发送低级别的油门信息。同样,如果调用了 xy 轴的函数,那就必须调用 `AC_PosControl::update_xy_controller()` 函数。
- `AP_Motors` 库含有“电机混合模式”代码。这些代码负责将从 `AC_AttitudeControl` 和 `AC_PosControl` 库发送过来的滚转、俯仰、偏航角度和油门值信息转换为电机的相对输出值 ( 例如 : PWM 值 )。因此,这样高级别的库就必须使用如下函数 :
  - ✧ 1) `set_roll()`,`set_pitch()`,`set_yaw()` : 接受在[-4500,4500]角度范围内的滚转、俯仰和偏航角。这些参数不是期望角度或者速率,更准确的讲,它仅仅是一个数值。例如, `set_roll(-4500)`将代表飞行器尽可能快的向左滚转。
  - ✧ 2) `set_throttle()`:接受一个范围在[0,1000]的相对油门值。0 代表电机关闭,1000 代表满油门状态。
- 虽然对于不同飞行器构型 ( 如四旋翼, Y6, 传统直升机等 ) 的控制代码中有许多不同的类,但这些类中都有一个相同的函数 `output_armed` ,负责将这些滚转、俯仰、偏航和油门值转换为 PWM 类型输入值。这转换的过程中,会应用到 `stability patch` ,用来控制由于飞行器构型限制所引起的轴系的优先级问题 ( 例如四旋翼的四个电机不可能在做最大速度滚转时四个电机的油门同时达到最大,因为必须一部分电机输出小于另一部分才能引起滚转 )。在执行函数 `output_armed` 的最后,还将调用 `hal.rcout->write()` ,把期望 PWM 值传递给 `AP_HAL` 层。
- `AP_HAL` 库 ( 硬件抽象层 ) 提供了针对所有飞控板统一的接口。实际控制中, `hal.rc_out->write()` 函数将接受到的来自于 `AP_Motors` 类中指定的 PWM 值,输出至飞控板对应的 PWM 端口 ( pin 端 )。

## 3.3 添加新的参数

英文参考 : <http://dev.ardupilot.com/wiki/code-overview-adding-a-new-parameter/>

本节源自 : <http://liung.github.io/blog/apm/2014-09-02-APM-添加新的参数.html>

### 1、在主执行代码中添加参数

**第一步 :**

**Step #1 :**

在文件 `Parameters.h` 参数类中的枚举变量 ( enum ) 的合适位置,像下面代码块最后一行一样添加你自己的新的参数。你需要注意下面这些事情 :

- 尽量在执行类似功能的参数区域添加新的参数,或者最坏的情形下就是在“Misc ( 混合 )”区域的末尾添加。
- 确保你添加的参数区域中还可以有编号添加新的参数。检查是否能继续添加参数的方法是 : 检查参数的计数,确保你所要添加的参数的上一个元素编号要小于你的下一部分代码的编号。比如, `Misc` 部分的第一个参数起始于 #20。`my_new_parameter` 是 #36。如果下一部分参数开始于 #36,那么我们就不能在这里添加这个新参数。
- 不要在一个代码块的中间添加新的参数,那样容易造成现存参数对应的信息的改变。
- 不要在参数旁边用“弃用 ( deprecated )”或“移除 ( remove )”做注解,这是因为一些使用者将此注释用作在 eeprom 上的旧的参数的默认注解,如果你添加的新参数也是这样注解,那么就让人就会看起来很奇怪和疑惑。

```
enum {
    // Misc
    //
    k_param_log_bitmask = 20,
    k_param_log_last_filename,    // *** Deprecated - remove
                                   // with next eeprom number
                                   // change
    k_param_toy_yaw_rate,        // THOR The memory
                                   // location for the
                                   // Yaw Rate 1 = fast,
                                   // 2 = med, 3 = slow

    k_param_crosstrack_min_distance, // deprecated - remove with next eeprom number change
    k_param_rssi_pin,
```

```

k_param_throttle_accel_enabled,    // deprecated - remove
k_param_wp_yaw_behavior,
k_param_acro_trainer,
k_param_pilot_velocity_z_max,
k_param_circle_rate,
k_param_sonar_gain,
k_param_ch8_option,
k_param_arwing_check_enabled,
k_param_sprayer,
k_param_angle_max,
k_param_gps_hdop_good,            // 35

```

```

k_param_my_new_parameter,        // 36

```

## 第二步：

### Step #2：

在枚举变量后面的参数类中声明上面枚举变量提到的参数。可使用的类型包括 `AP_Int8`, `AP_Int16`, `AP_Float`, `AP_Int32`, `AP_Vector3` (目前还不支持 unsigned integer 无符号整型)。新的枚举变量的名称应该保持一致，只是去掉了前缀 `k_param_`。

```

// 254,255: reserved
};

```

```

AP_Int16    format_version;
AP_Int8     software_type;

```

```

// Telemetry control

```

```

//

```

```

AP_Int16    sysid_this_mav;
AP_Int16    sysid_my_gcs;
AP_Int8     serial3_baud;
AP_Int8     telem_delay;

```

```

AP_Int16    rtl_altitude;
AP_Int8     sonar_enabled;
AP_Int8     sonar_type;    // 0 = XL, 1 = LV,
                          // 2 = XLL (XL with 10m range)
                          // 3 = HRLV

```

```

AP_Float    sonar_gain;
AP_Int8     battery_monitoring;    // 0=disabled, 3=voltage only,
                          // 4=voltage and current

```

```

AP_Float    volt_div_ratio;
AP_Float    curr_amp_per_volt;
AP_Int16    pack_capacity;    // Battery pack capacity less reserve
AP_Int8     failsafe_battery_enabled;    // battery failsafe enabled
AP_Int8     failsafe_gps_enabled;    // gps failsafe enabled
AP_Int8     failsafe_gcs;    // ground station failsafe behavior
AP_Int16    gps_hdop_good;    // GPS Hdop value below which represent a good position

```

```

AP_Int16    my_new_parameter;        // my new parameter's description goes here

```

### 第三步：

#### Step #3：

在 [Parameters.pde](#) (.cpp) 文件中向 var\_info 表中添加变量的声明信息。

```
// @Param: MY_NEW_PARAMETER
// @DisplayName: My New Parameter
// @Description: A description of my new parameter goes here
// @Range: -32768 32767
// @User: Advanced
GSCALAR(my_new_parameter, "MY_NEW_PARAMETER", MY_NEW_PARAMETER_DEFAULT),
```

地面站（如 Mission Planner）中将使用 @Param ~ @User 的注释信息向使用者说明用户所设置的变量的范围等。

### 第四步：

#### Step #4:

在 [config.h](#) 中添加你的新参数。

```
#ifndef MY_NEW_PARAMETER_DEFAULT
# define MY_NEW_PARAMETER_DEFAULT    100    // default value for my new parameter
#endif
```

向主执行代码添加参数的工作就完成了！添加到主代码中(并非库中)的参数就可以通过诸如 **g.my\_new\_parameter** 这样来使用。

## 2、向库中添加参数

同样可以使用下列步骤向库中添加新的参数。以 [AP\\_Compass](#) 库为例：

### 第一步：

#### Step #1：

首先在库代码的.h 头文件添加新的变量(如 [Compass.h](#))。可使用的类型包括 AP\_Int8, AP\_Int16, AP\_Float, AP\_Int32, AP\_Vector3f。然后添加你的参数的默认值（我们将在 Step #2 中使用）。

```
#define MY_NEW_PARAM_DEFAULT    100

class Compass
{
public:
    int16_t product_id;          ///< product id
    int16_t mag_x;               ///< magnetic field strength along the X axis
    int16_t mag_y;               ///< magnetic field strength along the Y axis
    int16_t mag_z;               ///< magnetic field strength along the Z axis
    uint32_t last_update;        ///< micros() time of last update
    bool healthy;                ///< true if last read OK

    ///< Constructor
    Compass();

protected:
```

```

    AP_Int8 _orientation;
    AP_Vector3f _offset;
    AP_Float _declination;
    AP_Int8 _use_for_yaw;          ///AP_Int16 _my_new_lib_parameter;    ///< description of my new parameter
};

```

## 第二步：

### Step #2：

然后在.cpp 文件（如 Compass.cpp）中添加变量包含有@Param ~ @Increment 的 var\_info 表信息，以便允许 GCS（地面站）向用户显示来自地面站的关于该参数值的范围设定。当添加新参数时应注意：

- 自己添加的代码编号（下面的编号 9）一定要比之前变量的大。
- 参数的名称（如 MY\_NEW\_P）包括对象自动添加的前缀要少于 16 个字符。比如罗盘对象的前缀为“COMPASS\_”。

```

const AP_Param::GroupInfo Compass::var_info[] PROGMEM = {
    // index 0 was used for the old orientation matrix

    // @Param: OFS_X
    // @DisplayName: Compass offsets on the X axis
    // @Description: Offset to be added to the compass x-axis values to compensate for metal in the frame
    // @Range: -400 400
    // @Increment: 1

    <snip>

    // @Param: ORIENT
    // @DisplayName: Compass orientation
    // @Description: The orientation of the compass relative to the autopilot board.
    // @Values: 0:None,1:Yaw45,2:Yaw90,3:Yaw135,4:Yaw180,5:Yaw225,6:Yaw270,7:Yaw315,8:Roll180
    AP_GROUPINFO("ORIENT", 8, Compass, _orientation, ROTATION_NONE),

    // @Param: MY_NEW_P
    // @DisplayName: My New Library Parameter
    // @Description: The new library parameter description goes here
    // @Range: -32768 32767
    // @User: Advanced

    AP_GROUPINFO("MY_NEW_P", 9, Compass, _my_new_lib_parameter, MY_NEW_PARAM_DEFAULT),

    AP_GROUPEND
};

```



这样，新添加的参数将以 `_my_new_lib_parameter` 包含在库中。需要指明的是：protected 保护类型的参数是不能够在类外被访问的。如果我们将其变为 public 类型，那么我们就可以在主代码中使用 `compass._my_new_lib_parameter` 参数了。

### 第三步：

#### Step #3:

前面提到的是在已经存在的类（比如 AP\_Compass）中定义一个新的变量。如果你重新定义了一个新类，在这个新类中添加参数。添加参数的方法如第二步。不过你还有一个工作要做，就是将这个新类，添加到 [Parameters.pde](#) 文件的 `var_info` 数组列表中去。下面加粗的代码就是一个示例。

```
const AP_Param::Info var_info[] PROGMEM = {
    // @Param: SYSID_SW_MREV
    // @DisplayName: Eeprom format version number
    // @Description: This value is incremented when changes are made to the eeprom format
    // @User: Advanced
    GSCALAR(format_version, "SYSID_SW_MREV", 0),

<snip>

    // @Group: COMPASS_
    // @Path: ../libraries/AP_Compass/Compass.cpp
    GOBJECT(compass, "COMPASS_", Compass),

<snip>

    // @Group: INS_
    // @Path: ../libraries/AP_InertialSensor/AP_InertialSensor.cpp
    GOBJECT(ins, "INS_", AP_InertialSensor),

    AP_VAREND
};
```

## 3.4 添加新的飞行模式

英文参考：<http://dev.ardupilot.com/wiki/apmcopter-adding-a-new-flight-mode/>

本节源自：<http://liung.github.io/blog/apm/2014-09-05-APM-ArduCopter-添加新的飞行模式.html>

这部分将涵盖一些怎样创建一个新的高级别的飞行模式的基本操作步骤（类似于自稳，悬停等），这些新模式处于“the onion”（洋葱头工程）中的高级别代码控制部分，如之前[姿态控制页面](#)描述的一样。不过遗憾的是本页面并没有提供给你关于所创建的理想飞行模式需要的所有信息，但是希望这将是一个好的开始。

**Step #1：**在文件 [defines.h](#) 中用 `#define` 定义你自己新的飞行模式，然后将飞行模式数量 **NUM\_MODES** 加 1。

```
// Auto Pilot modes
// -----

#define STABILIZE 0 // hold level position
#define ACRO 1 // rate control
#define ALT_HOLD 2 // AUTO control
#define AUTO 3 // AUTO control
#define GUIDED 4 // AUTO control
#define LOITER 5 // Hold a single location
#define RTL 6 // AUTO control
```



```

#define CIRCLE 7 // AUTO control
#define LAND 9 // AUTO control
#define OF_LOITER 10 // Hold a single location using optical flow sensor
#define DRIFT 11 // DRIFT mode (Note: 12 is no longer used)
#define SPORT 13 // earth frame rate control
#define FLIP 14 // flip the vehicle on the roll axis
#define AUTOTUNE 15 // autotune the vehicle's roll and pitch gains
#define POSHOLD 16 // position hold with manual override
#define NEWFLIGHTMODE 17 // new flight mode description
#define NUM_MODES 18

```

**Step #2 :** 类似于相似的飞行模式的 [control\\_stabilize.pde](#) 或者 [control\\_loiter.pde](#) 文件，创建新的飞行模式的<new flight mode>.pde 控制 sketch 文件。该文件中必须包含一个 `_init()` 初始化函数和 `_run()` 运行函数，类似于 static bool `althold_init`(bool ignore\_checks) 和 static void `althold_run()`

```

/// -*- tab-width: 4; Mode: C++; c-basic-offset: 4; indent-tabs-mode: nil -*-
/*
 * control_newflightmode.pde - init and run calls for new flight mode
 */
// newflightmode_init - initialise flight mode
static bool newflightmode_init(bool ignore_checks)
{
    // do any required initialisation of the flight mode here
    // this code will be called whenever the operator switches into this mode

    // return true initialisation is successful, false if it fails
    // if false is returned here the vehicle will remain in the previous flight mode
    return true;
}

// newflightmode_run - runs the main controller
// will be called at 100hz or more
static void newflightmode_run()
{
    // if not armed or throttle at zero, set throttle to zero and exit immediately
    if(!motors.armed() || g.rc_3.control_in <= 0) {
        attitude_control.relax_bf_rate_controller();
        attitude_control.set_yaw_target_to_current_heading();
        attitude_control.set_throttle_out(0, false);
        return;
    }

    // convert pilot input into desired vehicle angles or rotation rates
    // g.rc_1.control_in : pilots roll input in the range -4500 ~ 4500
    // g.rc_2.control_in : pilot pitch input in the range -4500 ~ 4500

```

```

    // g.rc_3.control_in : pilot's throttle input in the range 0 ~ 1000
    // g.rc_4.control_in : pilot's yaw input in the range -4500 ~ 4500

    // call one of attitude controller's attitude control functions like
    // attitude_control.angle_ef_roll_pitch_rate_yaw(roll angle, pitch angle, yaw rate);

    // call position controller's z-axis controller or simply pass through throttle
    // attitude_control.set_throttle_out(desired throttle, true);
}

```

**Step #3 :** 在文件 [flight\\_mode.pde](#) 文件的 **set\_mode()**函数中增加一个新飞行模式的 case ( C++中 switch..case 语法 ) 选项 , 然后调用上面的\_init()函数。

```

// set_mode - change flight mode and perform any necessary initialisation
static bool set_mode(uint8_t mode)
{
    // boolean to record if flight mode could be set
    bool success = false;
    bool ignore_checks = !motors.armed(); // allow switching to any mode if disarmed. We rely on the arming check to perform

    // return immediately if we are already in the desired mode
    if (mode == control_mode) {
        return true;
    }

    switch(mode) {
        case ACRO:
            #if FRAME_CONFIG == HELI_FRAME
                success = heli_acro_init(ignore_checks);
            #else
                success = acro_init(ignore_checks);
            #endif
            break;

        case NEWFLIGHTMODE:
            success = newflightmode_init(ignore_checks);
            break;

    }
}

```

**Step #4:**在文件 [flight\\_mode.pde](#) 文件的 **update\_flight\_mode()**函数中增加一个新飞行模式的 case 选项 然后调用上面的\_run()函数。

```

// update_flight_mode - calls the appropriate attitude controllers based on flight mode
// called at 100hz or more
static void update_flight_mode()
{
    switch (control_mode) {
        case ACRO:

```

```

        #if FRAME_CONFIG == HELI_FRAME
            heli_acro_run();
        #else
            acro_run();
        #endif
        break;

```

```

        case NEWFLIGHTMODE:
            success = newflightmode_run();
            break;

```

```

    }
}

```

**Step #5:** 在文件 [flight\\_mode.pde](#) 文件的 `print_flight_mode()` 函数中增加可以输出新飞行模式字符串的 case 选项。

```

static void
print_flight_mode(AP_HAL::BetterStream *port, uint8_t mode)
{
    switch (mode) {
        case STABILIZE:
            port->print_P(PSTR("STABILIZE"));
            break;

        case NEWFLIGHTMODE:
            port->print_P(PSTR("NEWFLIGHTMODE"));
            break;

```

**Step #6 :** 在文件 [Parameters.pde](#) 中向 `FLTMODE1 ~ FLTMODE6` 参数中正确的增加你的新飞行模式到 `@Values` 列表中。

```

// @Param: FLTMODE1
// @DisplayName: Flight Mode 1
// @Description: Flight mode when Channel 5 pwm is 1230, <= 1360
// @Values:

```

0:Stabilize,1:Acro,2:AltHold,3:Auto,4:Guided,5:Loiter,6:RTL,7:Circle,8:Position,9:Land,10:OF\_Loiter,11:ToyA,12:ToyM,13:Sport,[17:NewFlightMode](#)

```

// @User: Standard
GSCALAR(flight_mode1, "FLTMODE1", FLIGHT_MODE_1),

```

```

// @Param: FLTMODE2
// @DisplayName: Flight Mode 2
// @Description: Flight mode when Channel 5 pwm is >1230, <= 1360
// @Values:

```

0:Stabilize,1:Acro,2:AltHold,3:Auto,4:Guided,5:Loiter,6:RTL,7:Circle,8:Position,9:Land,10:OF\_Loiter,11:ToyA,12:ToyM,13:Sport,[17:NewFlightMode](#)

```

// @User: Standard
GSCALAR(flight_mode2, "FLTMODE2", FLIGHT_MODE_2),

```

**Step #7** 如果你想让你的新飞行模式出现的 Mission Planner 的平视显示器 HUD 和飞行模式组件中 ,你需要相应修改 Mission Planner 代码。关于 Mission Planner 如何编译的问题，请参考我的另外一篇文章：[http://blog.sina.com.cn/s/blog\\_402c071e0102v4kx.html](http://blog.sina.com.cn/s/blog_402c071e0102v4kx.html)。



### 3.5 调用代码，使之定时运行

英文参考：<http://dev.ardupilot.com/wiki/code-overview-scheduling-your-new-code-to-run-intermittently/>

本节源自：<http://liung.github.io/blog/apm/2014-09-05-APM-ArduCopter-规划新代码使之按一定频率运行.html>

本页面将向你介绍如何规划你的新代码块使之可以按需运行。

#### 1、用代码调度器（scheduler）运行你的代码

在给定时间间隔内来运行你的代码的最灵活的方式就是使用调度器。这可以通过将你的函数添加到文件 [ArduCopter.pde](#) 中的 `scheduler_tasks` 数组来完成。需要表明的是：实际上该文件中有两个任务列表，上面的任务列表是针对高频 CPUs（如 Pixhawk），对应的调度频率是 400Hz，下面的是针对低频 CPUs（如 APM2），对应的调度频率是 100Hz。

添加一个任务是相当的简单，你只要在列表添加新的一行代码就可以了（列表中位置越靠前意味着拥有更高的级别）。任务项中的第一列代表了函数名，第二列是以 2.5ms 为单位的数字（或者 APM2 中以 10ms 为单位），所以，如果你想要你的函数执行频率为 400Hz，那么该列就需要填写为“1”，如果想要 50Hz，那么就需要改为“8”。任务项的最后一列代表该函数预计运行花费的微秒（百万分之一秒）时间。这可以帮助调度器来预估在下一个主循环开始之前是否有足够的时间来运行你的函数。

```
/*
 scheduler table - all regular tasks apart from the fast_loop()
 should be listed here, along with how often they should be called
 (in 10ms units) and the maximum time they are expected to take (in
 microseconds)
 */
static const AP_Scheduler::Task scheduler_tasks[] PROGMEM = {
    { update_GPS,          2,    900 },
    { update_nav_mode,     1,    400 },
```

```

{ medium_loop,          2,    700 },
{ update_altitude,     10,   1000 },
{ fifty_hz_loop,       2,    950 },
{ run_nav_updates,     10,    800 },
{ slow_loop,           10,    500 },
{ gcs_check_input,     2,    700 },
{ gcs_send_heartbeat,  100,   700 },
{ gcs_data_stream_send, 2,   1500 },
{ gcs_send_deferred,    2,   1200 },
{ compass_accumulate,  2,    700 },
{ barometer_accumulate, 2,    900 },
{ super_slow_loop,     100,  1100 },
{ my_new_function,     10,   200 },
{ perf_update,         1000,  500 }
};

```

## 2、作为循环的一部分运行你的代码

为了代替在代码调度器中加入一个新的函数入口，你还可以在现有的任何时间循环事件中添加你的函数。除了在 fast-loop 循环中添加外，这种方法对比起上面的代码调度器方法并没有什么实质性好处。但当你的代码添加到 fast-loop 循环中时，就意味着它将以最高的优先级来执行（它几乎能 100% 达到所确保的 400hz 运行速度）。

- [fast\\_loop](#): APM2 上运行频率 100hz，Pixhawk 上 400Hz
- [fifty\\_hz\\_loop](#): 运行频率 50hz
- [ten\\_hz\\_logging\\_loop](#): 运行频率 10hz
- [three\\_hz\\_loop](#): 运行频率 3.3hz
- [one\\_hz\\_loop](#): 运行频率 1hz

所以举个例子，如果你想让你的代码运行频率为 10hz，那么你就要将它添加到 [ArduCopter.pde](#) 文件的 `ten_hz_logging_loop()` 函数声明中。

```

// ten_hz_logging_loop
// should be run at 10hz
static void ten_hz_logging_loop()
{
    if (g.log_bitmask & MASK_LOG_ATTITUDE_MED) {
        Log_Write_Attitude();
    }
    if (g.log_bitmask & MASK_LOG_RCIN) {
        DataFlash.Log_Write_RCIN();
    }
    if (g.log_bitmask & MASK_LOG_RCOUT) {
        DataFlash.Log_Write_RCOUT();
    }
    if ((g.log_bitmask & MASK_LOG_NTUN) && mode_requires_GPS(control_mode)) {
        Log_Write_Nav_Tuning();
    }
    // your new function call here
    my_new_function();
}

```

## 3.6 增加新的 MAVLink 消息

英文参考：<http://dev.ardupilot.com/wiki/code-overview-adding-a-new-mavlink-message/>

本节源自：<http://liung.github.io/blog/apm/2014-09-05-APM-增加新的 MAVLink 通讯协议消息.html>

MAVLink 协议：<https://pixhawk.ethz.ch/mavlink/>

地面站之间的数据和指令通信都是通过串行接口使用 [MAVLink 协议](#)来传递的。本页面将提供关于添加新的 MAVLink 信息的一些高级建议。

这些指令仅在 Linux 上测试完成（准确的说，是在 Windows 上运行的 Ubuntu 虚拟机上测试完成的）。关于设置虚拟机的方法在[SITL\(软件层面仿真\)页面](#)有相关介绍。如果你要运行 SITL，你最好遵循下面的一些建议。这些指令不能直接在 Windows 或者 Mac 平台上本地运行。

**Step #1：**确保你已经安装了最新的 ardupilot 代码，同时也检查一下 mavproxy 是否是最新版本。mavproxy 工具可以通过在终端窗口运行下面命令进行升级。

```
sudo pip install --upgrade mavproxy
```

**Step #2：**先确定你所要添加的信息的类型，以及如何和已有的 MAVLink 消息兼容。

比如：你可能会想要向飞行器发送一个新的导航指令，让它可以在任务中期（自动模式中）模仿一个特技动作（比如翻筋斗）。在这个例子中，你需要一个类似于 **MAV\_CMD\_NAV\_WAYPOINT**（可以在 MAVLink 消息页面搜索 MAV\_CMD\_NAV\_WAYPOINT）一样的新的导航指令 **MAV\_CMD\_NAV\_TRICK**。

又或者你想要从飞行器发送一个新的传感器数据类型到地面站，可能类似于 SCALED\_PRESSURE 消息。

**Step #3：**在 [common.xml](#) 和 [ardupilotmega.xml](#) 文件中添加你的信息的定义声明。

如果你希望将该指令添加到 MAVLink 协议中，那么你应该添加该指令到 `./ardupilot/libraries/GCS_MAVLink/message_definitions/common.xml` 文件中。如果你仅仅个人使用或者仅仅和 ArduCopter ArduPlane，ArduRover 搭配使用，那么它就应该被添加到 `ardupilotmega.xml` 文件中。

**Step #4：**重新生成你的所有 include 文件，确保添加的信息在主代码中可以被识别。

首先将目录切换到 ardupilot 文件夹下，然后执行下面命令：

```
./libraries/GCS_MAVLink/generate.sh
```

成功执行后，你应该看到下面这些文件都应该被更新。

```
./libraries/GCS_MAVLink/include/mavlink/v1.0/ardupilotmega/ardupilotmega.h
./libraries/GCS_MAVLink/include/mavlink/v1.0/ardupilotmega/version.h
./libraries/GCS_MAVLink/include/mavlink/v1.0/common/version.h
```

文件 version.h 仅简单的更新了文件的日期和时间，但是 **ardupilotmega.h** 文件已经应该有了你的新消息的定义声明。

**Step #5：**在飞行器主代码中添加函数方法用来控制向/从地面站发送/接收指令。

这些顶层代码指令绝大部分包含在飞行器的 [GCS\\_MAVLink.pde](#) 文件中或在 [./libraries/GCS\\_MAVLink/GCS](#) 类中。

在我们想要添加一个新的导航指令的例子中（比如执行特技动作），应该需要下面信息：

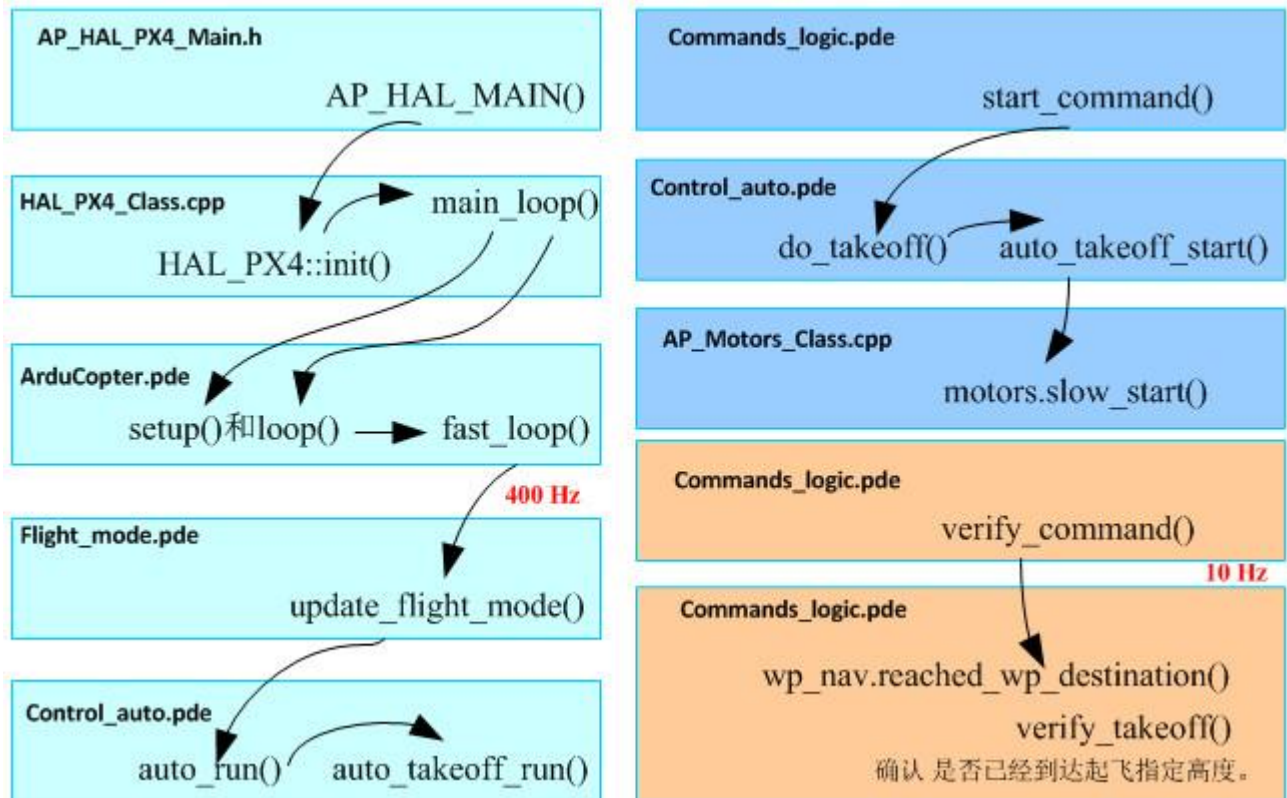
- 扩展 [AP\\_Mission](#) 库中的 `mission_cmd_to_mavlink()`和 `mavlink_to_mission_cmd()`方法，将 mavproxy 的指定转换到一个 **AP\_Mission::Mission\_Command** 结构体中。

```
// 将 Mission_Command 对象 转换到 mavlink 消息，该消息能够被发送到 GCS 地面站。
bool AP_Mission::mission_cmd_to_mavlink(const AP_Mission::Mission_Command& cmd, mavlink_mission_item_t& packet)
{... ...}

// 将 mavlink 消息 转换到 Mission_Command 对象，该对象可以被存储到 eeprom
bool AP_Mission::mavlink_to_mission_cmd(const mavlink_mission_item_t& packet, AP_Mission::Mission_Command& cmd)
{... ...}
```



- 在飞行器的 `commands_logic.pde` 文件中分别添加 `start_command()` 函数和 `verify_command()` 函数的一个 case 分支 ,用来校验新的消息指令 `MAV_CMD_NAV_TRICK` 是否接收到。这些需要你调用自己创建的两个新函数 `do_trick()` 和 `verify_trick()` ( 具体参考下面 )。
- 创建两个新函数 `do_trick()` 和 `verify_trick()` , 用来控制飞行器如何执行特技动作 ( 这可能需要调用 `control_auto.pde` 中的另一个函数来设置 `auto_mode` 变量 然后调用新方法 `auto_trick_start()` )。当指令第一次被唤醒时将使用 `do_trick()` 函数。`verify_trick()` 函数将会以 10hz 频率( 或者更高 )被重复调用直到特技动作完成 ,当特技动作执行完毕之后 `verify_trick()` 函数应该返回 `True`。



## 4 编译代码

如何编译代码：

参考：<http://dev.ardupilot.com/wiki/building-the-code/>

参考：<http://dev.ardupilot.com/wiki/building-px4-with-make/> ( 直接进入 Pixhawk 编译 )

参考：<http://liung.github.io/blog/apm/2014-09-06-APM-编译代码.html>

APM:Plane,Copter,Rover:

Windows 用户：

- [Windows 平台上用 Arduino 编译 ArduPilot](#)
- [Windows 平台上用 Make 方法编译 Pixhawk/PX4](#)
- [使用 Atmel Studio 或者 Visual Studio 编译 APM \( 推荐\\*\\*\\*\\* \)](#)

MacOS 用户：

- [MacOS 平台上用 Arduino 编译 APM2.x](#)
- [MacOS 平台上用 Make 方法编译 Pixhawk/PX4](#)

Linux 用户：

- [Linux 平台上用 Make 方法编译 APM2.x](#)
- [Linux 平台上用 Make 方法编译 Pixhawk/PX4](#)
- [Linux 平台上用 Make 方法编译 BeagleBone Black](#)

地面站 Mission Planner :

- [Windows 平台上使用 Visual Studio 编译 Mission Planner](#)

## 4.1 APM-Windows 平台上用 Make 方法编译 Pixhawk 和 PX4

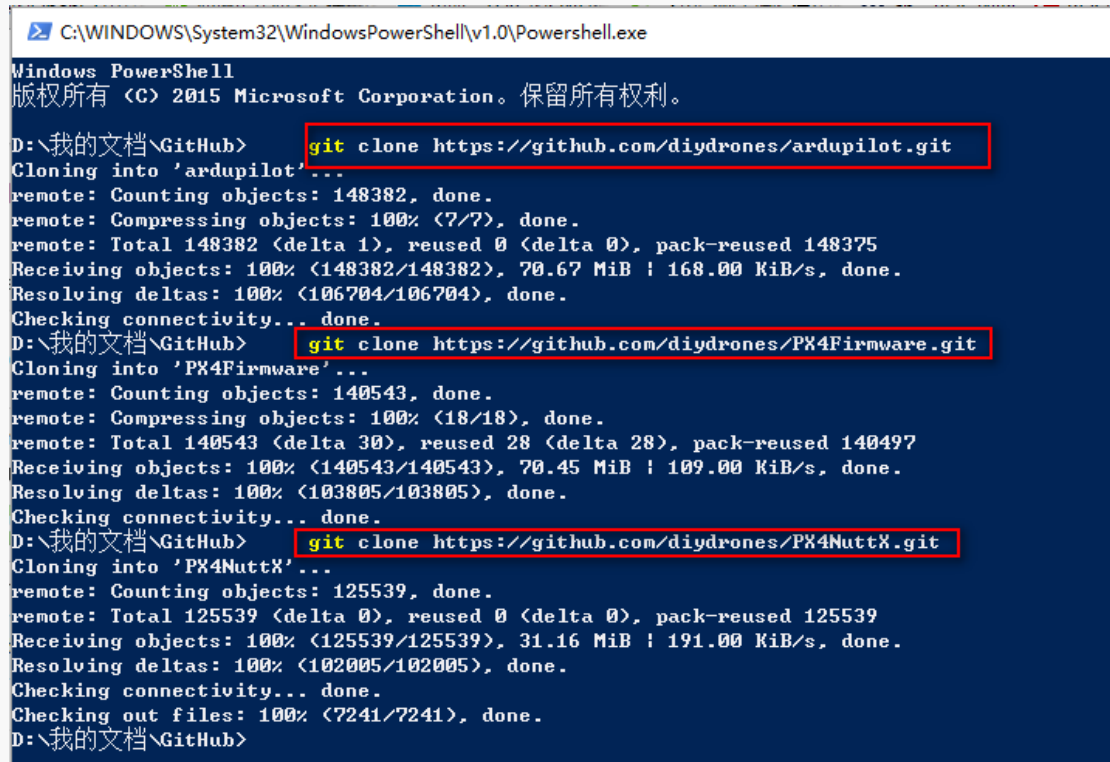
<http://liung.github.io/blog/apm/2014-09-07-APM-Windows 平台上用 Make 方法编译 Pixhawk 和 PX4.html>

<http://dev.ardupilot.com/wiki/building-px4-with-make/>

### 1 Windows 平台下的 Make 方法

- 1、安装 [Github for Windows](#) , 或者 <https://desktop.github.com/>
- 2、确保你的 github 上关于行尾结束符的设置没有发生变动。
  - 当你安装了 Git 后, “Git shell( or Bash )” 也会安装好。点击 “Git shell( or Bash )” 图标, 然后在弹出的 Git “MINGW32” 终端窗口键入:  
`git config --global core.autocrlf false`
- 3、“克隆” ( clone ) Ardupilot, PX4Firmware 和 PX4NuttX 代码库到你的本地电脑上:
  - 到 [GitHub/diydrones/ardupilot](#) 的 web 页面点击 “Clone in Desktop” 按钮
  - 到 [GitHub/diydrones/PX4Firmware](#) 的 web 页面点击 “Clone in Desktop” 按钮
  - 到 [GitHub/diydrones/PX4NuttX](#) 的 web 页面点击 “Clone in Desktop” 按钮
  - 到 [GitHub/diydrones/uavcan](#) 的 web 页面点击 “Clone in Desktop”按钮

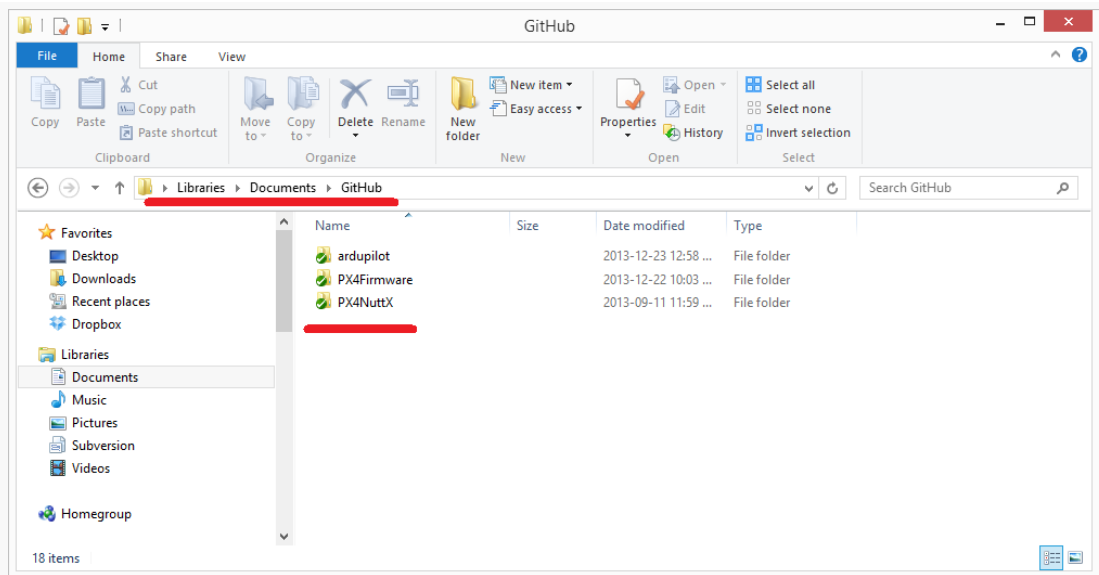
如果以上链接没有 “Clone in Desktop”按钮, 可以使用 git shell 工具, 通过命令行下载, 命令如下:



```
C:\WINDOWS\System32\WindowsPowerShell\v1.0\Powershell.exe
Windows PowerShell
版权所有 (C) 2015 Microsoft Corporation。保留所有权利。

D:\我的文档\GitHub> git clone https://github.com/diydrones/ardupilot.git
Cloning into 'ardupilot'...
remote: Counting objects: 148382, done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 148382 (delta 1), reused 0 (delta 0), pack-reused 148375
Receiving objects: 100% (148382/148382), 70.67 MiB | 168.00 KiB/s, done.
Resolving deltas: 100% (106704/106704), done.
Checking connectivity... done.
D:\我的文档\GitHub> git clone https://github.com/diydrones/PX4Firmware.git
Cloning into 'PX4Firmware'...
remote: Counting objects: 140543, done.
remote: Compressing objects: 100% (18/18), done.
remote: Total 140543 (delta 30), reused 28 (delta 28), pack-reused 140497
Receiving objects: 100% (140543/140543), 70.45 MiB | 109.00 KiB/s, done.
Resolving deltas: 100% (103805/103805), done.
Checking connectivity... done.
D:\我的文档\GitHub> git clone https://github.com/diydrones/PX4NuttX.git
Cloning into 'PX4NuttX'...
remote: Counting objects: 125539, done.
remote: Total 125539 (delta 0), reused 0 (delta 0), pack-reused 125539
Receiving objects: 100% (125539/125539), 31.16 MiB | 191.00 KiB/s, done.
Resolving deltas: 100% (102005/102005), done.
Checking connectivity... done.
Checking out files: 100% (7241/7241), done.
D:\我的文档\GitHub>
```





#### 4、初始化并更新子模块

对于多旋翼，进入下载的源码的 ardupilot/ArduCopter 目录，在 git shell 中输入如下命令：

```
git submodule update --init --recursive
```

结果如下：

```
D:\我的文档\GitHub> git config --global core.autocrlf false
D:\我的文档\GitHub> git submodule update --init --recursive
fatal: Not a git repository (or any of the parent directories): .git
D:\我的文档\GitHub> cd D:\GitHub\githubnew\ardupilot\ArduCopter
D:\GitHub\githubnew\ardupilot\ArduCopter [master]> git submodule update --init --recursive
Submodule 'modules/PX4Firmware' (git://github.com/diydrones/PX4Firmware.git) registered for path 'modules/PX4Firmware'
Submodule 'modules/PX4NuttX' (git://github.com/diydrones/PX4NuttX.git) registered for path 'modules/PX4NuttX'
Submodule 'modules/gbenchmark' (git://github.com/google/benchmark.git) registered for path 'modules/gbenchmark'
Submodule 'gtest' (git://github.com/diydrones/googletest) registered for path 'modules/gtest'
Submodule 'modules/mavlink' (git://github.com/diydrones/mavlink) registered for path 'modules/mavlink'
Submodule 'modules/uavcan' (git://github.com/diydrones/uavcan.git) registered for path 'modules/uavcan'
```

注意：git submodule update --init --recursive，这个命令需要分别在 ardupilot/ArduCopter，PX4Firmware，uavcan 等路径下分别运行一次。

#### 5、下载 [px4\\_toolchain\\_installer\\_v14\\_win.exe](http://pixhawk.org/dev/toolchain/installation) 并安装 PX4 工具链（在该页面的底部搜索“PX4 Toolchain Installer”）

PX4 下载地址

地址：<http://pixhawk.org/dev/toolchain/installation>

地址：<http://dev.px4.io/starting-installing-windows.html>

#### 6、打开 PX4Console

- win7 平台上可以在开始>>所有程序>>PX4 Toolchain 找到 PX4Console，并运行。或者直接去该目录 C:\px4\toolchain\msys\1.0\px4\_console.bat 运行。
- “cd”到步骤三中 Ardupilot 固件的文件目录，然后进入 ArduCopter 目录（确保这里 ArduCopter 中的 A 和 C 都是大写形式）。在命令行中可以这样执行：

```
cd /c/Users/<username>/Documents/GitHub/ardupilot/ArduCopter
```

#### 7、打开 PX4 控制台开始编译固件，首先切换到 ArduCopter 目录，然后输入下面命令执行编译

- make px4 <- 将会编译四旋翼形式的 PX4 和 PixHawk 固件
- make px4-v2 <- 将会编译四旋翼形式的 PixHawk 固件

- make px4-v2-hexa <- 将会编译六旋翼形式的 Pixhawk 固件(其它可支持的后缀名包括“quad”、“octa”和“heli”)
- make clean <- 清理 (“clean”) ardupilot 文件目录
- make px4-clean <- 清理 (“clean”) PX4Firmware 和 PX4NuttX 文件目录以便于下次重新编译
- make px4-v2-upload <- 编译并加载 Pixhawk 的四旋翼固件(如果使用该命令，那么就不需要执行下面的步骤七了)

编译好的固件以 .px4 文件扩展名结尾，位于在 ArduCopter 目录：

```
LINK: /d/github/githubnew/ardupilot/modules/PX4Firmware/Build/px4fmu-v2_APM.build/firmware.elf
BIN: /d/github/githubnew/ardupilot/modules/PX4Firmware/Build/px4fmu-v2_APM.build/firmware.bin
%% Generating /d/github/githubnew/ardupilot/modules/PX4Firmware/Build/px4fmu-v2_APM.build/firmware.px4
make[2]: Leaving directory /d/github/githubnew/ardupilot/modules/PX4Firmware/Build/px4fmu-v2_APM.build
%% Copying /d/github/githubnew/ardupilot/modules/PX4Firmware/Images/px4fmu-v2_APM.px4
make[1]: Leaving directory /d/github/githubnew/ardupilot
text data bss dec hex filename
997068 2820 57312 1057200 1021b0 d:/github/githubnew/ardupilot/modules/PX4Firmware/Build/px4fmu-v2_APM.build/firmware.elf
fatal: Not a git repository: d:/github/githubnew/ardupilot/modules/../../.git/modules/modules/PX4Firmware
Failed to get px4 hash
fatal: Not a git repository: d:/github/githubnew/ardupilot/modules/PX4NuttX/nuttX/../../.git/modules/modules/PX4NuttX
Failed to get nuttx hash
PX4 ArduCopter Firmware is in ArduCopter-v2.px4
Administrator@dengkan /d/github/githubnew/ardupilot/ArduCopter
$
```

7、使用 Mission Planner 加载固件的方法：初始设置( Initial Setup )>>安装固件( Install Firmware )界面点击 “Load custom firmware” 链接。

生成的文件如下：

t (D:) > GitHub > githubnew > ardupilot > modules > PX4Firmware > Images				
名称	修改日期	类型	大小	
aerocore.prototype	2016/3/21 星期...	PROTOTYPE 文件	1 KB	
px4fmu-v1.prototype	2016/3/21 星期...	PROTOTYPE 文件	1 KB	
px4fmu-v2.prototype	2016/3/21 星期...	PROTOTYPE 文件	1 KB	
px4fmu-v2_APM.bin	2016/3/21 星期...	BIN 文件	977 KB	
px4fmu-v2_APM.px4	2016/3/21 星期...	PX4 文件	872 KB	
px4fmu-v4.prototype	2016/3/21 星期...	PROTOTYPE 文件	1 KB	
px4io-v1.prototype	2016/3/21 星期...	PROTOTYPE 文件	1 KB	
px4iov2.prototype	2016/3/21 星期...	PROTOTYPE 文件	1 KB	
px4io-v2.prototype	2016/3/21 星期...	PROTOTYPE 文件	1 KB	
px4io-v2_default.bin	2016/3/21 星期...	BIN 文件	52 KB	
px4io-v2_default.px4	2016/3/21 星期...	PX4 文件	50 KB	
px4-stm32f4discovery.prototype	2016/3/21 星期...	PROTOTYPE 文件	1 KB	
ArduCopter-v2.px4	2016/3/21 星期...	PX4 文件	872 KB	
firmware.px4	2016/3/21 星期...	PX4 文件	50 KB	
firmware.px4	2016/3/21 星期...	PX4 文件	872 KB	
px4fmu-v2_APM.px4	2016/3/21 星期...	PX4 文件	872 KB	
px4io-v2_default.px4	2016/3/21 星期...	PX4 文件	50 KB	

Pixhawk

px4io(STM32F103)固件：px4io-v2\_default.px4

px4FMU(STM32F427)固件：ArduCopter-v2.px4






有关 bootloader:

bootloader 的下载地址：<https://github.com/PX4/Bootloader>

make px4fmu-v2\_bl px4fmu 编译

make px4io\_bl      px4io 编译

生成的文件如下：

 px4io_bl.elf	2016/3/21 星期...	ELF 文件	137 KB
 px4io_bl.bin	2016/3/21 星期...	BIN 文件	4 KB
 px4fmu4_bl.elf	2016/3/21 星期...	ELF 文件	178 KB
 px4fmu4_bl.bin	2016/3/21 星期...	BIN 文件	10 KB
 px4fmu2_bl.elf	2016/3/21 星期...	ELF 文件	178 KB
 px4fmu2_bl.bin	2016/3/21 星期...	BIN 文件	10 KB

## 2 加快编译时间的一些小建议

杀毒防护软件有可能会减慢编译时间，特别对 PX4 尤为明显，所以建议包含有 Ardupilot、PX4Firmware 和 PX4NuttX 源代码的文件夹不在你的杀毒软件实时扫描范围内。

当执行完 `make px4-clean` 后第一次编译将会非常的慢，因为要重新编译每一个文件。

## 5 下载固件

用 MissionPlanner 地面站软件下载固件。请参考《Pixhawk 学习指南 WalkAnt20160317》学习指南。（在 <http://blog.sina.com/antinformation> 中）