第一部分:

ArdurPilot 代码分为 5 个主要部分，基本结构分类如下:

1 vehicle directories
2 libraries
3 tool
4 mk
5 external support code 外部支持代码

其中 vehicle directories 为模型类型，当前共有 4 种模型，分别为:ArduPlane,ArduCopter, APMrover2 和 AntennaTracjer。都是.pde 文件，就是为了兼容 arduino 平台；

Libraries 中为主程序；

tool 为工具，只要提供支持；

mk 为 mavlink 通信协议，mavlink 是对微型飞行棋数据头信息进行引导的库，它可以高速的传递 c 语言并发送这些数据包到地面控制站，针对不同的硬件板，编译可以采用"make TARGET"的形式。如果要移植到新的硬件，可以在 mk/targets.mk 文件中添加，如:make px4-v2-j8

第二部分: 学习 sketch 例程代码

Sketch，是指使用.pde 文件编写的主程序

Sketch 代码的几个特点:

1、 pde 文件包含很多 includes；其中 include 为 pde 文件转变为 C++文件后，提供必要的库引用支持。

2、 定义了 hal 引用声明；在 Sketch 例程中 hal 引用声明定义如下:

Const AP_HAL::HAL&hal=AP_HAL_BOARD_DRIVER;//pixhawk 等价于 AP_HAL_PX4 这样定义方便访问硬件接口，比如 console 终端、定时器、I2C、SPI 接口等。

实际的定义实在 HAL_PX4_Class.cpp,如下:

const HAL_PX4 AP_HAL_PX4

hal 是针对 AP_HAL_PX4 的引用

3、 setup()和 loop()函数;每个 sketch 都有一个 setup()和 loop()函数。板子启动时，setup()调用。这些函数都由 HAL 代码中的 main()函数调用(HAL_PX4_Class.cpp 文件中的 main_loop())。Setup()函数只调用一次，用于初始化所有 libraries。

Loop()函数被调用，执行任务。

4、 AP_HAL_MAIN()宏指令；每一个 sketch 最底部，都有一个"AP_HAL_MAIN();"指令，它是一个 HAL 宏，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
程序的执行过程为：程序起点 AP_HAL_MAIN()->hal.init()->hal.main_loop()->sketch 中的 setup()和 loop()。
其中 loop()函数；usage()函数；init()函数定义在 HAL_PX4_Class.cpp 中

关于函数理解的几点补充:
main(int argc,char *argv[]
argc:整数，用来统计你运行程序时送给 main 函数的命令行参数的个数
*argv[]:字符串数组，用来存放指向你的字符串参数的指针数组，每一个元素指向一个参数
argv[0]:指向程序运行的全路径名
argv[1]:指向在 DOS 命令行中执行程序名后的第一个字符串
argv[2]:指向执行程序名后的第二个字符串

.....

argv[argc]为 NULL；
argc,argv,是在 main()函数之前被复制的，编译器生成的可执行文件，main()不是真正的；入口点，而是一个标准的函数，这个函数名与具体的操作系统有关；

第三部分: 串行接口 UART 和 Console

1、5 个 UART
目前定义了 5 个 UART，它们的用途分别是
（1） uartA:串行端口，通常是 Micro USB 接口，运行 MAVLink 协议。
（2） uartB:GPS1 模块。
（3） uartC:主数传接口，也就是 Pixhawk telem1 接口。
（4） uartD:次数传接口，也就是 telem2 接口。
（5） uartE:GPS2 模块。
2、调试终端
作为 5 个 UART 的补充，额外有一个调试终端，可以查看 AP_HAL/AP_HAL_Boards.h，如果定义了如果定义了 HAL_OS_POSIX_IO 说明这个平台提供调试终端。

第四部分:学习 RCInput and 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。Adrupilot 支持集中不同 RC Input(取决于具体的硬件飞控)
其中 Spektrum/DSM—on PX4,pixhawk and Linux;pwm—on apm1 and apm2
RC Output,是指飞控接受到 RC 输入后，再将其处理后，输出到伺服（使物体的位置、方位、状态等输出被控量能够跟随输入目标的任意变化，精度非常高）和电机上。RC Output 默认 50HZ pwm 信号。对于 AdruCopter 多轴飞行器和直升机，输出频率 400HZ。
1、 RCInput 对象(AP_HAL)

RCInput 对象声明：
AP_HAL::RCInput* rcin;

2、 RCOutput 对象（AP_HAL）
RCOutput 对象声明
AP_HAL::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（两线式串行总线，一条串行数据线 SDA，一条串行时钟线 SCL）、经由协议处理器（PX4IO）输出等程序。

3、 RC_Channel 对象
hal.rcin 和 h.rcout 对象，为低层次调用。最常用的是使用更高级封装的 RC_Channel 对象来实现 input 和 output。它允许用户对参数进行配置，例如每个通道的值 min/max/trim 值，同时支持辅助 AUX 通道函数，还可以对 input 和 output 进行比例缩放处理等

4、 RC_Channel_aux 对象
RC_Channel_anx class,它是 RC_Channel 的一个子类

第五部分:存储与 EEPROM 管理

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;

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。在 class AP_HAL::Storage 仅有 3 个函数:

- Init() 初始化存储系统；
- Read_block() 读块数据
- Write_block() 写块数据

（2） StorageManager 库(libraries/StorageManager)，是 hal.storage 更高级的封装；

StorageManage 库提供对存储区域”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:

- 参数区；
- 飞行区域限制点数据区；
- 航点数据区；
- 集结点数据区；

（3） DataFlash(libraries/StorageManager)用于日志存储；

主要实现日志存储，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。它能够非常有效的存储这些数据

（4） Posix Io 函数，是传统文件系统读写函数。

有些板子是带操作系统的，如 Linux 和 NuttX。AP_Terrain library 就是一个典型的例子。地形数据对于 EEPROM 是非常大的，经常需要随机存储。HAL_OS_POSIX_IO 的宏定义在 AP HAL Board.h 文件中。

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

第六部分：APM:Copter 程序库

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

第七部分：姿态控制预览：

这种有手动飞行模式，诸如自稳模式（Stabilize Mode）、特技模式（Acro Mode）等

程序的执行过程：

1、 flight_mode.pde 中的 update_flight_mode()函数被调用；

2、 进入 control_*.pde 飞行控制文件（比如：control_stabilize.pde，control_rtl.pde 等），执行*_run()函数（比如：自稳模式的 stabilize_run），_run 函数负责将用户的输入数据转换为此时飞行模式下的倾斜角、滚转速率、爬升率对应的数值等。

3、_run 执行后会将数据传到 AC_AttitiudeControl 文件夹中，来调整飞行器的姿态。

在 AC_AttitiudeControl 库中有最通用的三种方法调整飞行器的姿态：

1、angle_ef_roll_pitch_rate_ef_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_ef_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_bf_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。

在选择飞行器的姿态调整的方法后，调用其中的 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，再将这些输入发送给 AP_Motors 库，AP_Motors 库的代码负责将从 AC_AttitudeControl 和 AC_PosControl 库发送过来的滚转、偏航角度、俯仰数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要使用如下函数：

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

4、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，再调用 hal.rcout->write()，把期望 PWM 值传递给 AP_HAL 层，输出至飞控板对应的 PWM 端口（pin 端）。

第八部分：上面是利用自带的飞行模式，还可以自己加入所需要的飞行模式。下面是加入模式的步骤，在那个文件里面添加相应代码：

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.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 -*-
// 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

// @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

// @User: Standard

GSCALAR(flight_mode2, "FLTMODE2", FLIGHT_MODE_2),

第九部分：添加新的参数

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_filenumber,                // *** 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_arming_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种提到的参数。

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 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_Int8 _auto_declination;                      ///

//////////////////////////////////////////////////////////////////
    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 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



    // @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),

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

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

    AP_VAREND
};
```