神弓科技软件编程规范V1.0

版本	说明	修改日期	操作人
《神弓科技软件编程规范V1.0》	初始版本	2023-11-01	覃剑

这是一份神弓科技开发人员的开发指引。SG_Ardupilot 做为一份多人协作软件，它需要由不同的同事采用合作的方式完成，SG_Ardupilot 的开发人员请遵守此编程规范。

1.目录名称

目录名称如果无特殊的需求，请使用开头字母大小其余小写的形式；目录名称应能够反映部分的意思，例如各芯片移植由其芯片名称构成或芯片类别构成；Components 目录下能够反映组件的意义。

2.文件名称

文件名称如果无特殊的需求(如果是引用其他地方，可以保留相应的名称)，请使用全小写的形式。另外为了避免文件名重名的问题，一些地方请尽量不要使用通用化、使用频率高的名称。

设备驱动源码文件：`drv_class.c` 的命名方式，如：

- drv_spi.c
- drv_gpio.c

3.头文件定义

C 语言头文件为了避免多次重复包含，需要定义一个符号。这个符号的定义形式请采用如下的风格：

```
#ifndef __FILE_H__
#define __FILE_H__
/* 头文件内容 */
#endif
```

C++头文件形式：

```
#pragma once
/* 头文件内容 */
```

即定义的符号两侧采用 "_" *以避免重名，另外也可以根据文件名中是否包含多个词语而采用* "" 连接起来。

4.文件头注释

因考虑到代码闭源且只有内部使用，以前应对后续可能的代码第三方审核，建议使用中文注释，在每个源文件文件头上，应该包括相应的版权信息，文件说明，更改记录：

```
/*
 * 版权所有：神弓科技
 *
 * 文件说明：此文件为导引头驱动通信协议，采用RS422，数据格式为8N1波特率230400
 *
 * 更改记录：
 * 日期              作者          笔记
 * 2020-08-18      覃剑         第一个版本
 * 2021-10-09      覃剑         修改数据解析线程频率，从200Hz更改为500Hz
 */
```

例如采用如上的形式。

5.结构体定义

结构体名称请使用小写英文名的形式，单词与单词之间采用 "_" 连接，例如：

```
    struct sg_list_node
    {
        struct sg_list_node *next;
        struct sg_list_node *prev;
    };
```

其中，**"{"，"}" 独立占用一行**，后面的成员定义使用缩进的方式定义。

结构体等的类型定义请以结构体名称加上 "_t" 的形式作为名称，例如：

```
    typedef struct sg_list_node sg_list_t;
```

因为内核中对象引用方便的缘故，采用了对象内核指针作为类型定义的形式，例如：

```
    typedef struct sg_timer* sg_timer_t;
```

6.宏定义

请使用大写英文名称作为宏定义，单词之间使用 "_" 连接，例如：

```
    #define SG_TRUE                         1
```

7.函数名称、声明

函数名称请使用小写英文的形式，单词之间使用 "_" 连接。提供给上层应用使用的 API接口，必须在相应的头文件中声明；如果函数入口参数是空，必须使用 void 作为入口参数，例如：

```
sg_thread_t sg_thread_self(void);
```

内部静态函数命名：以下划线开头，使用 `_class_method` 格式，不携带`_sg_`开头，如内核或驱动文件中的函数命名：

```
/* IPC对象初始化 */
static sg_err_t _ipc_object_init(void)

/* UART驱动程序配置控制 */
static sg_err_t _uart_configure(void)
static sg_err_t _uart_control(void)
```

8.类名称定义

类的定义单词开头字母大写，其余小写形式，单词之间使用 "_" 连接，先写有公有函数，再到公用变量定义，接着到保护类型的变量及函数定义，然后私有函数，最后是私有变量定义例如：

```
class AC_PosControl
{
public:
    // 构造函数
    AC_PosControl(const AP_AHRS_View& ahrs, const AP_InertialNav& inav,
                  const AP_Motors& motors, AC_AttitudeControl&
attitude_control);

    // calc_leash_length-根据最大速度、加速度计算高度步长限制
    void calc_leash_length_z(void);

    //变量定义xxxx

protected:
    // 极限标志结构体
    struct poscontrol_limit_flags {
        uint8_t pos_up      : 1;    // 1 如果在上升时达到了高度步长限制
        uint8_t pos_down    : 1;    // 1 如果在下降时达到了高度步长限制
        uint8_t vel_up      : 1;    // 1 如果已经达到了上升的最大垂直速度限制
        uint8_t vel_down    : 1;    // 1 如果已经达到了下降的族弟啊垂直速度限制
        uint8_t accel_xy    : 1;    // 1 如果已经达到了水平最大加速度的限制
    } _limit;

    float       _speed_down_cms;        // 最大下降速度 cm/s
    float       _speed_up_cms;          // 最大上升速度 cm/s
    float       _speed_cms;             // 最大水平速度 cm/s

    // 高度控制器运行函数
    void run_z_controller(void);

private:
    // desired_vel_to_pos - 使用所需速度移动位置目标
    void desired_vel_to_pos(float nav_dt);

    Vector3f    _vel_target;            // 计算的速度目标,单位为cm/s
    Vector3f    _vel_error;             // 期望加速度和实际加速度之间的误差，单位为cm/s

}
```

9.注释编写

请使用**中文**做为注释，使用中文注释方便内部所有开发者成员更好的理解代码逻辑。

语句注释:

源代码的注释不应该过多，更多的说明应该是代码做了什么，仅当个别关键点才需要一些相应提示性的注释以解释一段复杂的算法它是如何工作的。对语句的注释只能写在它的**上方或右方**，其他位置都是非法的。

```
/* 你的中文注释 */
```

函数注释:

注释以 `/**` 开头，以 `*/` 结尾，中间写入函数注释，组成元素如下，每个元素描述之间空一行，且首列对齐:

- @简述 + 简述函数作用。在描述中，着重说明该函数的作用。
- @函数说明+ 函数说明。在上述简述中未能体现到的函数功能或作用的一些点，可以做解释说明。
- @举例+ 相关 API 罗列。若有与当前函数相关度较高的 API，可以进行列举。
- @参数+ 以参数为主语 + be 动词 + 描述，说明参数的意义或来源。
- @返回值+ 枚举返回值 + 返回值的意思，若返回值为数据，则直接介绍数据的功能。
- @注意事项+ 函数使用注意要点。在函数使用时，描述需要注意的事项，如使用环境、使用方式等。

```
/**
 * @简述      该函数将初始化一个静态事件对象。
 *
 * @说明       对于静态事件对象，其内存空间由编译器在编译期间分配，并应放置在读写数据段上或未初始化的数据段上，
 *             相比之下，rt_event_create（）函数将自动分配内存空间并初始化该事件。
 *
 * @例子       rt_event_create()
 *
 * @参数      event是指向要初始化的事件的指针。假设事件的存储将在您的应用程序中分配。
 *
 * @参数      name是指向事件名称的指针。
 *
 * @参数      flag是事件标志，用于确定多个线程等待的排队方式，
 *             RT_IPC_FLAG_PRIO挂起的线程将按优先级顺序排队，
 *             RT_IPC_FLAG_FIFO挂起的线程将在先进先出方法中排队（也称为先到先得（FCFS）调度策略）。
 * @返回值     返回操作状态。当返回值为RT_EOK时，初始化成功，如果返回值是任何其他值，则表示初始化失败。
 *
 * @注意事项   此函数只能从线程中调用。
 */
sg_err_t sg_event_init(rt_event_t event, const char *name, rt_uint8_t flag)
{
    ...
}
```

10.缩进及分行

缩进请采用 4 个空格的方式。如果没有什么特殊意义，请在 "{" 后进行分行，并在下一行都采用缩进的方式，例如：

```
    if (condition)
    {
        /* others */
    }
```

唯一的例外是 switch 语句，switch-case 语句采用 case 语句与 switch 对齐的方式，例如：

```
    switch (value)
    {
    case value1:
        break;
    }
```

case 语句与前面的 switch 语句对齐，后续的语句则采用缩进的方式。分行上，如果没有什么特殊考虑，请**不要在代码中连续使用两个以上的空行。**

11.大括号与空格

从代码阅读角度，建议每个大括号单独占用一行，而不是跟在语句的后面，例如：

```
    if (condition)
    {
        /* others */
    }
```

匹配的大括号单独占用一行，代码阅读起来就会有相应的层次而不会容易出现混淆的情况。空格建议在非函数方式的括号调用前留一个空格以和前面的进行区分，例如：

```
    if (x <= y)
    {
        /* others */
    }

    for (index = 0; index < MAX_NUMBER; index ++)
    {
        /* others */
    }
```

建议在括号前留出一个空格(涉及的包括 if、for、while、switch 语句)，而运算表达式中，运算符与字符串间留一个空格。另外，不要在括号的表达式两侧留空格，例如：

```
    if ( x <= y )
    {
        /* other */
    }
```

这样括号内两侧的空格是不允许的。

12.日志信息

代码中多使用gcs().send_text的方式来输出日志到地面站，内存卡记录格式使用，例如：

```
//输出日志到地面站
gcs().send_text(MAV_SEVERITY_INFO,"Ready to track");

//输出日志到内存卡
AP::logger().Write("PSC",
                        "TimeUS,TPX,TPY,PX,PY,TVX,TVY,VX,VY,TAX,TAY,AX,AY",
                        "smmmmnnnnoooo",
                        "F000000000000",
                        "Qfffffffffff",
                        AP_HAL::micros64(),
                        double(pos_target.x * 0.01f),
                        double(pos_target.y * 0.01f),
                        double(position.x * 0.01f),
                        double(position.y * 0.01f),
                        double(vel_target.x * 0.01f),
                        double(vel_target.y * 0.01f),
                        double(velocity.x * 0.01f),
                        double(velocity.y * 0.01f),
                        double(accel_target.x * 0.01f),
                        double(accel_target.y * 0.01f),
                        double(accel_x * 0.01f),
                        double(accel_y * 0.01f));
```

- 日志应该是以输出易懂易定位问题的方式。"天书式"的日志系统是糟糕的，不合理的，不应该出现在代码中。
- 严禁在运行频率大于50Hz线程中直接打印大量日志，尽可能的避免或轻量化。
- AP::logger().Write()函数第一个日志项必须是时间，单位为微秒，整个日志名称描述字符不能大于60个字符

13.函数

函数应该尽量精简，仅完成相对独立的简单功能。函数的实现不应该太长，函数实现太长，应该反思能够如何修改(或拆分)使得函数更为精简、易懂。

14.格式化代码

格式化代码是指通过脚本自动整理你的代码，并使其符合 RT-Thread 的编码规范。本文提供以下两种自动格式化代码方法，可以自行选择或配合使用。

使用 formatting 格式化

使用 formatting 扫描文件来格式化代码：formatting 可以满足编码规则的基本要求，如：

- 将源文件编码统一为 UTF-8
- 将 TAB 键替换为 4 空格
- 将每行末尾多余的空格删除，并统一换行符为 '\n'