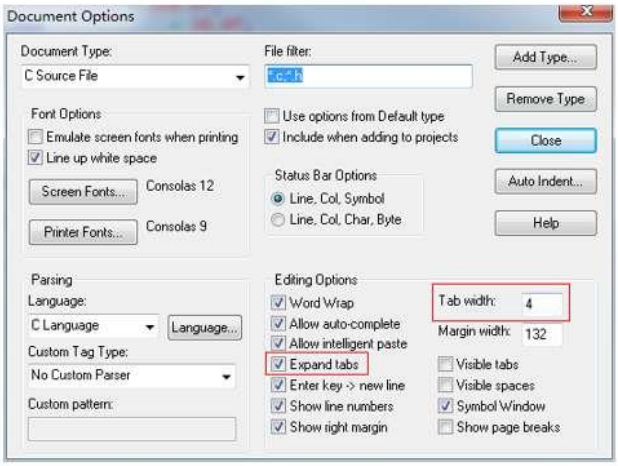
1 排版规范

排版规范主要是指在编写代码的过程中对代码进行合理缩进、空行，使得代码变得简洁、清晰、明了。

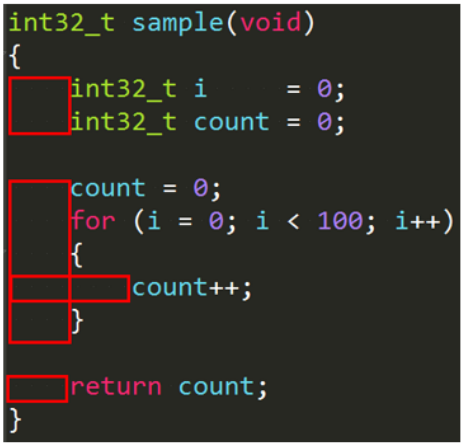
R 1-1 【A】代码的编写采用缩进风格，每级缩进为 4 个空格，不允许使用 TAB 制表符

在编写函数体、代码块时，遇到需要缩进的地方全部采用空格的方式进行缩进，不允许使用未经转换的 TAB 制表符。绝大多数编辑器都支持将 TAB 转换为空格的功能，在设置时请将 TAB 设置为 4 个空格，方便快速缩进。

1) 对于 Source Insight 编辑器，在 Options->Document Options 中勾选“Expand tabs”复选框，并在右侧的“Tab width”中设置 4，代表一个 tab 对应 4 个空格，并且将所有 tab 转换为 4 个空格。



R1-1 示例：



```
int32_t sample(void)
{
    int32_t i     = 0;
    int32_t count = 0;

    count = 0;
    for (i = 0; i < 100; i++)
    {
        count++;
    }

    return count;
}
```

R 1-2 【B】相对独立的代码块之间、变量定义之后、函数返回之前，必须加空行

1) 函数体开头定义变量之后，空一行；

2) 不同功能的代码块之间，空一行；

3) 函数返回 return 语句之前，空一行。

R1-2 示例：

```
int32_t sample(void)
{
    int32_t i     = 0;
    int32_t count = 0;

    count = 0;
    for (i = 0; i < 100; i++)
    {
        count++;
    }

    return count;
}
```

R 1-3 【A】空格的使用

1) **诸如 if, for, while, switch 关键字后与括号之前必须加空格；**

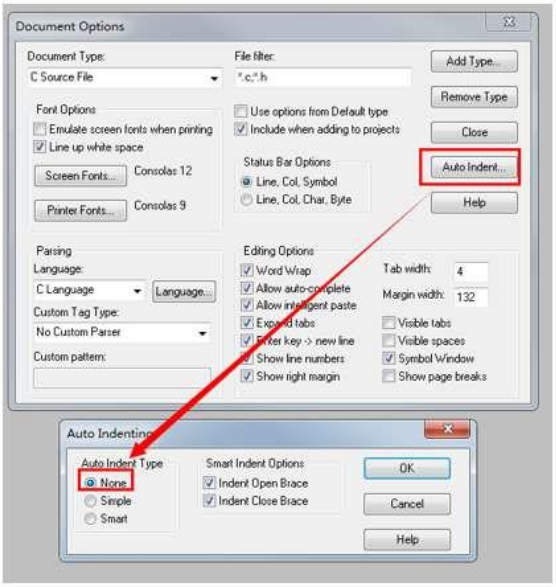
2) 在空行处不能有空格、tab，在语句或表达式行末也不能有空格、tab；

如：（CR,LF 表示换行回车）

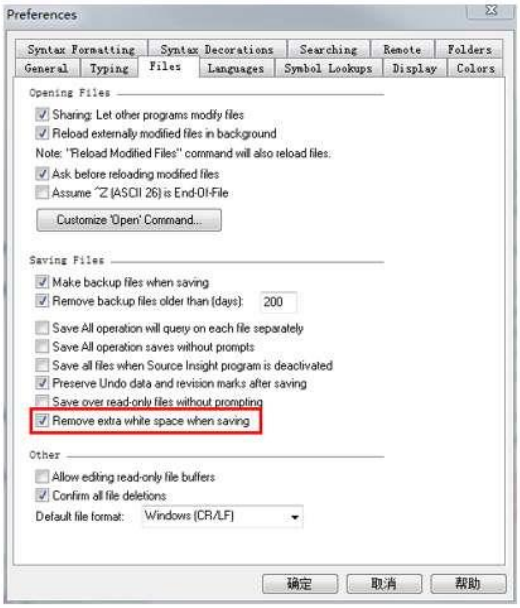
```
int32_t sample(void)CRLF
{CRLF
    int32_t i     = 0;CRLF
    int32_t count = 0;CRLF
CRLF
    count = 0;CRLF
    for (i = 0; i < 100; i++)CRLF
    {CRLF
        count++;CRLF
    }CRLF
CRLF
    return count;CRLF
}
```

对于 Source Insight 编辑器，在遇到句末换行回车时会因为自动缩进补充一行多余的空格。为避免换行时引入不必要的空格，取消编辑器的自动对齐功能即可，设置如下，在 Document

Options->Auto Indent 中选择 None，表示不适用自动缩进对齐功能，一行结束回车时光标跳转到每行开头。



上述选项也可以选择 Simple，表示一行结束后跳到对应语句的开头，但是这样会引入多余的空格，此时务必要设置好另一个选项：在 Source Insight 中可以设置自动清除掉空行中多余的空格以及每行句末的空格，设置方法如下：勾选 Options->Preferences->Files 下的“Remove extra white space when saving”选项。



3) 双目操作符（如关系“>,<,>=,<=,==,!=”、赋值“=”、算术“+,-,*,/,%”、逻辑“&&,||”和位运算“>>,<<,&,^,|”操作）的前后必须加一个空格；

如：

```
a = b + 2;
if ((a > b) && (a > c))
```

4) 在函数定义以及函数调用时填写的函数形参与实参与逗号之间必须加一个空格，即逗号运算符“,”后需要加一个空格；

如：

```
void sample(int32_t *max, int32_t a, int32_t b, const int8_t *string)
{
    xxx;
}
```

```
sample(&max, a, b, "This is a format demo!");
```

5) 单目操作符“!,++,--,&” 前后不加空格，缩写赋值运算符“+=,-=,*=,/=,%=,>>=,<<=,&=,|=,^=”缩写操作符中间不加空格，左右必须加空格；

如：

```
a++;
c = &b;
d *= a;
d |= c;
d <<= b;
```

6) 下标运算符“[]”，结构体指针箭头“->”，点号“.”前后不加空格。

如：

```
item = opentable[index + i];
p->pid = out;
stu.no = 0x1234;
```

7) 指针的解引用符号 “*”，定义时靠近变量名。

如：

```
void sample(int32_t *max, fp32 *offset, int8_t *ptr)
{
    xxx;
}
```

R 1-4 【A】if…else…语句排版规范

1) 两个分支语句必须加大括号，且两个分支后面的左大括号必须换行；

2) if 和 else 关键字与后面的左括号之间隔一个空格；

3) if 和 else 的条件判断表达式内部的双目操作符注意要用空格隔开。

R1-4 示例：

```
int32_t sample(int32_t a, int32_t b)
{
    int32_t max = a;

    if (a > b)
    {
        max = a;
    }
    else
    {
        max = b;
    }

    return max;
}
```

R 1-5 【A】for 语句排版规范

1) 必须用大括号把循环体括起来，且左大括号必须换行；

2) for 关键字与左括号之间必须加一个空格；

3) 括号内的变量赋值、条件判断和累加条件注意要用空格隔开，分号左边不需要空格。

R1-5 示例：

```
count = 0;
for (i = 0; i < 100; i++)
{
    count++;
}
```

R 1-6 【A】while、do…while 语句排版规范

1) 必须用大括号把循环体括起来，且左大括号必须换行；

2) while 关键字与左括号之间必须加一个空格；

3) 括号内的条件判断内部的双目操作符注意要用空格隔开。

R1-6 示例：

```
i = 0;
count = 0;
while (count <= 100)
{
    i++;
}
```

```
i = 0;
count = 0;
do
{
    i++;
    count++;
}while (count <= 100)
```

R 1-7 【A】switch 语句排版规范

1) 必须用大括号把循环体括起来；

2) 每个 case 与 switch 关键字对齐，且单独成一行；

3) 每条 case 下的语句与 case 关键字隔 4 个空格；

4) break 单独成一行，写在每条 case 语句的最后一行；

5) 每个 switch 语句必须有 default 分支。

R1-7 示例：

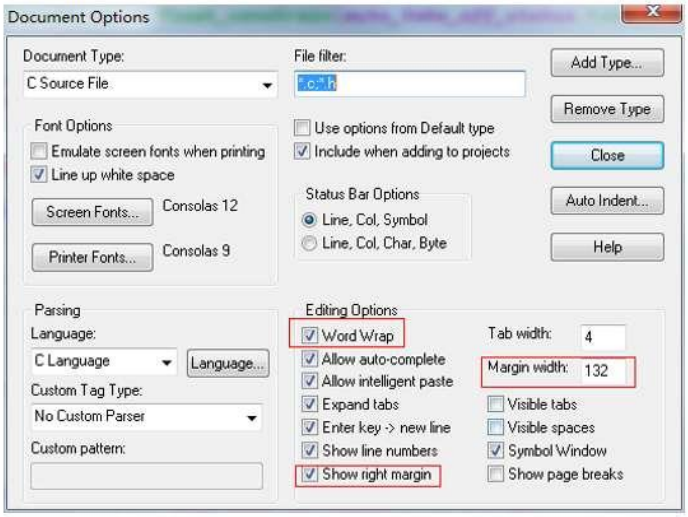
```
switch (branch)
{
case LS_COMP:
    xxx;
    xxx;
    break;
case LS_DIRECT:
    xxx;
    xxx;
    break;
default :
    break;
}
```

R 1-8 【B】较长的语句一行放不下时，操作符放在新行之首，使用空格缩进对齐

当一条语句过长时（如状态条件的准入条件、if 条件判断、for 循环条件、函数实参等），需要进行适当的划分分行。

1) 较长的表达式或语句，在低优先级操作符处划分，操作符放在新行之首，新行必须空格对齐，对齐准则是以第一个括号后为准，列对齐；

2) 要求每行语句列长度不超过 132 个字符（表格除外），超过的部分按照逻辑功能换行。为了方便检查，在 Source Insight 中，Options->Document Options，勾选“Show right margin”复选框，并在右侧的“Margin width”设置 132。此时编辑区会出现一列灰色的竖线，代码不允许超过该条竖线，超过的部分按照上述原则换行。同时需要勾选“Word Wrap”复选框，勾选之后 SI 会自动在 132 列处换行。



R1-8 示例：

```
#define EMERGENCY_BREAK_FLAG (((g_real.config.avoid_obstacle_limit_cfg.avoid_obstacle_enable == 1)
                             && (g_status.vo.status_data.bit.avoid_not_connect_flag == 0))
                             || ((get_horiz_mode() >= HORIZ_CTRL_VEL)
                             && (g_status.vo.avoid_obstacle_flags.func_en == 1)
                             && (g_status.vo.avoid_obstacle_flags.stop_flag == 1))) ? TRUE:FALSE
```

```
for (i = 0, j = 0;
    (i < first_word_length) && (j < second_word_length);
     i++, j++)
{
    xxx;
}

n7stat_str_compare((BYTE *) & stat_object,
                   (BYTE *) & (act_task_table[taskno].stat_object),
                   sizeof (_STAT_OBJECT));

n7stat_flash_act_duration(stat_item, frame_id * STAT_TASK_CHECK_NUMBER
                          + index, stat_object);
```

R 1-9 【A】条件判断中使用括号区分判定优先级

在条件判断表达式中，出现较为复杂的逻辑时（如比较与逻辑操作符），必须用括号把优先级更高的每一条运算括起来，防止出现逻辑错误。

R1-9 示例：

```
if ((a > b) && (a > c))
{
    max = a;
}
```

2 注释规范

注释的目的在于向其他人解释某段代码的功能和用途，方便其他人在阅读、修改、重构代码时能够快速、准确的理解代码的意图。写注释时务必遵守以下 5 条总规：

- **所有注释一律用英文**
- 不要写无意义的注释，不要重复描述代码，而是从代码所实现的功能以及注意事项角度层面上进行解释
- 边写代码边注释，修改代码的同时修改对应注释，以保证注释与代码的一致性。宁愿没有注释，也不能有错误的注释
- 避免在注释中使用非常规缩写，如有必要，应对缩写进行必要的说明
- 代码注释不要过少，也不宜太多，注释总量占总代码量的 20~30%较为合适

R 2-1 【B】优秀的代码可以自我解释，不需要注释即可轻易读懂

说明：优秀的代码不写注释也可轻易读懂，注释无法把糟糕的代码变好，需要很多注释来解释的代码

往往存在坏味道，需要重构。

示例：注释不能消除代码的坏味道：

```c
/* 判断m是否为素数 */
/* 返回值：是素数，不是素数 */
int p(int m)
{
   int k = sqrt(m);
   for (int i = 2; i <= k; i++)
      if (m % i == 0)
         break; /* 发现整除，表示m不为素数，结束遍历*/
   /* 遍历中没有发现整除的情况，返回 */
   if (i > k)
     return 1;
   /* 遍历中没有发现整除的情况，返回 */
   else
     return 0;
}
```

重构代码后，不需要注释：

```c
int32_t is_prime_number(int32_t num)
{
    int32_t i           = 0;
    int32_t sqrt_of_num = 0;

    if (num < 0)
    {
        return FALSE;
    }

    sqrt_of_num = sqrt(num);
    for (i = 2; i <= sqrt_of_num; i++)
    {
        if (num % i == 0)
        {
            return FALSE;
        }
    }

    return TRUE;
}
```

R 2-2 【B】文件头与头文件注释

R 2-2 【B】文件头与头文件注释

文件头注释是指放在.c 文件开头部分的注释，每一个 c 文件都要求有文件头注释。头文件注释则是指放在.h 文件开头部分的注释，每一个 h 文件都要求有头文件注释。两种注释均采用 Doxygen 格式，包含有文件名、简介、说明、版本历史和自定义注释五个部分：

```
/**
 ****************************(C) COPYRIGHT 2016 DJI****************************
 * @file       fmu_sa_led.c/h
 * @brief      This file handles ....
 *             .........
 *             .........
 * @note       blablabla
 * @history
 *  Version    Date           Author         Modification
 *  V1.0.0     Sep-11-2016     xxx            1. add...
 *                                           2. remove...
 *  V2.0.0     Nov-04-2016     xxx            1. repair...
 *
 @verbatim
 ==============================================================================
             1      max_thrust     1
 sys_gain = ----- * ----------- = ----- * thrust_to_mass_ratio * g
            100        mass       100
 ctrl_gain = k / sys_gain,  0.6 * 100 / (thrust_to_mass_ratio * g) = 3

 ==============================================================================
 @endverbatim
 ****************************(C) COPYRIGHT 2016 DJI****************************
 */
```

*/

R2-2 示例：

```
/**
  ****************************(C) COPYRIGHT 2016 DJI****************************
  * @file       fmu_sa_led.c/h
  * @brief      This file handles ....
  *             .........
  *             .........
  * @note       blablabla
  * @history
  *  Version    Date            Author          Modification
  *  V1.0.0     Sep-11-2016     xxx             1. add...
  *                                             2. remove...
  *  V2.0.0     Nov-04-2016     xxx             1. repair...
  *
  @verbatim
  ==============================================================================
                1      max_thrust     1
  sys_gain = ----- * ----------- = ----- * thrust_to_mass_ratio * g
              100         mass       100
  ctrl_gain = k / sys_gain,  0.6 * 100 / (thrust_to_mass_ratio * g) = 3

  ==============================================================================
  @endverbatim
  ****************************(C) COPYRIGHT 2016 DJI****************************
  */
```

1）file 部分填写对应的.c/h 文件名

2）brief 部分填写本.c/h 文件所要做的主要工作

3）note 部分填写本.c/h 文件需要注意的事项

4）history 部分：

a) Version 填写文件版本。大重构变更首数字，如 V1.0.0，V2.0.0；功能性的小范围重构，如增加和删除某些功能，修改中间一位数字；修复小 bug、打补丁，修改最后一位数字。三位数字的范围是 0~255。

b) Date 填写修改日期。为了避免月份与天数混淆，月份填写英文缩写，天数和年份填写对应阿拉伯数字

月份中文	月份英文	对应缩写
一月	January	Jan
二月	February	Feb
三月	March	Mar
四月	April	Apr
五月	May	May
六月	June	Jun
七月	July	Jul
八月	August	Aug
九月	September	Sep
十月	October	Oct
十一月	November	Nov
十二月	December	Dec

c) Author 填写对应修改人

d) Modification 填写修改的内容

5） verbatim 至 endverbatim 部分可以填写一些具有图形和符号化的注释，帮助理解本文件的意图。Doxygen 可以将此部分按照原始格式输出。

6） 注释头部分的“COPYRIGHT 2016 DJI”中的年份需要每年更新，由修改人负责

R 2-3 文件代码块分区注释

先空着

R 2-4 【B】函数头注释

函数头注释放置在每个函数的开头，用于解释本函数所做的事情。不必每个函数都有注释，只需对部分函数注释即可，如数学函数库、功能性/算法性较强的函数以及某些关键函数（api 函数必须要加函数头注释，解释函数的使用方法）。函数头注释规范如下：

```
/**
  * @brief          limit the input value to a given range, for int8_t only
  * @author         Yu Yun
  * @param[in]      value: input value
  * @param[in]      min: lower bound
  * @param[in]      max: upper bound
  * @retval         value: output of the constrained output
  */
```

R2-4 示例：

```
/**
  * @brief          limit the input value to a given range, for int8_t only
  * @author         Yu Yun
  * @param[in]      value: input value
  * @param[in]      min: lower bound
  * @param[in]      max: upper bound
  * @retval         value: output of the constrained output
  */
```

1） brief 填写本函数的功能，遇到复杂的函数需要详细注释

2） author 填写本函数的作者（创建人）

3） param[in]填写函数形参的含义

4） retval 填写函数返回值的含义，如有范围，则注明范围，如是 bool 类型，需注明状态含义。

R 2-5 【A】结构体、枚举与全局变量注释

对于结构体、联合体、枚举里面的变量命名，如果不是充分自注释的，必须加以注释。对数据结构的注释应放在上方相邻位置，不可放在下面。对结构中的每个域的注释必须放在对应域的右方，采用/* xxx */格式，并用空格对齐。

1) 对于复杂结构体、联合体、枚举类型，需要进行注释的，在数据结构的上方相邻位置用/* xxx */形式进行注释；

2) 对结构体、联合体、枚举类型中的每个域，需要注释的写在对应域的右方，采用/*!< xxx */格式，并用空格对齐。对齐标准为最长的语句后空一个 tab（实际空格数由编辑器决定）。

R2-5 示例：

```
typedef struct
{
    LS_cfg_t            cfg;

    /* process state */
    fp32                gain_dyn;       /*!< dynamic gain */
    fp32                cmd;            /*!< command */
    fp32                fdbk;           /*!< feedback */
    uint8_t             sat_flag;

    fp32                err;            /*!< error  */
    fp32                i_err;          /*!< i_err, saturation */
    fp32                P;              /*!< Proportional term */
    fp32                I;              /*!< integral term */
    fp32                PI_out;         /*!< PI out */
    butter_comp_1st_t   comp;           /*!< compensator */
    boost_comp_2nd_t    boost_comp;     /*!< compensator */
    fp32                out;            /*!< output */
} loop_shaping_t;
```

对于全局变量，必须要有较详细的注释，包括功能、取值范围以及注意事项等。

```
/*
 * Six status flag that indicate the aircraft's situational awareness.
 * These status are given by imu and navi_data.
 * The value of status is as follows:
 * 1 --- TRUE  corresponding status is valided
 * 0 --- FALSE corresponding status is invalided
 */
uint8_t gyro          = 0;
uint8_t atti          = 0;
uint8_t velocity      = 0;
uint8_t posi          = 0;
uint8_t absolute_posi = 0;
uint8_t rel_vertposi  = 0;
uint8_t null          = 0;
```

R 2-6 【B】switch 中的 no break 注释

在某些特殊情况下，需要连续执行多条 case 语句而不需要 break，此时必须要在上一条 case 语句处理完，下一条语句之前添加/* no break */注释。与代码隔一个 tab（实际空格数由编辑器决定）。

R2-6 示例：

```
switch (rc_lost_action)
{
case FAIL_SAFE_GO_HOME:
    xxx;
    break;
case FAIL_SAFE_AUTO_LANDING:
    xxx;
    break;
case FAIL_SAFE_HOVER: /* no break */
case FAIL_SAFE_INVALID:
    xxx;
    break;
default :
    break;
}
```

R 2-7 【A】代码块(//)与单行注释(//)**

无论哪种注释方法，当注释在代码右侧时，一个注释原则是与被注释代码距离一个 tab（实际空格数由编辑器决定）。

对于某些需要注释的代码块：

1） 注释时一律采用/* xxx */形式，写在被注释代码的上方，与前一段代码空一行隔开，左侧与被注释代码对齐；

2） 注意注释中的左右两个星号与注释内容加一个空格；

3） 遇到需要多行注释的代码块，采用如下格式，注意*号与注释正文空一格，左侧与被注释代码对齐。

```
/*
 * This is a comment demo!
 * This is a comment demo!
 * This is a comment demo!
 */
```

R2-7 示例：

```
/* This is a comment demo */
```

```
void rc_general_mode(void)
{
    uint8_t real_cmd_tr_mode;
    uint8_t user_cmd_tr_mode;

    /* get command target mode according to user_mode and ioc_mode_cur */
    user_cmd_tr_mode = get_user_cmd_tr_mode();

    /* find real command target mode according to 5 conditions in the mode_table[] */
    real_cmd_tr_mode = find_real_cmd_tr_mode(user_cmd_tr_mode);

    request_command_mode(REQ_RC_NORMAL, real_cmd_tr_mode);

    if (IS_RC_MODE_SWITCH_CHANGE)
    {
        reset_all_at_next("rc mode switch");
    }
}
```

```
/*
 * This is a comment demo!
 * This is a comment demo!
 * This is a comment demo!
 */
```

对于某些需要注释的单行表达式：

1）// xxx 形式的注释只允许出现在单行语句的注释中，注释时写在需要注释的代码行右侧，对齐标准为最长的语句后空一个 tab（实际空格数由编辑器决定）。当有多条语句并列注释时，保证一整列的注释对齐；

2）注意注释中的//与注释正文隔一个空格；

R2-7 示例：

```
request_command_mode(REQ_RC_NORMAL, ATTI_HOLD); // change to atti_hold
request_command_mode(REQ_RC_NORMAL, ATTI);      // change to atti
request_command_mode(REQ_RC_NORMAL, ATTI_FPV);  // change to atti_FPV
```

R 2-8 【B】TODO 注释

如果需要在代码中标注改进注意事项，便于后续的研发，一律采用 TODO 注释，标注在待改进的代码上方。规范如下：

// TODO(your dji AD): xxx

```
TODO: This action request should be moved to ctrl_situation_handler()
if (get_landing_cmd())
{
    set_landing_type(REQ_LANDING_FORCE);
    request_action_control(REQ_RC_GO_HOME, ACT_AUTO_LANDING,REQ_R_VERT_LOW_LIMIT_LANDING);
    get_fmu_debugger()->log(DEBUG_FLYMODE, "Req force landing!");
    reset_all_landing_cmd_status();
}
```

尤其注意 TODO 写法规范，未正确书写会导致 Source Insight 无法高亮 TODO 关键字：

1) 双斜杠//后必须有一个空格
2) TODO 必须英文大写
3) TODO 后必须跟一个冒号（半角）
4) TODO 后与冒号之间加入英文半角括号，括号里添加 TODO 添加人的 dji AD

R 2-9 【B】Debug 代码整段注释

在某些代码调试阶段中需要注释、保留整段代码，不建议采用常规注释（如/* */, //），一律用宏定义开启/关闭整段代码。规范如下：

```
#if 0
   xxx;
#endif
```

R2-9 示例：

```
#if 0
    facotry_install_data.torsion_output = g_status.factory_produce.const_torsion_out;
    facotry_install_data.tilt_x_output = g_status.factory_produce.const_tilt_x_out;
    facotry_install_data.tilt_y_output = g_status.factory_produce.const_tilt_y_out;

    facotry_install_data.vib_gyro_x_value   = g_status.factory_produce.vib_gyro_x_value;
    facotry_install_data.vib_gyro_y_value   = g_status.factory_produce.vib_gyro_y_value;
    facotry_install_data.vib_gyro_z_value   = g_status.factory_produce.vib_gyro_z_value;
    facotry_install_data.vib_acc_x_value    = g_status.factory_produce.vib_acc_x_value;
    facotry_install_data.vib_acc_y_value    = g_status.factory_produce.vib_acc_y_value;
    facotry_install_data.vib_acc_z_value    = g_status.factory_produce.vib_acc_z_value;
    facotry_install_data.hover_thrust       = g_status.factory_produce.const_hover_thrust;
#endif
```

3 命名规范

命名规范针对代码中出现的所有标识符，如函数名、变量名、宏定义等。

R 3-1 【A】标识符采用 Unix 风格

Unix 风格命名的特点是标识符中不同动作、名词之间用下划线隔开，如：get_system_status()，check_rc_is_lost()。所有函数名、变量名、宏定义等全部采用 Unix 风格。

R 3-2 【A】缩写规范

英文全称	对应缩写	英文全称	对应缩写
command	cmd	increment	inc
target	tgt	decrease	dec
current	cur	error	err
feedback	fdbk	maximum	max
feedforward	fdfwd	minimum	min
control	ctrl	private	priv
reference	ref	previous	prev
system	sys	internal	int
altitude	alti	external	ext
position	pos	argument	arg
velocity	vel	buffer	buff
accelerate	acc	clock	clk
frequency	freq	compare	cmp
filter	fltr	hexadecimal	hex
vector	vec	register	reg
integral	intg	semaphore	sem
process	proc	synchronize	sync
		configuration	cfg, config
temp	tmp	device	dev
counter	cnt	initialize initialization	init
		message	msg
		request	req
		battery	bat
		voltage	vol
		navigation	navi
		parameter	para

R 3-3 【B】结构体与枚举名称

结构体与枚举在用 typedef 定义时，名称结尾必须增加_t 后缀。自定义类型名必须与大括号空一格。

R3-3 示例：

```
typedef enum
{
    REQ_LANDING_NORMAL,
    REQ_LANDING_FORCE,
    REQ_LANDING_SIZE,
}requrester_landing_type_t;

typedef struct
{
    uint8_t requester_id;
    fp32    cmd_delay_time;
}cmd_req_t;
```

R 3-4 【A】数据类型定义

代码中只允许出现以下数据类型定义（32 位单片机），用右列的新数据类型替换掉左列的原始数据类型：

原始数据类型	新数据类型
signed char	int8_t
unsigned char	uint8_t
signed short	int16_t
unsigned short	uint16_t
signed int	int32_t
unsigned int	uint32_t
signed long long	int64_t
unsigned long long	uint64_t
float	fp32
double	fp64
unsigned char	bool_t

4 宏规范

所有的宏写在.c/.h 文件的前面（#include 之后，全局变量/函数定义之前），不允许出现在函数体内。

R 4-1 【A】所有宏必须采用大写字母

代码中出现的宏定义表达式，必须全部用大写字母，并采用 Unix 风格。

R4-1 示例：

```
#define DEFAULT_RC_LOST_SEARCH_ENABLE (0)
```

R 4-2 【A】带参数的宏定义表达式必须使用完备的括号

在使用带参数的宏定义时，必须给每项参数加一个括号，防止出现意想不到的风险。

R4-2 错误示例：

```
#define RECTANGLE_AREA(a, b)    (a * b)
```

应改为

```
#define RECTANGLE_AREA(a, b)    ((a) * (b))
```

R 4-3 【A】用宏来定义多条表达式时，应放在大括号中

R4-3 错误示例：

```
#define INIT_VALUE(a, b) \
    a = 1; \
    b = 2;

for (i = 0; i < limit; i++)
    INIT_VALUE(RectArray[i].Length, RectArray[i].Width);
```

上述代码中，b=2 只会在退出 for 循环后执行一次，并不是我们想要的结果。正确写法如下：

```
#define INIT_RECT(a, b) \
{                       \
    a = 1;              \
    b = 2;              \
}

for (i = 0; i < limit; i++)
{
    INIT_RECT(RectArray[i].Length, RectArray[i].Width);
}
```

同时注意如上图红框中所示换行标记也要对齐。

5 函数规范

R 5-1 【A】函数定义与声明规范

在.c 文件中定义新函数，对于私有函数，即不对外公开的函数前面必须添加 static 声明。如果函数需要对外公开，则需要在该函数的.h 头文件中进行声明，声明时必须添加前缀 extern。

R5-1 示例：

私有函数：.c 文件中：

```
static void launching_check(fmu_fsm *fsm)
{
    xxx;
}
```

对外公开函数：.c 文件中：

```
fp32 float_square(fp32 x)
{
    xxx;
}
```

需要在对应.h 文件中进行声明：

```
extern fp32 float_square(fp32 x);
```

R 5-2 【A】局部变量声明规范

函数的局部变量在定义时，必须对其进行初始化，不允许出现没有初始化的局部变量。

- 每一个局部变量占据一行；
- 局部变量的声明遵循“就近原则”，变量声明的位置应该靠近与之紧密相关的代码块。
- 数据类型长度不一致的，以最长的类型右侧空一格对齐；
- 右侧的赋值等号必须对齐，对齐原则为以最长的变量右侧空一格为准；
- 数组、结构体全体初始化为 0 时，不要在大括号内添加空格。

R5-2 示例：

```
void battery_voltage_protect_handle(void)
{
    static uint8_t  cancel_smart_battery_go_home = 0;
    uint32_t        cancel_vol1_protect          = 0;
    battery_status  bat_status                   = {0};
    fp32            tgt_vert_vel                 = 0.0f;
    fp64            imu_bais[3]                  = {0.0};
}
```