|  |
| --- |
| **源代码工作室** |
| 代码编写规范 |
| 版本号 01-001-003 |

|  |
| --- |
| 罗 斌  [2014年02月02日] |

目 录

[代码编写规范 1](#_Toc342488787)

[1 总则 1](#_Toc342488788)

[2 名词 1](#_Toc342488789)

[3 规范内容 1](#_Toc342488790)

[3.1 命名 1](#_Toc342488791)

[3.1.1 变量命名 1](#_Toc342488792)

[3.1.2 结构体、联合体命名 1](#_Toc342488793)

[3.1.3 函数命名 2](#_Toc342488794)

[3.1.4 宏命名 2](#_Toc342488795)

[3.1.4 文件命名 2](#_Toc342488796)

[3.2 格式 2](#_Toc342488797)

[3.2.1 版面格式 2](#_Toc342488798)

[3.2.2 排版格式 3](#_Toc342488799)

[3.2.3 内容顺序 4](#_Toc342488800)

[3.3 信息头部 4](#_Toc342488801)

[3.3.1 文件信息 4](#_Toc342488802)

[3.3.2 函数信息 5](#_Toc342488803)

[3.4 注释 6](#_Toc342488804)

[3.4.1 变量注释 6](#_Toc342488805)

[3.4.2 宏注释 7](#_Toc342488806)

[3.4.3 代码注释 7](#_Toc342488807)

[4 附则 7](#_Toc342488808)

代码编写规范

# 1 总则

1.1 为确保源代码工作室的代码风格一致和方便阅读，制定本规范。

1.2 本规范对编码格式、说明信息、命名、注释进行了规定。

1.3 对于参与本项目的开发者，请遵守本规范。

# 2 名词

无

# 3 规范内容

规范明确了命名、格式、信息头部、注释的编写要求。

## 3.1 命名

各种命名均应反映其用途。名称原则上采用其功能的英文全称，禁止使用拼音代替英文。涉及多个单词时，用下划线连接。对于已经约定俗成的，可以采用简化的命名方式。

### 3.1.1 变量命名

3.1.1.1 变量名全部使用小写字母，全局变量不能使用下划线开头。

3.1.1.2 变量命名的基本格式为：模块名称\_功能描述

例如 : timer\_pool

### 3.1.2 结构体、联合体命名

名称以下划线开始，以“\_t”结尾。定义结构体时，同时定义别名，别名为结构体名称去掉前导下划线的部分。

例如：

typedef struct \_proc\_list\_t

{

spin\_lock\_t pl\_lock;

proc\_t \* pl\_list;

}proc\_list\_t;

### 3.1.3 函数命名

3.1.3.1 原则上函数名要以字母开头，且第一个字母要求大写，其余字母采用小写。

3.1.3.2 收录入系统库的通用函数，以下划线紧接着字母开头，所有字母采用小写。可以采用缩写，以及省略名称中间的下划线。

例如: \_memzero

3.1.3.3 命名的基本格式为 : 模块名称\_功能描述

例如: Proc\_create

### 3.1.4 宏命名

3.1.4.1 原则上全部采用大写字母。用于表示函数时，使用与函数相同的命名原则。

3.1.4.2 命名的基本格式为 : 模块名称\_功能描述

例如: PROC\_LAST\_PROC

### 3.1.4 文件命名

文件名应反映其模块和用途。

## 3.2 格式

格式包含版面格式、排版格式、内容顺序。

### 3.2.1 版面格式

3.2.1.1 打印程序的纸张规格为A4（210mm×297mm）。页边距设置为：上25mm，下20mm，左25mm，右20mm。

3.2.1.2 采用等宽字体，字体大小为9号，行间距为12磅。为保证中文部分显示正常，应采用中文等宽字体。

3.2.1.3 程序文本每行最多为80个半角英文字符,含行尾的回车符。

### 3.2.2 排版格式

程序应保证在各种文本编辑器中都有相同的显示效果。

3.2.2.1 宏定义格式

顶格书写，宏名称与#define关键字间隔一个空格，定义体从37列开始。

例如：

#define TIMER\_MAX 8

3.2.2.2 别名定义格式

顶格书写，原名与typedef关键字间隔一个空格，别名从29列开始。

例如：

typedef unsigned int uint\_t;

3.2.2.3 函数定义格式

顶格书写，函数名原则上从13列开始，同一模块的函数名保持对齐。参数较多，需要分行时，参数列表应对齐。

例如：

proc\_t \* Proc\_create(const char \* name ,byte\_t priority , byte\_t prionum,

proc\_entry\_t entry,void \* param , void \* stack,

int stacksize);

void Proc\_exit (int code);

result\_t Proc\_kill (int pid);

3.2.2.4 结构体、联合体定义格式

顶格书写，成员类型从第5列开始，成名名称从29列开始。

例如

typedef struct \_list\_node\_t

{

struct \_list\_node\_t \* ln\_prev;

struct \_list\_node\_t \* ln\_next;

}list\_node\_t;

3.2.2.5 花括号格式

花括号要与本层级在同一列，括号要独占一行，表示分隔。

例如：

void \* Clk\_ticks\_hook\_get(void)

{

return clk\_ticks\_hook;

}

3.2.2.6 缩进格式

程序以缩进方式来区分作用范围。每个层级缩进4个空格，不能使用制表符缩进。最大缩进级数为3级。

例如：

for( i = 0 ; i < TIMER\_MAX ; i++)

{

if( TIMER\_IS\_FREE(timer\_pool + i) )

{

tm = timer\_pool + i;

tm->tm\_handle = (tm\_handle\_t)(-1);

break;

}

}

### 3.2.3 内容顺序

3.2.3.1 头文件内容按常数定义、数据类型定义、全局变量声明、函数声明的顺序排列。宏跟随其相关的内容。依赖性少的内容安排在靠前位置。

3.2.3.2 源文件内容按变量声明、内部函数、外部函数排列。依赖性少的内容安排在靠前位置。

3.2.3.3 包含文件根据需要安排，原则安排在文件头。

## 3.3 信息头部

信息头用于提供基本的参考信息。Lenix规定文件头和函数须提供信息头。

### 3.3.1 文件信息

源代码工作室要求每个文件，包括汇编文件、头文件、源文件等，都需要提供文件头。文件头提供的信息包括：版权信息、创建时间、用途、注意事项等内容。基本格式为：

/\*

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

// 产品名称

// 2011 - 2012 @ 源代码工作室

// 保留所有权利

// \*\*\*----------------------------\*\*\*

// 名 称:

// 创建时间: xxxx-xx-xx 创建者:

// 修改时间: 修改者:

//

// 主要功能:

// 说 明:

// 变更记录:

// 版 本 号 | 时 间 | 作 者 | 主要变化记录

//=============================================================================

// 00.00.000 | xxxx-xx-xx | xxxx | xxxxxx

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

\*/

版本变化记录是用于记录较为重要的变化，按时间的降序排列，即最近的日期排在最上方。

### 3.3.2 函数信息

Lenix要求每个函数都需要提供函数信息头。信息头包含函数作用，参数，返回值等信息。基本格式为：

/\*

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

// 名 称:

// 功 能:

// 参 数:

// 名称，第9列 :类型，第21列 | 说明：第37列开始

// pci : pci\_t \* | pci对象指针

// 返回值:

// 注 意:

// 变更记录

// 时 间 | 作 者 | 说 明

//=============================================================================

// xxxx-xx-xx | |

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

\*/

变更记录用于记录较为重要的变化，按时间的降序排列，即最近的日期排在最上方。

## 3.4 注释

注释统一采用C语言风格，避免使用C++风格的注释。多行注释的情况，首行仅放置注释起始标志，尾行仅放置注释结束标志，中间行以\*号开头，后紧接两个空格。注释部分应保持星号对齐。基本格式如下：

/\*

\* …………

\*/

### 3.4.1 变量注释

3.4.1.1 原则上要求每个变量都应提供说明，对于约定俗成的变量名，可以省略。

3.4.1.2 应在定义变量后立即注释，说明变量用途。注释与分号的距离应控制在12个空格以内，且保持4字符对齐。

例如：

volatile uint32\_t ticks; /\* 时钟节拍计数器 \*/

3.4.1.3 如注释不能在一行内写明，可以将注释放置在变量定义的上方。

例如：

/\*

\* 空闲进程，用以保证系统中没有可运行的进程时，调度程序仍然可以找到可运行的进程

\*/

static proc\_t \* proc\_idle;

3.4.1.4 如果变量名采用了缩写，应在注释中说明。

例如：

/\*

\* 运行态进程列表,

\* Running Statu Process List, 简写为RSPL

\*/

static proc\_t \* proc\_rspl[PROC\_PRIORITY\_MAX + 1];

3.4.1.5 连续定义多个变量时，在方便阅读的情况下，应保持注释整齐。

例如：

static device\_t dev\_pool[DEVICE\_MAX]; /\* 系统设备对象池 \*/

static device\_t \* dev\_fdl; /\* 空闲设备列表 \*/

static device\_t \* dev\_sdl; /\* 系统设备列表 \*/

3.4.1.6 需要修改已有注释，但原有注释不宜删除时，应采用新增注释，并添加时间的方式。增加的注释放在末尾。

例如：

/\*

\* xxxxx

\*

\* 2012.06.07

\* xxxxx

\*

\* 2012.08.03

\* xxxxx

\*/

### 3.4.2 宏注释

宏注释参照函数的信息头格式。

### 3.4.3 代码注释

3.4.3.1 代码的注释原则上安排在需要说明的代码上方，并保持与代码相同层次。

例如：

/\*

\* 跳过空闲的进程对象，由于自身的PID仍为0，而系统不会产生0的编号，所以不用

\* 理会自身

\*/

if( PROC\_IS\_FREE(&proc\_pool[i]) )

continue;

3.4.3.2 如代码本身较短，且需要注释的内容也较短，可以安排在同一行。注释与分号应控制在12个空格以内，并且保证4个字符对齐。

例如：

if( ++pid < 1 ) pid = 1; /\* 0是系统进程的PID \*/

# 4 附则

4.1 本规范以后会逐步扩大范围，并细化。

4.2 本规范仅对发布后的源代码有效。对于已经采用更早规范的编码，如未对代码进行改动，不需要进行修改。

4.3 欢迎提出意见和建议，我们将采纳合理部分。