@嵌入式软件-代码风格规范-C 语言@

说明:

本文档规范公司嵌入式软件开发小组 C 语言编码规范.

清晰 简单 透明

1.文件名及布局

文件名亦由小写字母组成，且最好是用有意义的单词，单词间亦用下划线分隔(_)；C 头文件后缀为“.h”，C 代码文件后缀为“.c”，带宏汇编文件后缀为“.S”,不带宏汇编文件后缀为“.s”.

例如：DevSpec_CodeStyle-C_sample_code.h DevSpec_CodeStyle-C_sample_code.c

代码存放亦采用如下布局：

```
PACKAGE_DIR/
├── ChangeLogs
├── Makefile
├── README
├── scripts
│   └── TARGET.mk_Platform.BoardName.ProductModel.sh
├── tests
│   ├── Special
│   ├── Common
│   └── Platform.BoardName.ProductModel
├── TARGETs
│   └── TARGET.mk_Platform.BoardName.ProductModel
├── ext
│   ├── include
│   └── lib
├── build
├── demo
├── doc
├── include
├── lib
├── release
└── src
```

$(PACKAGE_DIR)：表示软件包所在的目录；

README：简单的软件包内容和用途说明，及如何编译安装等说明；

Makefile：编译此软件包的 make 脚本；

ChangeLogs：说明软件包修改记录,必须说明修改者,日期/时间,修改内容；

doc：存放软件包相关的文档，包括需求(Requirements),规格(Specifications),设计(Design),数据手册(DS-Datasheets)，工作流程(WF-Workflow),开发指导(DG-Development Guider),开发笔记(DN-Development Notes),用户指导(UG-User Guider),应用笔记(AN-Application Notes)等；

（可编辑源文档推荐采用 libreoffice 格式，发布文档推荐采用 PDF 格式）

include：存放此软件包的头文件，特别是当软件包为库时，存放此库对外发布的头文件；

src：存放软件包的 C 源代码文件，软件包内部头文件也可放在此目录；

lib：存放软件包自己生成的库；

tests: 存放软件包测试用例. 每个软件包包含三种类型测试用例“通用”，“专用”和“特殊”；每个测试用例都必须注明测试目的、环境和结果等信息；所有的测试集合必须有一个汇总文档(TC-Test Cases)；

通用(Common)：任何 TARGET.mk_xxx 都能编译测试用例并执行；

专用(Platform.BoardName.ProductModel)：针对特定的平台(Platform)+硬件平台(BoardName)+产品(ProductModel)而编写的测试用例，仅能在此目标下编译测试用例并执行；

每个软件包必须针对特定的目标编写和汇集(符号链接到通用测试用例)测试用例，并在软件包发布(打包)前通过这些用例进行发布验证(RV-Release Verify)；

特殊(Special)：特殊的测试用例，仅软件包内部使用；

demo: 存放工程示例. 如果示例代码也存在类似测试相关的差异，需要通过文件名或目录方式进行区分；

build: 编译过程临时生成的“.o”，“.a”，“.lib”放在此目录.

TARGETs: 此软件包支持的所有生成目标(TARGET.mk_xxx)；

参考：http://eswsrv/eswiki

release: 此软件包打包发布文件；

ext: 存放此软件包外部依赖的内容，例如头文件和库等，例如“ext/include/xeR.h”；

scripts：存放构建此软件包的脚本，例如在软件包目录下执行：

```
$ ./scripts/TARGET.mk_Platform.BoardName.ProductModel.sh
```

2.源代码文件头格式

每个源代码文件头部应包含如下内容，可参考 DevSpec_CodeStyle-C_sample_code.h DevSpec_CodeStyle-C_sample_code.c：

```
/*********************************************************************************
 * Filename: DevSpec_CodeStyle-C_sample_code.h
 *********************************************************************************
 * Description:
 *           this is a sample .h file header.
 * Version:
 *           V_0.2  Jan-4-2007
 * Copyright:
 *           HISOME <www.hisome.com.>
 * Author:
 *           developer <developer@hisome.com.>
 * History:
 *     Aug-8-2007  developer1<developer1@hisome.com>
 *           V_0.2 add a new API hx_new_func().
 *     Jan-4-2007  developer<developer@hisome.com>
 *           V_0.1 initial version of this code.
 *********************************************************************************/
```

Description: 对此文件所包含的内容的简短描述, 供后期代码维护人员了解此文件内容和用途.

Version: 此文件中代码的当前版本及当前版本的最后更新日期.

Copyright: 须填写 HISOME <http://www.hisome.com.>, 表示此代码版权归“杭州恒生数字设备科技有限公司” 所有.

Author: 表示此文件的创建者.

History: 表示此文件的更新记录, 包括日期, 修改者, 修改记录.

3.头文件函数声明要求

函数声明处必须有函数用途说明, 参数说明(IN 表示输入,OUT 表示输出,INOUT 表示传入可修改指针), 函数返回值, 可能错误描述等注释.

函数注释中亦带有此函数是否可重入信息,即是否支持多线程并发调用.

函数名亦用小写单词组成, 单词间用下划线(_)连接。函数名必须有合适的前缀以表示其所属的库或所在位置, 例如：如果某库的名字为 libxeXXX，则其开放的 API 函数应该用“xeXXX_”作为前缀；单下划线(_)开头的函数名表示库内部模块间接口函数；双下划线(__)开头的函数名表示“*.c”文件内部函数, 且必须用 static 关键字修饰此函数；

对于库对外公开的 API 函数, 声明时必须用 extern 关键字修饰此函数.

建议:

- 函数参数数量不亦太多, 小于等于 4 个为适；
- 函数参数为基本类型时(char,short,int,long,float,double)可直接传值. 若为复杂数据类型(struct ...)则应传递指针,不存在函数修改参数内容情况下需用“const”关键字修饰此参数；
- 函数必须有返回值,一般过程性函数操作成功返回“XER_SUCCESS”,失败返回“XER_FAIL”,可参考 C 库中 errno 机制获取错误代码；特殊情况可返回“void”或其它类型,但必须在函数声明注释中具体说明；若存在函数返回指针情况,失败时必须返回“NULL”；

```
/*******************************************************************************
 * Function: xeXXX_sample_func();
 *------------------------------------------------------------------------------
 * Description:
 *     this is a sample func declare.
 *------------------------------------------------------------------------------
 * @arg1: IN arguments 1, Must supply;
 * @arg2: IN arguments 2, May NULL;
 * @arg3: IN/OUT, hold return values;
 * @flags: IN other flags;
 *------------------------------------------------------------------------------
 * Returns:
 *  XER_ERROR: FAIL;
```

```
 *  XER_SUCCESS: SUCCESS;
 *-------------------------------------------------------------------------------
 * possible errors:
 *           ERRNO1: invalid arguments;
 *           ERRNO2: out of memory;
 *-------------------------------------------------------------------------------
 * History:
 *     Add Modification history here if it has
 ********************************************************************************/
extern xeR_t xeXXX_sample_func(IN const char *arg1,IN const char *arg2,
                               INOUT char *arg3, IN unsigned long  flags);
```

4. 头文件数据结构声明要求

数据结构定义处亦有其描述，表示用途等注释.

数据结构成员亦有其意义和使用等注释.

```
/*******************************************************************************
 * Structure:
 *     xeXXX_sample_struct;
 * Description:
 *     this is a sample struct.
 *     It is used to ......
 ********************************************************************************/
typedef struct xeXXX_sample_struct
{
      int member1;/*this member used to ...*/
      char *name;/*name of ..., should free() it after use.*/
}xeXXX_sample_t;
```

5. 代码注释要求

5.1 模块内部函数和数据结构注释要求

对于模块内部函数注释，亦遵照<<3. 头文件函数声明要求>>.

对于模块内部数据结构注释，亦遵照<<4. 头文件数据结构声明要求>>

5.2 代码注释

在代码中，有涉及条件判断，逻辑运算，算术运算等处，都必须添加足够的注释,其它人员只看注释就明白代码如何工作.

注释应采用 English,不推荐使用中文. 若英文表达能力不好,尽量编写最简单的语句表达意思.

推荐风格如下:

风格 1:

```
a = b*3; /*a is 3 times large than b.*/


//a is 3 times large than b.
a = b*3;
```

风格 2:

```
/*************************************
 * why we should check totalnum
 * 1. totalnum>1024
 *        reason 1 ......
 * 2. totalnum <=1024
 *             reason 2 ......
 *************************************/
    if( totalnum > 1024)
    {
         ......
    }
    else
    {
         ......
         return -1;/*why we return -1*/
    }//END of if( totalnum > 1024)
```

风格 3

```
#ifdef CONDITION_A
      ......
#else /*NOT CONDITION_A*/
      ......
#endif /*NOT CONDITION_A*/

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

#ifdef CONDITION_A
      ......
#endif /*CONDITION_A*/
```

```
#ifndef CONDITION_A
      ......
#else /*CONDITION_A*/
      ......
#endif /*CONDITION_A*/

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

#ifndef CONDITION_A
      ......
#endif /*NOT CONDITION_A*/
```

风格 4:

```
#if 0
     ${OLD_CODE_HERE};
#endif//0

     /*********************************************
     * Oct-22-2008 modified by devA<devA@hisome.com>
```

```
      * ${REASON}......
      ********************************************/
      ${NEW_CODE_COMES_HERE};
      ......
      //END of Modification
```

6. 宏和枚举定义要求

宏定义亦用大写单词，分隔符亦用下划线(_)．枚举定义参考宏定义，且必须有注释.

例如:

```
/* ioctl function codes */
#define FIO_NREAD      (1)/* get num chars available to read */
#define FIO_FLUSH      (2)/* flush any chars in buffers */
#define FIO_OPTIONS    (3)/* set options (FIOSETOPTIONS) */
#define FIO_BAUDRATE   (4)/* set serial baud rate */
#define FIO_DISKFORMAT (5)/* format disk */
```

7. 函数编写要求

函数框架应采用如下形式:

```
const char*
func_with_lots_of_args (int an_integer, long a_long, short a_short, \
                        struct a_struct *as_p)
{
      ......
}
```

函数返回值类型放在函数名上一行，若返回类型为 int 也可和函数名放在同一行．函数参数直接跟在函数名圆括号之后,若函数参数太长需要换行,行尾要添加续行符'\',后续行亦和上行参数起始位置对齐或略偏后.

函数体应采用如下形式:

```
static int __sample_file_level_func(int argA, int argB)
{
      int l_argA=0,l_argB=0;

      ////////////////////////////////////////////////////////////////
      if(argA==argB)
      {
            ......
      }//END of argA==argB
      else if(argA < argB)
      {
            ......
      }//END of argA<argB
      else
```

```
    {
        .....
    }

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

    do
    {
        ......
    }while(1<argA);

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

    int i;
    for(i=argA; i<argB; i++)
    {
        ......
    }//END of for

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

    switch(argA)
    {
    case ARGA_CASE_A:
    {
        ......
    }
    break;
    case ARGA_CASE_B:
    {
        ......
    }
    break;
    default:
    {
        ......
    }
    }//END of switch argA
}
```

函数体内的逻辑块采用缩进形式,缩进采用 TAB 方式,数量为 4. 可通过修改编辑器的参数修改此值.

推荐 if(...){}else if(...){}else{}格式:

格式 1:

```
if(1==argA)
{
    ......
}
```

当出现常数条件判断时,亦把常数放在比较/逻辑运算符左边.

格式2:

```
if( (1==argA || 2==argB) \
      && (3==argC || 4==argD) )
{
      ......
}
```

当判断条件数量多,且逻辑复杂时,用圆括号组织逻辑,换行时把逻辑运算符放在下行首位置.

若干建议

1. 将所有变量保持在尽可能小的范围内.不到万不得已,不要声明全局变量. 若必须存在全局变量,则应增加对此全局变量的操作函数,不可直接操作全局变量. 如果变量可以声明在函数的范围内,就不要在文件范围上声明. 如果变量可以声明为循环体内的局部变量,就不要在函数范围上声明.
2. 在声明位置初始化所有变量,以明确其用途.
3. 检查所有的返回值,特别是系统调用和C库函数,合理判断errno错误代码并做合理处理.
4. 当使用固定长度缓冲区时,必须谨慎对待缓冲区空间大小,特别是字符串相关处理,一定存在一个额外字节保存'\0',表示字符串结束.
5. 尽量不使用非常用缩写，除非在WIKI上有术语解释及代码注释

当前版本: V0.4
最后修改: 2012-10-15 18:01:58
更新说明:

V0.4 2012-10-15 wuhm<wuhm@hisome.com>
1. 调整部分规范

V0.3 2008-10-21 wuhm<wuhm@hisome.com>
1. 增加函数编写规范;
2. 增加部分细节说明;

V0.2 2008-02-25 wuhm<wuhm@hisome.com>
1. 根据汪春欢意见, 函数注释内容需包括函数是否可重入信息;

V0.1 2008-02-01
初始版本讨论稿 wuhm <wuhm@hisome.com>