Q/NEOWAY

深圳市有方科技股份有限公司企业制度
（管理标准）
NEOWAY-2018-05-18

有方科技软件代码规范

目录

修 订 记 录			
版本号	修改章节	修改内容	生效日期
V1.0	全文	全文	2016/1/20
V1.1	全文	全文	2018/05/18

审 批 记 录			
版本号	修订人	审核人	批准人
V1.0	刘慧	郑小兵	郑小兵
V1.1	刘慧	刘春旭	宋仁杰

1 目的

- 统一公司软件编程风格；
- 提高软件代码的易读性、可靠性和稳定性；
- 减少软件维护成本，最终提高软件产品生产力。

2 基本原则

- 维持代码的易读、易维护；
- 保持代码的简明清晰；
- 所有代码与原始基线一致；
- 尽可能复用代码。

3 代码规范

3.1 编码风格

3.1.1 缩进

原始基线中缩进有的采用的是制表符 TAB，有的采用的是 4 个或 2 个空格；我们的缩进原则是同一个文件中缩进符只能有一种，要么是 TAB 键，要么是空格，这样不管用任何编辑器打开都是对齐的。

- 默认源文件如果是用 2 个或 4 个空格缩进，那么在该文件中修改或添加代码时，缩进就用 2 个或 4 个空格；
- 默认源文件如果是用 TAB 键缩进，那么缩进就用 TAB；
- 如果是新添源文件，则与使用基线的主流风格保持一致；以目前我们使用的高通平台 8909/9X07 平台为例，MODEM 侧均是 2 个空格为缩进，APP 侧主要是 4 个空格或一个 TAB 为缩进。

3.1.2 花括号{}

花括号在在使用上有两种风格，一种是类 Window 风格，一种是类 Linux 风格；格式如下：

类 Window 风格：

```
if (*f_pos)
```

```
{
    addr = end;
    return 0;
}
else
{
    addr = start;
    return 0;
}
```

类 Linux 风格：

```
if (*f_pos) {
    addr = end;
    return 0;
} else {
    addr = start;
    return 0;
}
```

Linux 选定的风格是左括号紧跟在语句的最后，与语句在相同的一行，而右括号要新起一行，作为该行的第一个字符；如果接下来的语句是相同语句块的一部分，那么右括号就不要单独占一行，而是与那个标识符在同一行，Android 是基于 Linux 内核的，其风格也与 linux 保持一致。

花括号使用原则是同一个文件只能有一种风格：

- 默认源文件如果使用的是类 Window 风格的花括号，则在该文件中修改或添加代码时，则应使用类 Window 风格的花括号；
- 默认源文件如果使用的是类 Linux 风格的花括号，则在该文件中修改或添加代码时，则应使用类 Linux 风格的花括号；
- 如果是新建源文件，则与使用基线的主流风格保持一致；以目前我们使用的高通平台 8909/9X07 平台为例，MODEM 使用的是类 Window 风格花括号，APP 侧使用的是类 Linux 风格花括号。

下面我们列举一下类 Linux 风格各种语法的花括的使用方法：

if 语句

例 1：

```
if (*f_pos) {
    addr = end;
```

```
    return 0;
}
```

例 2：

```
if (*f_pos) {
    addr = end;
    return 0;
} else {
    addr = start;
    return 0;
}
```

例 3：

不需要括号的语句尽量不要使用括号，例如：

```
if (*f_pos)
    return 0;
```

while 循环

例 1:

```
while (addr != end) {
    func(addr);
    addr ++;
}
```

例 2：

```
do {
    func(addr);
    addr ++;
} while (addr == end);
```

for 循环

例 1:

```
for (add = start; addr != end; addr++) {
    func1(addr);
    func2(addr);
}
```

结构体

例 1：

```
struct hello_android_dev {
    int val;
    struct semaphore sem;
```

```
    struct cdev dev;
};
```

例 2：

```
static struct file_operations hello_fops = {
    .open = hello_open,
    .read = hello_read,
    .write = hello_write,
};
```

C++中的类是一种特殊的结构体，因此编码风格与上面例子相同。

switch 语句

switch 下属的 case 标记可以与 switch 声明对齐，这样有助于减少排版缩进；如果代码不是很复杂，case 标记对比 switch 标记可以缩进；

例 1：

```
switch (animal) {
case ANIMAL_CAT:
    handle_cat();
    break;
default:
    printk(...);
    break;
}
```

例 2：

```
switch (animal) {
    case ANIMAL_CAT:
        handle_cat();
        break;
    default:
        printk(...);
        break;
}
```

上面两种写法均可以

函数体

函数不需要采用上面的书写格式，例如：

```
int func(void)
{
    /*...*/
}
```

3.1.3 空格

空格放在关键字周围:

```
if (foo)
while (foo)
for (i = 0; i < NR_CPUS; i++)
switch (foo)
```

函数名和圆括之间无空格:

```
wake_up_process(task);
typeof(*p)
size_t nlongs = BITS_TO_LONG(nbits);
```

参数前后不加空格，参数与参数之间要加空格:

```
int prio = test( int a, int b ); /*Bad style*/
int prio = test(int a, int b);
```

二元或三元操作符前后加空格:

```
int sum = a + b;
int nr = nr ? 1 : 0;
if (x < y)
mask = POLLIN | POLLRDNORM
```

一元操作符，操作符和操作数之间不加空格:

```
int len = foo.len
foo++;
--foo;
int inverted = ~mask
```

3.1.4 每行代码长度

代码的长度尽量限制在 80 个字符以内(如果刚好超过几个字符可以不断行)，如果必须要超过的，请分两行或多行进行编写，后面的行与第一行保持一个 TAB（2 个空格或 4 个空格）倍数的缩进，尽量与括号对齐:

例如:

函数定义或申明:

```
void nwy_client_voice_ind_cb
(
    client_handle_type hndl,
    uint32 msg_id,
```

```
    void *ind_struct,
    uint32 ind_len
)
void nwy_client_voice_ind_cb(client_handle_type hndl, uint32 msg_id,
                             void *ind_struct, uint32 ind_len)
void nwy_client_voice_ind_cb(client_handle_type hndl,
                             uint32 msg_id,
                             void *ind_struct,
                             uint32 ind_len)
```

函数调用：

```
nwy_test_func1(hndl,
               MCM_VOICE_COMMAND_REQ_V01,
               &req_msg,
               resp_msg,
               NULL,
               &token_id);
nwy_test_func1(hndl, MCM_VOICE_COMMAND_REQ_V01, &req_msg,
               resp_msg, NULL, &token_id);
```

上述换行方式均可以，但是同一个源文件，尽量保持换行风格一致。

3.1.5 命名规范

基本原则仍然是保持与原代码风格一致；命名场景特别多，针对公司目前的主流平台，我们暂时只作下面的一些要求。

3.1.5.1 文件命名

- 新添加文件名一律以 nwy 开关；

3.1.5.2 函数命名

- 函数一律采用小写字母和下划线组合的方式命名；
- 函数名的意思要简洁准确；
- 新添的函数，函数名要以 nwy_开头，如 `void nwy_voice_call_request()`。

3.1.5.3 变量命名

- 变量一律采用小写字母和下划线组合的方式命名；
- 变量名尽量简短，意思表达简洁准确，且不应超过 20 个字符；
- 变量名一律不要以 nwy 开头。

3.1.5.4 结构体命名

- 结构体名称一律采用小写字母和下划线组合的方式命名；

- 新建结构体或枚举类型，要以 nwy_开头，如果加上了 typedef 关键字，结构体名末尾加上_t。如：

```
typedef enum nwy_voice_ind_type{
    NWY_VOICE_CALL_IND = 0, /**< Voice call msg */
    NWY_VOICE_MUTE_IND = 1, /**< Voice mute msg */
    NWY_VOICE_DTMF_IND = 2, /**< Voice dtmf msg */
}nwy_voice_ind_type_t;
```

3.2 变量使用规范---建议

- 要尽可能少的使用全局变量；
- 作用域为同一文件的全局变量，应在变量前加上 static；
- 变量在使用前，应该初始化；
- 数组在每次使用前，都应检查是否应执行清空操作。

3.3 注释

代码注释原则上要求全部使用英文，如果默认的源文件使用中文，则也可以使用中文。

3.3.1 默认源文件或函数添加注释

如果在基线中已有的源文件或函数中添加，修改或删除代码，则遵循以下注释规范。

3.3.1.1 增加代码

格式：

```
/*Begin: Add + Author +  Reason + Date*/
code
/*End: Add + Author +  Reason + Date*/
```

范例：

```
/*Begin: Add by liuhui for/to client init in 2018.05.16*/
nwy_client_init(&client_hndl);
/*End: Add by liuhui for/to client init  in 2018.05.16*/
```

3.3.1.2 修改代码

格式：

```
/*Begin: Modify + Author +  Reason + Date*/
```

```
code
/*End: Modify + Author +  Reason + Date*/
```

范例：

```
/*Begin: Modify by liuhui for/to fix the bug 62349 in 2018.05.16*/
/*if (x <= y)
    mask = POLLIN;*/
if (x < y)
    mask = POLLIN | POLLRDNORM;
/*End: Modify by liuhui for/to fix the bug 62349 in 2018.05.16*/
```

3.3.1.3 删除代码

格式：

```
/*Begin: Delete + Author +  Reason + Date*/
/*code*/
/*End: Delete + Author +  Reason + Date*/
```

范例：

```
/*Begin: Delete by liuhui for/to fix the bug 62349 in 2018.05.16*/
/*if (x <= y)
    mask = POLLIN;*/
/*End: Delete by liuhui for/to fix the bug 62349 in 2018.05.16*/
```

3.3.1.4 注意事项

- 不要叠加注释，只需要保留最新的注释即可，如下是禁止的；

```
/*Begin: Modify by wsj to change gpio in 2018.05.16*/
/*Begin: Modify by liuhui fo to enable gpio interrupt in 2018.05.16*/
```

- 如果宏控添加了注释，则宏控所包含的代码是可以不用加注释的，如：

```
/*Begin: Add by liuhui for douple tap in 2018.05.16*/
#define CONFIG_DOUBLE_TAP
/*End: Add by liuhui for douple tap in 2018.05.16*/

#ifdef CONFIG_DOUBLE_TAP
static void bma2x2_double_tap_enable(struct bma2x2_data *data)
{

}
#endif
```

3.3.2 新增源文件或函数添加注释

如果是在我们自己添加的源文件或函数中修改代码，则遵循以下注释规范。

3.3.2.1 新增源文件

新增的源文件，均以如下格式开头，虚线框的长度建议为 80 个字符左右。

```
/*====*====*====*====*====*====*====*====*====*====*====*====*
   Copyright (c) 2017 Neoway Technologies, Inc.
   All rights reserved.
   Confidential and Proprietary - Neoway Technologies, Inc.
   Author: liuhui
   Date: 2018.05
*====*====*====*====*====*====*====*====*====*====*====*====*====*/
```

3.3.2.2 新增函数

函数注释不是必须的，建议对一些重要的函数按照如下 2 种格式添加注释：

注释 1：

虚线框和长度建议为 70 个字符左右。

```
/*************************************************************
 FUNCTION nwy_get_adc
/*************************************************************
@Desc:  Get adc value of the channel
@Para:  fd - file handle
        port - the Channel of Adc
@Return: 0 - True, -1 - Error
*************************************************************/
```

注释 2：

```
/**
 * kobject_set_name - Set the name of a kobject
 * @kobj: struct kobject to set the name of
 * @fmt: format string used to build the name
 * @Return: 0 - True, -1 - Error
 * This sets the name of the kobject.  If you have already added the
 * kobject to the system, you must call kobject_rename() in order to
 * change the name of the kobject.
 */
int kobject_set_name(struct kobject *kobj, const char *fmt, ...)
```

下面的是 Linux 内核的函数注释风格，黄色部分为可选，Linux 内核建议选用注释 2；其它场景，两种注释风格为可选，但要求保证同一源文件中只能使用一种注释风格。

3.3.2.3 代码注释

代码注释非常重要，但注释必须按照正确的方式进行。一般情况下，我们应该描述我们的代码要做什么和为什么要做，而不是具体通过什么方式实现的。怎么实现由代码本身展现。此外，注释不应用包含谁写了哪个函数、修改日期和其他那些琐碎而无实际意义的内容。这些信息应该集中在文件最开头的地方。

```
/* This function is mostly/only used for network interface.
 * Some hotplug package track interfaces by their name and
 * therefore want to know when the name is changed by the user. */
kobject_uevent_env(kobj, KOBJ_MOVE, envp);
```

在注释中，重要信息常常以”XXX：”开头，而 bug 通常以”FIXME: “开头，就像：

```
/*FIXME: We assume dog == cat witch may not be true in the future*/
```

3.4 宏控

3.4.1 基本原则

- 所有宏名必须为大写加下划线；
- 宏控中不要出现项目名称；
- 去掉宏之后，编译正常；
- 修改基线中的源代码代码尽量使用宏控进行修改；

不添加宏控直接修改代码要满足以下 2 个条件：

1) 修改对平台上的所有项目有效或对其它项目无影响
2) 不是添加新的功能

3.4.2 #ifdef、#if defined 与#if

当判断单个宏是否有定义时，#ifdef 等价于#if defined，两者均可；例 1：

```
#ifdef XXX
#else
#endif
#if defined XXX
#else
#endif
```

当判断比较复杂的宏，建议使用#if defined；例 2：

```
#if defined (XXX) || defined (YYY)
#elif defined (KKK)
#endif
```

当判断一个宏的值是否满足要求，建议使用#if；例 3：

```
#define XXX 0
#define YYY 4
#if (XXX) || (YYY > 3) || defined (YYY)
#elif defined (KKK)
#endif
```

上面例子要求 XXX 与 YYY 必须被定义成一个数值，否则编译会出错；#if (XXX) 要求 XXX 为非 0 值才返回真，#if (YYY>3)要求 YYY 大于 3 才返回真；从上面例子可以看出#if 和#if defined 是可以混合使用的。

3.4.3 宏控缩进

所有宏定义都没有缩进，直接顶格开始写；

```
#if defined (FEATURE_JSR_BMA250_AUTO_CLB)
        this_client = client;
        acc_cali_bma2x2 = data;
        gbma250_data_offset.x = 0;
        gbma250_data_offset.y = 0;
        gbma250_data_offset.z = 0;
#endif
```

3.4.4 kernel 宏规范

3.4.4.1 kernel 宏命名

- 板级配置宏控必须是 CONFIG_BOARD_JSR_<板级配置>，这里可能多个项目共同使用同一个板级配置；

  如：CONFIG_BOARD_NWY_E45T=y

- 驱动宏控必须是 CONFIG_NWY_<驱动类型>_<IC 名>；

  如：CONFIG_NWY_SENSORS_LTR559=y

- 功能宏控必须是 CONFIG_NWY_<驱动类型>_<功能/差异>，驱动类型可选：

  如：CONFIG_NWY_ DOUBLE_BATTERY=y

3.4.4.2 kernel 宏添加位置

kernel 宏被添加在如下文件中：

64 位系统为：

kernel/arch/arm64/configs/msm_XXX_defconfig 和 msm-perf_XXX_defconfig

32 位系统为：

kernel/arch/arm/configs/msm_XXX_defconfig 和 msm-perf_XXX_defconfig

注意，所有新添加的宏控均需要放在上面两个文件中，带 perf 的对应的是 user 版本，不带 perf 的对应的是 eng 版本；要求将这些宏全部放在文件的末尾。

3.4.5 lk 宏规范

3.4.5.1 lk 宏命名

宏控命名格式：FEATURE_NWY_<功能/差异>；

如：DEFINES += FEATURE_NWY_ SUPPORT_CHARGER_SCREEN=1

3.4.5.2 lk 宏添加位置

所有的 lk 宏均被添加到如下文件中：

bootable/bootloader/lk/target/msm8916/nwy_feature.mk

每个项目所需要的宏必须被添加到该项目宏的下面，如下：

```
ifeq ($(TARGET_PRODUCT), D6000_ROM)
DEFINES += FEATURE_NWY_D6000_ROM
DEFINES += FEATURE_NWY_LCD_OTM1906C_FHD_VIDEO_MODE = 1
DEFINES += FEATURE_NWY_LCD_R69338_FHD_VIDEO_MODE = 1
endif
```

3.4.6 应用层宏规范

3.4.6.1 应用层宏命名

在应用层，如 MDM 平台的应用程序，Android 系统的 system，externel，framework 和 vendor 等文件夹下有很多用 C/C++编写的代码，这些地方的宏我们采用统一的规则。

其命名规则保持如下形式，其中模块名可选：

FEATURE_NWY_<模块>_<功能/差异>；

在 Android 系统的 Andoird.mk 中，宏定义形式一般有如下两种情形：

如：LOCAL_CFLAGS += -DFEATURE_NWY_SENSORS_ XXX

等价于#define FEATURE_NWY_ SENSORS_XXX

如：LOCAL_CFLAGS += -DFEATURE_NWY_ SENSORS_XXX=abc

等价于#define FEATURE_NWY_ SENSORS_XXX abc

在 MDM Linux 平台的 Makefile.mk 中，宏定义形式一般有如下两种情形：

如：**nwy_fota**_CPPFLAGS += -DFEATURE_NWY_ XXX

等价于#define FEATURE_NWY_ XXX

如：**nwy_fota**_CPPFLAGS += -DFEATURE_NWY_XXX=\"abc"\

等价于#define FEATURE_NWY_ XXX abc

3.4.6.2 应用层宏添加位置

在 Android 系统中，添加宏控的步骤如下：

1) 首先在 device/jsr/D6000_ROM/BoardConfig.mk 或者 D6000_ROM.mk 中添加一个 MAKEFILE 变量，这个名字与下面 Android.mk 中定义的宏只少一个 FEATURE 字符，并且该变量的值统一设置为 ture

如：NWY_CAMERA_LIST_XXX := true

2) 然后在对应模块的 Android.mk 中添加真正的宏，如下：

```
ifeq ($(JSR_CAMERA_LIST_XXX), true)
LOCAL_CFLAGS += -DFEATURE_JSR_CAMERA_LIST_XXX
else ifeq ($(JSR_CAMERA_LIST_YYY), true)
LOCAL_CFLAGS += -DFEATURE_JSR_CAMERA_LIST_YYY
else ifeq ($(JSR_CAMERA_LIST_ZZZ), true)
LOCAL_CFLAGS += -DFEATURE_JSR_CAMERA_LIST_ZZZ
endif
```

在 MDM 平台的 Linux 应用程序中，添加宏控的步骤如下：

1) 将应用程序所在模块的 Makefile.am 文件抽取到指定项目的 NWY_CUSTOM/Project/中；

2) 然后在抽取出来的 Makefile.am 文件中添加宏控，如下：

nwy_fota_CPPFLAGS += -DFEATURE_NWY_DISABLE_REBOOT

nwy_fota_CPPFLAGS += -DFEATURE_NWY_MODEL=\"N720W_PCIE_HK"\

3.4.7 Modem 宏控规范

3.4.7.1 Modem 宏命名

宏控命名格式：FEATURE_NWY_<模块>_<功能/差异>； 模块名可选；

如：#define FEATURE_NWY_AT_SN

3.4.7.2 Modem 宏添加位置

某个项目需要的宏被添加在对应模块的如下文件中：

<modem_proc>/build/ms/NWY_CUSTOM/nwycust_xxx.h

如果是所有项目共有的宏，则将该宏添加在如下文件中：

<modem_proc>/build/ms/nwycust_common.h

4 注意事项

- 本规范对公司主流平台有效，部分小众平台因风格差异较大，与基线保持一致即可；
- 本规范自生效日期起，对以后新编写和修改的代码有约束力，对于由开发工具自动生成的代码可以不约束；
- 本文档是公司软件开发的标准化文档，其它规范标准不再有效，以后如果需要修改或扩展规范，仅对本文档进行修订。