项目开发编码规范

V1.1

惠尔丰电子（北京）有限公司

2012 年 3 月

项目开发编码规范

文档信息

日期	版本	修订说明	修订人
2012.3.26	0.1	文档建立	姜楠
2012.4.3	1.0	初版	姜楠
2012.12.1	1.1	格式整理,删除释放流程及统一开发环境文档	姜楠

目　录

项目开发编码规范（C 语言）

1 前言

本文档规定应用程序中的编程规则。

目的 :为使参与项目的开发人员更容易了解项目中的代码、弄清程序的状况。使新的参与者可以很快的适应环境，防止部分参与者出于节省时间的需要，自创一套风格并养成终生的习惯，导致其它人在阅读时浪费过多的时间和精力。而且在一致的环境下，也可以减少编码出错的机会。

问题：由于每个人的标准不同，所以需要一段时间来适应和改变自己的编码风格，暂时性的降底了工作效率。从使项目长远健康的发展以及后期更高的团队工作效率来考虑暂时的工作效率降低是值得的，也是必须要经过的一个过程。标准不是项目成功的关键，但可以帮助我们在团队协作中有更高的效率并且更加顺利的完成既定的任务。

2 程序注释规则：

注释的作用是帮助对程序的阅读理解，而不是文档。注释语言必须准确、易懂、简洁。除非可以使用流利准确的英文描述，否则应使用中文注释，以便于日后的维护。

2.1 文件头注释：

文件头注释要求包含以下要素内容：

文件模块功能、作者信息、编写日期、修改日期、版本信息

例如：

```
/*****************************************************
 * FILE NAME:APPFILE.C
 * MODULE NAME:APPFILE
 * PROGRAMMER:Shaozhen Lin
 * DESCRIPTION:实现 POS 与文件系统的相关处理
 * REVISION:01.00 12/22/2012
 *****************************************************/
```

2.2 函数注释

函数注释要求包含以下内容：

函数名、函数功能、作者、参数说明、返回值说明、版本及修订说明等

例如：

```
/*****************************************************
 * FUNCTION NAME:Credit_Sale
 * DESCRIPTION:实现消费交易功能
 * PROGRAMMER:JiangNan
 * PARAMETERS:dummy-函数调用方式
 * RETURN:BANK_OK-成功
 *              BANK_FAIL-失败
 * REVISION:01.00 12/22/2012  增加交易多界面
 *****************************************************/
```

2.3 代码注释

函数体外注释符采用“/*…*/”，不使用“//…”。

函数体内注释符采用“//”，不使用“/*…*/”。

注释放置在被注释代码的上方或右方，不可放在下面，如放于上方则需与其上面的代码用空行隔开。

逻辑复杂的段落代码的上面要求有对段落处理的注释信息，分支语句也同样处理。

注释与代码同样样对齐。

边写代码边注释，修改代码同时修改相应的注释，以保证注释与代码的一致

性。不再有用的注释要删除。

全局变量要有较详细的注释，包括对其功能、取值范围、哪些函数或过程存取它以及存取时注意事项等的说明。

结构定义、宏定义、常量定义必须有注释，除非名称已完全体现其使用意义。

3 程序命名规则

3.1 变量命名规则：

1) 所有的变量均由一个或多个英文单词组成，用于表明该变量的用途，变量的命名应易于理解，增加程序的可读性。
2) 每个单词的首字母均大写，避免使用含义不清的缩写单词
3) 全局变量名以‘g’修饰，静态全局变量以‘m’修饰，指针变量以‘p’修饰（由于“匈牙利”命名法过于烦琐，其他匈牙利命名法规则不建议引入）

   例:int gFixParam;

   表示该变量为全局变量，含义为静态参数。

   int mFontSize;

   表示该变量为静态全局变量，含义为字体大小
4) 避免指针变量与变量重名，避免标识符完全相同的局部变量和全局变量
5) 常量、宏、自定义结构体采用大写字母和下划线组合的方式

3.2 函数命名规则：

1) 函数名称定义格式为“模块名_函数名（）”，一般情况下模块名即存放该模块的文件名。所有的函数名均由一个或多个英文单词组成。用于表明该函数的用途。每个单词的首字母均大写。
2) 内部函数以‘i’为前缀，表示此函数只供模块内部调用，不对外提供接口
3) 函数返回值类型及参数类型应显式声明，以便编译器语法检查
4) 函数的参数命名参考变量命名规则。
5) 请不要自己定义缩写,除非是标准缩写 如 ASCII,DES 等。
6) 如果参数是缩写，请采用大写字母。

7) 输入参数与输出参数顺序统一，按照先输入参数后输出参数定义函数参数

例如：

sint AppUtil_GetLast(char *FileName,void *TranData, int Record_Len)

表示 AppUtil 模块中读取文件最后一条交易记录的函数，返回值为 sint 型，输入参数为 FileName 输出为 TranData

4 宏定义规则

常量、返回值等应通过通过宏定义赋值，便于统一修改。

宏的定义包括宏变量和宏函数两种形式。

1) 宏变量的定义采用全部大写字母组成的单词来构成，单词中间通过下划线分隔。

例如：

```
#define SHARE          0
#define EXCLUSIVE      1
```

2) 宏函数的定义规则和函数的定义规则相同。注意用宏定义表达式时，要使用完备的括号。

例如：

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

如果定义成

#define RECTANGLE_AREA(a,b) (a*b)在传入值为表达式时则会出问题。

5 程序风格

1) 所有的程序应由头文件和程序文件构成。
2) 在头文件中，定义本文件所引出的函数和数据接口。
3) 所有并不引出的函数和数据作为内部函数在程序文件中定义。
4) 为避免多次重复包含头文件，在头文件中应利用宏定义来处理。
5) 需要重复使用的功能通过函数模块实现
6) 避免超长的参数列表
7) 单个函数实现的功能尽量简单，一个函数只做一件事情

5.1 程序文件命名规则

工程文件及模块代码文件名的命名要求尽可能使用有意义英文表示,但长度不得超过 8 字节，若超过 8 字节，可采用易于理解的缩略语表示。对于各英文单词,首字母必须大写,对于特定简称,需要全部采用大写,例如:“PIN”、“ATM”等。对于代码模块，其文件名将作为模块函数名的一部分使用。

示例：ComMngr, CUP_BJ, Credit, ISO8583。

终端程序的各类文件，按照如下定义使用文件扩展名：

- “.C”，代码文件；
- “.H”，函数声明文件，可包含其他类及结构定义、宏定义等；

5.2 头文件的结构

头文件由四部分组成，如图 1 - 1 所示

- 版本声明
- 预处理块
- 结构声明
- 函数声明

- 为了防止头文件被重复引用，应当用 ifndef/define/endif 结构产生预处理块。
- 用#include <filename.h> 格式来引用标准库的头文件（编译器将从标准库目录开始搜索）。
- 用#include “filename.h” 格式来引用非标准库的头文件（编译器将从用户的工作目录开始搜索）。
- 头文件中只存放“声明”而不存放“定义”
- 不提倡使用全局变量,尽量不要在头文件中出现象 extern int value 这类声明。

```
/* 版本声明 */
#ifndef  CREDIT_H
#define  CREDIT_H

#include “bank.h”

typedef struct
{
```

```
};

sint Credit_Logon(void *dummy)

#endif
```

图 1 - 1　头文件结构示例

5.3 代码文件的结构

代码文件由如下部分组成，如图 1 - 2 所示

- 包含的头文件声明
- 预编译处理语句：如果有则添加；
- 文件内部使用结构定义，宏定义；
- 常量定义；
- 全局变量定义；
- 局部变量定义；
- 私有函数头声明；
- 公共函数；
- 私有函数；

```
/*********************************************************************
 * FILE NAME:       PLU.C
 * PROGRAMMER:   CL.
 * DESCRIPTION:   "Price-Look-Up" MMI functions.
 * REVISION:        01.00 11-06-96.
*********************************************************************/
          /*=============================================*
           *          I N T R O D U C T I O N            *
           *=============================================*/
/* 介绍内容 */
          /*=============================================*
           *              I N C L U D E S                *
           *=============================================*/
#include <stdio.h>
#include "project.def"

          /*=============================================*
           *            D E F I N I T I O N S            *
```

```
        *============================================*/
/* 结构及宏定义、常量定义 */
        /*============================================*
         *           PUBLIC DATA                      *
         *============================================*/
/* 全局变量定义 */
        /*============================================*
         *           PRIVATE DATA                     *
         *============================================*/
/* 局部变量定义 */
        /*============================================*
         *  INCLUDE PRIVATE FUNCTIONS                 *
         *============================================*/
/* 私有函数声明 */
         /*===========================================*
          *  PUBLIC    FUNCTIONS                      *
          *===========================================*/
/* 全局函数代码 */

        /*===========================================*
                *   PRIVATE FUNCTIONS                 *

        *===========================================*/
/* 私有函数代码 */
```

图 1 - 2　代码文件结构示例

5.4 排版

空行起着分隔程序段落的作用。空行得体（不过多也不过少）将使程序的布局更加清晰。

- 文件中各段落间需要用空行隔开，空行可占总代码行的 8 - 16%。
- 在每个结构声明之后、每个函数定义结束之后都要加空行。
- 在一个函数体内，逻辑上密切相关的语句之间不加空行，其它地方应加空行分隔。

```
while (condition1)
{
    statment1;
```

```
    /* 空行 */
    if (condition2)
    {
        statment2;
    }
    else
    {
        statment3;
    }
    /* 空行 */
    statment4;
}
```

图 2 - 1

5.4.1 代码行

1) 一行代码只做一件事情，如只写一条语句。这样的代码容易阅读，并且方便于写注释。
2) if、for、while、do 等语句自占一行，执行语句不得紧跟其后。不论执行语句有多少都要加{}。这样可以防止书写失误。

风格良好的代码	风格不良的代码

x = a + b;
y = c + d;
z = e + f;
	X = a + b; y = c + d; z = e + f;

if (width < height)
{
dosomething();
}
	if (width < height) dosomething();

for (initialization; condition; update)
{
dosomething();
}
/* 空行 */
other();
	for (initialization; condition; update)
	dosomething();
	other();

5.4.2 缩进对齐

1) 程序块要采用缩进风格编写，缩进为 1 个 TAB，不使用空格。缩进处理包括： 函数内代码、if 语句、switch 语句、所有循环语句、其他{}内部处理等。（尽量采用可自动缩进的编辑器以提高效率）
2) 程序的分界符‘{’和‘}’应独占一行并且位于同一列，同时与引用它们的语句左对齐。
3) { }之内的代码块需要缩进。
4) 为便于识别，预编译指令代码行不缩进。

5.4.3 代码行内的空格

1) 关键字之后要留空格。象 const、virtual、inline、case 等关键字之后至少要留一个空格，否则无法辨析关键字。象 if、for、while 等关键字之后应留一个空格再跟左括号‘(’，以突出关键字。
2) 函数名之后不要留空格，紧跟左括号‘(’，以与关键字区别。
3) ‘(’向后紧跟，‘)’、‘,’、‘;’向前紧跟，紧跟处不留空格。
4) ‘,’之后要留空格，如 Function(x, y, z)。如果‘;’不是一行的结束符号，其后要留空格，如 for (initialization; condition; update)。
5) 赋值操作符、比较操作符、算术操作符、逻辑操作符、位域操作符，如“=”、“+=” “>=”、“<=”、“+”、“*”、“%”、“&&”、“||”、“<<”，“^”等二元操作符的前后应当加空格。
6) 一元操作符如“!”、“~”、“++”、“--”、“&”（地址运算符）等前后不加空格。
7) 象“[ ]”、“.”、“->”这类操作符前后不加空格。
8) 对于表达式比较长的 for 语句和 if 语句，为了紧凑起见可以适当地去掉一些空格，如 for (i=0; i<10; i++)和 if ((a<=b) && (c<=d))

void Func1(int x, int y, int z); /* 良好的风格 */ void Func1 (int x,int y,int z); /* 不良的风格 */
if (year >= 2000) /* 良好的风格 */ if(year>=2000) /* 不良的风格 */
if ((a>=b) && (c<=d)) /* 良好的风格 */ if(a>=b&&c<=d) /* 不良的风格 */
for (i=0; i<10; i++) /* 良好的风格 */

```
for(i=0;i<10;i++)   /* 不良的风格 */
for (i = 0; I < 10; i ++) /* 过多的空格 */
```

```
x = a < b ? a : b; /* 良好的风格 */
x=a<b?a:b;      /* 不良的风格 */
```

```
int *x = &y; /* 良好的风格 */
int * x = & y; /* 不良的风格 */
```

```
array[5] = 0; /* 不要写成array [ 5 ] = 0; */
a.Function(); /* 不要写成a . Function(); */
b->Function(); /* 不要写成 b -> Function(); */
```

5.4.4 长行拆分

1) 代码行最大长度宜控制在 80 个字符以内。代码行不要过长，否则眼睛看不过来，也不便于打印。
2) 长表达式要在低优先级操作符处拆分成新行，操作符放在新行之首（以便突出操作符）。拆分出的新行要进行适当的缩进，使排版整齐，语句可读。

```
if ((very_longer_variable1 >= very_longer_variable12)
  && (very_longer_variable3 <= very_longer_variable14)
  && (very_longer_variable5 <= very_longer_variable16))
{
    dosomething();
}
```

```
virtual CMatrix CMultiplyMatrix (CMatrix leftMatrix,
                                 CMatrix rightMatrix);
```

```
for (very_longer_initialization;
    very_longer_condition;
    very_longer_update)
{
    dosomething();
}
```

5.4.5 易读性

1) 注意运算符的优先级，并用括号明确表达式的操作顺序，避免使用默认优先级。
2) 避免使用不易理解的数字，用有意义的标识来替代。

3) 不要使用难懂的技巧性很高的语句，除非很有必要时。高技巧语句不等于高效率的程序，实际上程序的效率关键在于算法。
4) 不使用如下语句：if (0 == x)， 改为 if (x == 0)。
5) 虽然防止了写成 if (x = 0)的手误，但代码不易读。
6) 不使用易引起歧义的语句：if ((x == 0) && (f(x++) > 1))，应将 x++拿出来写。

6 良好的编程习惯

1) 关注编译器的 Warning 信息，认真分析处理每一个 Warning，调试完成后应当消灭全部 Warning
2) 简洁的编码风格：
   - 函数代码一般不超过 200 行（注释和空行除外）。
   - 降低函数间的耦合度，并提高函数的独立性以及代码可读性、效率和可维护性。
   - 为重复执行的短小代码单独编制函数或宏，如求最大值。
3) 安全的使用变量：
   - 变量使用前注意初始化
   - 减少不必要的默认数据类型转换或强制数据类型转换（特别注意将可见字符定义为 char 型，HEX 数据、BCD 数据定义为 byte 型）
   - 禁止将函数的参数作为工作变量使用，使用临时变量，在操作中对其值进行修改，在函数退出前赋值。
4) 编写函数时注意返回值定义，调用函数时注意判断函数返回值，可以快速定位问题及增强程序健壮性。
5) 开发过程中尽量使用 APPLIB 以及系统库中已有函数实现所需功能，不要重复开发，增加代码复杂度。
6) 对已有项目的维护需要做好修订记录，将所做修改记入 revision.txt 文件，并且在代码中标注修订人及修订时间。
7) 避免在代码中加入过多的调试信息、调试代码段，是程序难于阅读，在程序调试完成后建议删除调试代码
8) 保持程序清洁，不使用的变量、结构体成员及时清除，模板移植时注意修改个性信息，应用简称变量名称应注意更新。
9) 尽量使用通用的变量及宏命名，方便模板移植：
   建议使用

```
#define LOGOVFT              "F:logo.vft"      //待机界面
避免使用
#define CCBLOGOVFT           "F:CCB.vft"   //待机界面
```

项目工程规范

请根据《ADD 部门_VFI 开发环境指南》在专门的目录安装开发环境、公共库等。工程的建立可采用 eVo_DTK 中提供的 Wizard.exe 向导建立或者在其他已有工程基础上修改(注意：如果是在其他工程基础上修改请删除掉当前工程不需要的文件)

1 工程目录结构

对每一个工程，工程相关文件直接置于工程目录下，但必须采用专门的子目录统一管理该工程的文档、源代码、编译过程文件、调试文件、待释放的最终文件等。

工程各目录说明如下：

PROJECT	工程目录，存放 smk(makefile)/fst(签名工具配置)文件
DOC	存放本工程相关文档(需求、接口、问题记录等等)
SRC	存放所有源代码。其下可根据需要细分模块子目录
OBJ	存放编译过程中的目标文件。
RELEASE	存放本版本释放的所有应用、参数及释放说明。根据需要可细分成不同机型的目录。每一个机型的目录下都应该是一套完全的文件，除各释放文件外，该目录中要求包含至少一个文档文件“RELEASE.TXT”，该文件记录本工程历史版本修订情况。另外，根据工程释放的实际复杂度，可编制文档“README.TXT”，该文件对被释放的各文件作说明，同时说明使用注意事项等。
LIB	子目录存放本工程用到的所有外部开发的库文件。根据需要可细分子目录。
INCLUDE	子目录存放引用外部的文件，包括头文件及其他数据文件等。根据需要可细分子目录，如将数据文件集中置于下一级子目录中。

2 工程版本号定义

版本号管理：

- 版本号管理由 SVN 统一管理,公司内部释放版本号采用”XX.YY.ZZ”格式,ZZ 为小版本号、YY 为大版本号、XX 为拓展使用。如: 01.00.00.
- 版本号更新记录,由 revision 详细说明.

应用程序的版本在 SMK 文件中定义，编译时自动生成版本信息文件，应用在启动时读取版本信息文件，将版本数据放入到应用信息结构表中，并在子应用注册时传递给主应用。

版本号分为 2 种：

1.应用释放版本号：客户定义的版本号，一般用于显示打印等

2.管理版本号：内部管理版本号，用于 SVN 版本管理