命名规则：
匈牙利命名法是一种编程时的命名规范。基本原则是：变量名=属性+类型+对象描述
具体操作：
A) 函数： 函数名应能体现该函数完成的功能，关键部分应采用完整的单词，辅助部分若太长可采用缩写，缩写应符合英文的规范。每个单词的第一个字母大写。如：ShowPoints，CtrlDestBoard，SendResetMsg 等。

B) 变量： 变量的命名规则部分采用匈牙利命名规则(鼓励完全使用匈牙利名规则)。
变量的第一个或前两个字母小写，表示其数据类型，其后每个词的第一个字母大写。

前缀	含义		前缀	含义
a	数组		n	short int
b	BOOL		np	短指针
by	BYTE		p	指针
c	char		l	LONG
cb	字节记数		lp	长指针
cr	颜色参考值		s	串
cx,cy	短型（x,y长度的记数）		sz	以零结尾的串
dw	DWORD		tm	文本
fn	函数		w	WORD
h	HANDLE		x,y	短型（x或y的坐标）
i	int		g_	全局变量
m_	类的数据成员		uc	unsigned char

如 iCurrentValue，uTransitionCount 等。对于其他复合类型或自定义类型，请用适当的前缀

C) 结构：结构的定义有两个名称，一个是该结构的类型名，一个是变量名。按照C 语言的语法，这两个名称都是可选的，但二者必有其一。要求写类型名，类型名以tag 做前缀。struct后的类型名有tag前缀，自定义的结构名称一律用大写字母，下面是一个例子：

```
struct  tagVBXEVENT
{
    HCTL    hControl;
    HWND    hWindow;
    int         nID;
    int         nEventIndex;
    LPCSTR  lpEventName;
    int         nNumParams;
    LPVOID      lpParamList;
}veMyEvent;
tagVBXEVENT veMyEvent[MAXEVENTTYPE], *lpVBXEvent;
```

对于程序中常用的结构，希望能使用 typedef 定义，格式如下：

```
typedef tagMYSTRUCT
{
    struct members ... ...
}TMYSTRUCT,* PTMYSTRUCT,FAR * LPTMYSTRUCT;
#define MYSTRUCTSIZE      sizeof ( TMYSTRUCT );
```

1。模块描述
模块是为了实现某一功能的函数的集合，文件名使用缺省的后缀，在每一模块的开头应有如下的描述体：

```
/*****************************************************************************
***
* PROJECT CODE ： 项目代号或名称 *
* CREATE DATE ： 创建日期 *
* CREATED BY ： 创建人 *
* FUNCTION ： 模块功能 *
* MODIFY DATE ： 修改日期 *
* DOCUMENT ： 参考文档 *
* OTHERS ： 程序员认为应做特别说明的部分，如特别的编译开关 *
*****************************************************************************
**/
```

不同的修改人应在修改的地方加上适当的注释，包括修改人的姓名。另外，如有必要，要注明模块的工作平台，如单板 OS、DOS、WINDOWS 等。注明适用的编译器和编译模式。

2。函数描述
函数是组成模块的单元，一般用来完成某一算法或控制等。在每一函数的开头应有如下的描述体：

```
/*****************************************************************************
***
* FUNCTION NAME：函数名称 *
* CREATE DATE ： 创建日期 *
* CREATED BY ： 创建人 *
* FUNCTION ： 函数功能 *
* MODIFY DATE ： 修改日期 *
* INPUT ： 输入参数类型(逐个说明) *
* OUTPUT ： 输出参数类型(逐个说明) *
* RETURN ： 返回信息 *
*****************************************************************************
**/
```

可选的描述有：

```
* RECEIVED MESSAGES：收到的消息 *
* SENT MESSAGES ： 发送的消息 *
* DATABASE ACCESS ： 存取的数据库 *
* CALLED BY ： 该函数的调用者 *
* PROCEDURES CALLED：调用的过程 *
* RECEVED PRIMITIVES : 收到的原语 *
* SENT PRIMITIVES : 发送的原语 *
```

及其它程序员认为应有的描述。标题可以只大写第一个字母。例如：Function Name：

3。头文件描述

头文件：头文件一般包括了数据结构的定义，函数原形的说明，宏定义等，不许包含函数体和变量实体，文件名使用缺省的后缀.h，不使用类似.DEF 等非标准的后缀名，头文件的开始可包括如下的注释：

```
/*****************************************************************************
**
* CREATE DATE：创建日期 *
* CREATED BY ：创建人 *
* MODIFIED BY ：修改人 *
* USED BY ：由哪些模块使用 *
*****************************************************************************
**/
```

为了避免重编译，应加上条件编译语句，如文件headfille.h 应包含下列语句：

```
#ifndef __HEADFILE_H
#define __HEADFILE_H
.
.
.
#endif
```

书写风格

A) 函数：函数的返回类型一定要写，不管它是否默认类型，函数的参数之间应用一逗号加一空格隔开,若有多个参数，应排列整齐。例如：

```
int SendResetMsg( PTLAPENTITY pLAPEntity, int iErrorNo )
{
    int iTempValue;
    .
    .
    .
}
```

函数的类型和上下两个括号应从第一列开始，函数的第一行应缩进一个**TAB**，不得用空格缩进。(按大多数程序范例，**TAB** 为四个字符宽，我们规定：**TAB** 为四个字符宽。)

B) 语句：循环语句和 **if** 语句等块语句的第一个大括号‘**{**’ 另起一行

```
for ( count = 0 ; count < MAXLINE ; count++ )
{
    if ( (count % PAGELINE) == 0 )
    {
        .
        .
        .
    }
    .
    .
}
```

复杂表达式(两个运算符以上，含两个)必须用括号区分运算顺序，运算符的前后应各有一空格，习惯写在一行的几个语句(如IF 语句)，中间应有一空格，其它语句不鼓励写在同一行。

```
myStruct.iFirstNumber    = 0;
myStruct.lSecondNumber   = 1;
myStruct.pThePoint       = NULL;
```

C) 常量：常量一般情况下可用宏定义，用大写的方式，单词之间用下划线隔开 如：

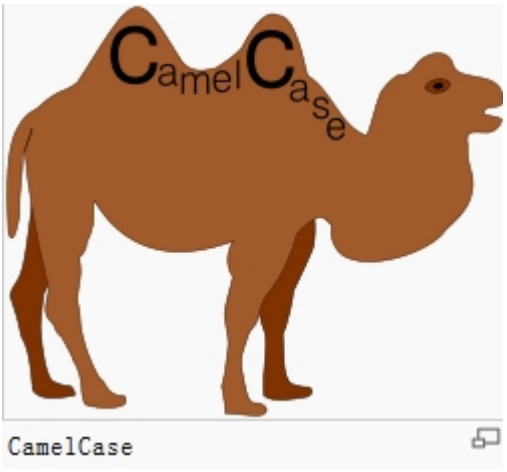
```
#define MAX_LINE 100
#define PI 3.1415926
```

不鼓励在程序中出现大量的数字常数。

注：对于一些有必要说明的缩写，可以在模块描述内加以说明。

附录：

驼峰式大小写（Camel-Case，Camel Case，camel case），电脑程序编写时的一套命名规则。当变量名和函数名称是由二个或多个单字链接在一起，而构成的唯一识别字时，利用“驼峰式大小写”来表示，可以增加变量和函数的可读性。“驼峰式大小写（Camel-Case）一词来自 Perl 语言中普遍使用的大小写混合格式，而 Larry Wall 等人所著的畅销书《Programming Perl》（O'Reilly 出版）的封面图片正是一匹骆驼。”



格式：

单字之间不以空格断开（例：camel case）或连接号（-，例：camel-case）、下划线（_，例：camel_case）链接，有两种格式：

小驼峰式命名法（lower camel case）：第一个单字以小写字母开始；第二个单字的首字母大写，例如：firstName、lastName。

大驼峰式命名法（upper camel case）：每一个单字的首字母都采用大写字母，例如：FirstName、LastName、CamelCase，也被称为 Pascal 命名法（Pascal Case）。