编码风格

一、 概述

正确无误的代码是程序良好运行的基本保证。所有的代码必须保证无任何编译警告，无法避免的警告用编译预处理指示符#pragma warning(disable:****)关闭它。

清晰可读的代码能大大减小后期维护的代价，使后继开发人员能很快地投入到代码中来。

严谨单一的代码能减少代码风格上的随机性，使同一开发团队组内的任何人都能轻易地适应其他人开发的代码。

简洁方便的代码风格才能在编码过程中更好地贯彻以上要求，提高编码效率。

本规范为 Windows 平台和 Visual C++环境而定制，分别从项目组织、文件结构、注释风格和代码规范等四个方面来描述一种编码风格，力求使代码组织做到正确无误、清晰可读、严谨单一、简洁方便。

这里所描述的规范，除了命名风格（包括常量、变量和函数命名等）和代码结构之外，均可使用 SourceStyler C++（选择类似 MFC 的风格）来格式化，使之成为标准代码。所以保证命名风格和代码结构的规范性尤其重要。

二、 方案

这里指的方案，与 Visual C++ 6.0 的 Workspace 和 Visual Studio .NET 的 Solution 同义。

(一) 目录组织

- 除公用文件（如平台库、DXSDK 等）外，一个方案内的所有文件都放置于与方案名同名的目录或子目录下。
- 一个项目内的所有文件放置于与项目名同名的目录或子目录下。
- 保持方案目录清洁，方案目录（及其子目录）下不要存放无用的文件或目录。
- 方案内的项目共用文件只保留一个拷贝。如 Project1 项目用到了 Project2 项目的一个头文件，这时候不要把头文件拷贝一份，而是在 Project1 项目中使用相对路径访问 Project2 项目中的头文件。

(二) 项目组织

- 合理划分项目，做到代码可充用性。
- 项目命名做到简单且有意义，注意使用大小写。
- 方案内应该设置好项目依赖性。
- 项目内的文件和目录应该使用相对路径。
- 把共用头文件和库文件的路径设置在编译器的选项内，而不是项目设置中。
- 原则上，项目应支持 UNICODE 编译，并为多平台编译预留最大的方便性。

三、 文件

- 文件长度一般不超过 1000 行，最长不超过 2000 行。
- 头文件中只存放声明，而不存放定义。
- 每个 C++类对应一个 CPP 文件和一个头文件。
- 头文件使用#pragma once 或#ifdef/#define/#endif 防止重复包含。
- 使用 TAB 制表符，养成良好、严格的缩进风格。
- 每行最长 120 字符，超过部分要做折行处理。
- 按块划分代码，用 120 个“//”的行作为划分（这里称之为块分隔符），每块之间留三行空行。
- 每个函数一个块。
- 每个代码块不超过 200 行。
- 代码块按下列先后顺序排布：
  - 版权声明和文件描述块。
  - 头文件包含块。
  - 宏定义块。
  - 结构和类声明块。
  - 全局变量和常量块。
  - 函数块。
- 多数时候应该看到文件名就能够顾名思义，文件描述并无多大作用，反而是累赘，所以可以省略。

四、 注释

- 注释是对代码的“提示”，而不是文档。程序中的注释不可喧宾夺主，注释太多了会让人眼花缭乱。注释的花样要少。
- 尽量使用“//”注释符。
- 每个代码块必须要有注释，紧跟于块分隔符的下一行。
- 尽量把注释写在被注释代码的前一行，而不要写在同一行上。
- 如果代码本来就是清楚的，则不必加注释。否则多此一举，令人厌烦。
- 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。
- 尽量避免在注释中使用缩写，特别是不常用缩写。

五、 代码

(一) 表达式

- 除一元操作符外，操作符前后保留一个空格。如：i++和 j = i + 2。
- 表达式用括号包围。如：if ((i > 10) && (i < 20))，而不是：if (i > 10 && i < 20)。
- 括号内紧跟括号处不留空格，括号外留一个空格。
- 表达是折行时，操作符在放置在行尾，而不是下一行首。

(二) 语句

- 使用空行分割功能不同德语句块。
- 每个语句单独一行。
- 语句控制关键字后留一个空格。
- 不使用连续赋值语句。
- 所有的语句块都使用花括号包围，如：

```
if (CONDITION)
{
j++;
}
```

  而不是：

```
if (CONDITION) i++;
```

- if/else 语句按下列格式书写：

```
if (CONDITION1)
{
…
}
else if (CONDITION2)
{
…
}
else
{
…
}
```

- for 语句按下列格式书写：

```
for (i = 0; i < 1024; i++)
{
if (CONDITION)
{
    break;
    }
```

…
}

- while 语句按下列格式书写：

```
while (CONDITION)
{
…
};
```

- do/while 语句按下列格式书写：

```
do
{
…
}
while (CONDITION);
```

- switch/case 语句按下列格式书写：

```
switch (EXPRESSION)
{
case CONDITION1:
…
break;

case CONDITION2:
…
break;

default:
…
break;
}
```

其中，default 语句块可以省略，其它必须严格按照格式书写。

(三) 变量

- 除非要求 C++和 C 同时兼容，否则在需要用到时在声明局部变量。
- 一般情况下，一行只定义一个变量。
- 局部变量定义处一般不赋初值，赋初值将导致编译器无法检查变量是否被引用。如果要赋初值，应该把赋值语句放在变量定义后面的执行代码中。
- 定义全局变量和静态变量时一定要赋初值（即便是初值为零）。
- 变量名称不宜过长。
- 严格遵循变量命名规范。
- 变量命名：

  [<作用范围>_]<类型><主体>。

- 作用范围标识如下表：

作用范围	标识
局部变量与结构成员变量	无
全局变量	g_
静态变量	s_
类、命名空间成员变量	m_
常量	c_

- 类型列表如下：

类型	标识
void/VOID	v
char/CHAR	c
int/INT	i
short/SHORT	s
long/LONG	l
float/FLOAT	f
double/DOUBLE	d
unsigned char/UCHAR	uc
unsigned int/UINT	u
unsigned short/USHORT	us
unsigned long/ULONG	ul
ATOM	a
BOOL	b
BYTE	b
COLORREF	cr
DWORD	dw
HANDLE/HWND/HDC/等	h
LPARAM	l
TCHAR	tc
WCHAR	wc
WORD	w
WPARAM	w
PSTR	psz
PTSTR	ptz
PWSTR	pwz
INT32	i64
INT64	i32
INT_PTR	ip
DWORD32	dw32
DWORD64	dw64
DWORD_PTR	dwp

- 尽量不使用 C/C++中原有的类型（小写的 int 等等），而是使用 Window SDK 定义过的（大写的）。其中指针如果存在尽量使用 P 开头的，而不是 LP 开头的。
- 如果一个变量是布尔型的，应该使用 BOOL 变量，而不使用 bool 和 int/INT。

- 变量主体名字要取得有意义。主体词语开头以大写字母为分割，而不是下划线。如 dwColorSpace 而不是 dw_color_space。
- 普通变量主体不好命名时，使用 Temp 作为主体，如 dwTemp。
- 结构变量主体不好命名时，使用 Info/Data 作为主体，如 WSADATA wdData。
- 临时整数形变量允许使用单个字符变量名，如 i、j、k、l、m、n、x、y、z 等。
- 临时指针变量语序使用一下变量名：p、q、r。
- 类、结构、枚举和联合变量使用类型简写，允许只使用类型标识。举例如下：

```
RECT rt;
RECT rtWnd;
BITMAPINFOHEADER bih;
PAINTSTRUCT ps;
OPENFILENAME ofn;
OPENFILENAME ofnFile;
CSelection sel;
CSelection selStart;
```

- 指针变量以 p 作为类型开头。除指向结构/类/联合/接口的指针之外，不允许省略类型。如：

```
PINT piTotal;
PRECT pRt;
PRECT prtRect;
CImage *pImg;
CImage *pimgFirst;
```

- 在指针和引用变量定义和使用处，*和&符号均紧靠变量名称，而不是变量类型。

(四) 常量

- const 常量命名规范与变量类似，除了它必须以 c_开头之外。

(五) 预处理指示符

- 所有预处理语句（以#开头的语句）都不缩进。
- #define 常量以几个大写字母开始，加上下划线，后面将上常量主体，如：IDM_FileOpen 和 NUM_StatusBar。
- #define 宏定义使用下划线开始，其它部分与函数相同，如_SafeRelease。
- 建议维护一个的 Macro.h 文件，用来存放经常使用的工具宏。

(六) 标号

- 标号双下划线开始。
- 原则上不使用标号和 goto 语句，而只使用下列语句：

```
do
{
```

```
…
if (…)
{
    break;
    }
    …
}
while (FALSE);
```

(七) 结构

- 结构规范按下列格式书写（其中粗体部分为标准部分，不可更改； typedef 一行可以省略）：

```
typedef struct _STRUCTNAME
{
DWORD dwMember;
…
}
STRUCTNAME, *PSTRUCTNAME;
typedef const PSTRUCTNAME PCSTRUCTNAME;
```

(八) 联合

- 联合规范与结构类似。

(九) 枚举

- 枚举规范与#define 常量类似。

(十) 类

- 组成类名的所有单词的首字母都大写，自然连接，并使用 C 作为统一前缀。
- 尽量不要从结构中继承。除了只有五行以内的内联函数外，类声明中不要出现函数实现代码。
- 用存取访修饰符 private/protected/public 分割类的代码块，且允许存在多个同样的存取访修饰符。
- 类声明的书写规范举例如下：

```
calss CBitmap: public CImage, public CMemFile
{
private:
DWORD m_dwMember;
```

```
protected:
    LONG m_lSize;

public:
…

private:
DWORD GetSize();
…

private:
…

public:
inline DWORD GetPitch() {return m_dwPitch;}
    …

};
```

- 构造函数初始化每个成员一行，书写格式如下所示：

```
CQueue::CQueue():
m_dwSize(0),
m_lMember(1)
{
…
}
```

(十一) 命名空间

- 命名空间成员不缩进，参照如下格式：

```
namespace Inner
{
DWORD m_dwSize;

class Y
{
Y();
~Y();
};
}
```

(十二) 模版

- 尽量不使用模版。
- 除非一个项目全部使用 C/C++标准库，否则应尽量使用 Windows 平台库，而不是 CRT 函数和 C++标准模版库 STL。

(十三) 函数

- 组成函数名的所有单词的首字母都大写，自然连接，无前缀。
- 如果函数没有参数，则不要使用 VOID 或 void 填充，以保持代码整洁。
- 避免函数有太多的参数，参数个数尽量控制在 5 个以内。如果参数太多，在使用时容易将参数类型或顺序搞错。
- 尽量不要使用类型和数目不确定的参数。
- 在函数体的入口处，使用断言对参数的有效性进行检查。
- 不要省略返回值的类型，函数返回类型规则与变量类型的规则相同。
- 如果参数是指针，且仅作输入用，则应在类型前加 const，以防止该指针在函数体内被意外修改。如：VOID MemCopy(PVOID pvDst, const PVOID pvSrc);
- 如果某函数是导出函数，或者存在外部 ASM 版本，应该使用__stdcall 调用方式，即使用 WINAPI 标志符。否则，不使用调用修饰符（默认为__cdecl）。
- 尽量在函数的开始处集中定义局部变量（特殊情况除外，如 new 一个新对象）。

(十四) 内联汇编

- 尽量不使用固有特性函数（如__debugbreak），而不是汇编代码（如__asm int 3）。
- 如果要使用汇编代码，因该尽量使用独立汇编文件，而不是内联汇编。
- 如果使用了内联汇编，应该使用#ifdef _M_IX86 等条件编译来确定目标平台。
- 内连汇编代码尽量保持与 MASM 一致的风格（如使用 0FFH 表示十六进制）。
- 按以下规则编写宏：
  - 使用花括号把__asm 块包围住。
  - 把__asm 关键字放在每条汇编指令之前
  - 使用经典 C 风格的注释（“/* comment */”）,不要使用汇编风格的注释（“; comment”）或单行的 C/C++ 注释（“// comment”）。

(十五) MASM 汇编

- 尽量不使用汇编，而使用 Intrinsic 实现 MMX/SSE/SSE2 优化。
- 与 C/C++代码类似，按块划分汇编代码，用分号“;”加上 119 个“//”的行作为划分（块分隔符），每块之间留三行空行。
- 每个 PROC 和 MACRO 一个块。
- 每个块不超过 200 行。
- 块按下列先后顺序排布：

  - 指令集、模块属性等。
  - EQU 常量。
  - .DATA/.DATA?段变量。
  - .CODE 段，PROC 和 MACRO 块。
  - END 指示符
- CPU 指令和寄存器为小写。
- 指示符（如 PROC）和操作符（如 PTR）均为大写。
- 变量与标号命名、表达式风格与上文描述保持一致。
- 使用.IF/.ELSEIF/.ELSE/.ENDIF、.WHILE/.ENDW、/.REPEAT/.UNTIL/.UNTILCXZ 和.BREAK/.CONTINUE 指示符来控制代码流程，避免使用转跳指令。
- 使用 TAB 制表符缩进代码。使用 TAB 制表符对齐第一个操作符。
- 代码书写格式举例如下：

```
…
;////////////////////////////////////////////////////////////////////////////////////////////////////////////////
; IDCT and add/cpoy block.
IdctBlock_mmx PROC USES ebx, pbMb: PTR BYTE, iStride: SDWORD,
                                psBlock: PTR SWORD, bCopy: SDWORD
LOCAL qwTemp[32]: QWORD
LOCAL qwScratch1: QWORD
LOCAL qwScratch2: QWORD
…
; Add block.
mov             ebx, psBlock
mov             edx, pbMb
mov             eax, iStride

mov             ecx, bCopy
or              ecx, ecx
.IF ZERO?
    ; Add block.
    pxor        mm7, mm7
    FOR i, <0, 1, 2, 3>
        …
        IF (i LT 3)
            lea         edx, [edx + 2 * eax]
        ENDIF
    ENDM
.ELSE
    ; Copy block.
    …
.ENDIF

emms
ret
```

```
IdctBlock_mmx ENDP
;////////////////////////////////////////////////////////////////////////////////////////////////////////////////
…
```

- 当某个函数存在 C/C++版本和 MASM 版本（如 MMX、SSE2 优化版本）时，按以下风格书写：
  - 在.h 头文件中声明一种新的函数类型，并用这种类型声明一个函数指针变量；在.c/.cpp 文件中定义函数指针变量。
  - 假设 C/C++函数名成为 AddBlock，函数类型为函数名称的全部大写形式，即 ADDBLOCK。
  - MASM 函数使用 STDCALL 调用方式，C/C++函数使用 extern “C” __stcall（使用 WINAPI 类型替换__stdcall 关键字）调用方式。
  - 保证 C/C++函数名称符合前面所述规范，比如 AddBlock，而不是 AddBlock_c。
  - MASM 函数名称为 C/C++函数名称加上下划线，再加上汇编版本的特征，如 AddBlock_mmx/AddBlock_mme/AddBlock_xme 等。
  - 使用函数指针来访问，函数指针命名为 p 加上函数名称，如 pAddBlock。即使指针变量为全局变量，变量名称也不要以 g_开头。
  - 函数指针一般为全局变量，声明时初始化为肯定能使用的函数（比如 C/C++版本或 MMX 版本）。在程序运行后，判断运行环境，再确定最优的函数。
  - AddBlock.h 文件部分内容举例如下（粗体部分为固定不变格式）：

```
// AddBlock function and function pointer declaration.
typedef VOID (WINAPI ADDBLOCK)(PBYTE pbMb, PSHORT psBlock);
ADDBLOCK AddBlock;
ADDBLOCK AddBlock_mmx;
ADDBLOCK AddBlock_xme;

extern ADDBLOCK *pAddBlock;
```

  - AddBlock.c 文件部分内容举例如下（粗体部分为固定不变格式）：

```
// AddBlock function pointer.
ADDBLOCK *pAddBlock = AddBlock;

VOID WINAPI AddBlock (PBYTE pbMb, PSHORT psBlock)
{
…
}
```

  - AddBlock.mmx.asm 文件部分内容举例如下（粗体部分为固定不变格式）：

```
.686
.MMX
.MODEL FLAT, STDCALL
…
AddBlock_mmx PROC pbMb: PTR BYTE, psBlock: PTR SWORD
…
ret
AddBlock_mmx ENDP
```