**C#编码规范  
版本1.0**

**Content**

**1 规范目的…………………………………………2**

**2 适用范围…………………………………………2**

**3 编码规范…………………………………………2**

1. **概述 ………………………………………2**
   1. **前言 ……………………………………………2**
   2. **规范制定原则…………………………………**…**2**
2. **文件组织**

**2.1 C#源文件…………………………………………2**

**2.2 目录规范………………………………………3**

**2.3 缩进…………………………………………3**

**2.4 注释…………………………………………4**

**2.5 声明…………………………………………5**

**2.6 语句…………………………………………5**

**2.7 命名规范……………………………………5**

**2.8 命名指南……………………………………5**

1. **目的**

良好的编程风格是提高程序可靠性非常重要的手段，也是在大的软件项目中多人合作开发的基础。为了提高大家C#源程序的质量和可维护性，通过本规范定义来规避不好的编程风格，增强代码的可读性与可维护性，以便于自己和他人阅读。

1. 适用范围

本规范适用于软件开发中心所开发的后台软件、主站系统平台及应用软件的C#编码。

1. 编码规范

第一章 概述

* 1. 前言

为了保证软件程序的质量，提高程序的可维护性和可测试性，特制定本C#编程规范。

* 1. 规范制定原则
* 方便代码的交流和维护。
* 只规定需要规定的事情，不要强制施加个人喜好或者过时的做法。
* 不影响编码的效率，不与大众习惯冲突。
* 使代码更加美观、阅读更为方便。
* 正确、简单和清晰第一，坚持KISS原则。使代码的逻辑更清晰、更易于理解

第二章 文件组织

2.1 C#源文件

保持你的类和文件尽量简短，不要超过2000行，拆分你的代码使得结构更清晰。把每一个类放在单独的文件里，并以类名来命名文件（后缀.cs作为扩展名），这种约定可以简化我们的编程工作。

2.2 目录规范

为每一个命名空间创建一个目录。\*这样做可以很容易地映射命名空间到文件目录。

* 1. 缩进
     1. 换行

如果一个表达式不适合放在同一行的时候，可以根据以下原则对其断行处理。

* 在逗号后断行。
* 在操作符后断行。
* 在低优先级别处断行。
* 把新一行的表达式和前面的表达式在开始的地方要对齐排列

例1：longMethodCall(expr1,

expr2);

例2：var=a\*b/(c+f)+

4\*z;

* + 1. 空白符

一个缩进标准是用空格无法达到的。有人喜欢用两个空格，有人喜欢用四个空格，而其他人更喜欢用八个空格，设置更多的空格。最好使用制表符。制表符有以下有点：

* 每个人都可以设定他们喜欢的缩进层级
* 它只是一个字符，而不是二、四、八…因此可以减少敲键盘的次数（即使使用智能缩进，有时你也需要手动设定缩进，或者删除智能缩进或者其他的问题）
* 如果你想增加缩进（或者减少），标记光标位置，然后可以用tab增加缩进，用shift+tab减少缩进。这对绝大多数文本编辑器都适用。

这里，我们定义制表符作为标准缩进字符。

不要用空格做缩进，用制表符做缩进！

2.4 注释

2.4.1 块注释

应经常避免使用块注释，因为使用///注释作为C#标准注释是被推荐的。如果你要使用块注释，应该用下面的样式：

/\* Line 1

\* Line 2

\* Line 3

\*/

因为这种样式使（人）读者很容区分块注释和编码。你也可以使用旧的C语言的样式风格来做单行注释，尽管这不被推荐。一旦你使用了这种样式，注释后便就应该断行，因为如果放在一行，很难分清楚注释和后边的代码。

/\* blah blah blah \*/

2.4.2 单行注释

你应该使用//注释样式来注释掉代码（SharpDevelop有快捷键Alt+/）。它也可以用来注释掉代码片段。单行注释在代码文件中使用时，一定要缩进到缩进位上。应该在第一行注释掉代码，这样能够使注释掉的代码和为注释的代码很容易区分。 一条经验法则：通常，注释的长度不应该超过代码，如果解释太多，就说明这是一段复杂的、有潜在bug的代码。

2.5 声明

2.5.1 每一行声明的数量

要求一行只能有一个声明，这样做的好处可以利于注释，

避免一行声明多个变量或者不同类型的变量。

2.5.2 初始化规范

尽量在局部变量声明时就初始化。

2.5.3 类、借口和命名空间的声明

当编写C#类、接口和命名空间的时候， 开始的大括号“{”出现在声明语句的下一行结束的大括号“}”自己在一行，缩进到和它对应的大括号的位置上。

2.5.4 方法的声明

关于方法的声明，大括号和类的大括号放置规则一样，并且方法名字和它参数列表前边的圆括弧“(”不应该有空白符。

2.6 语句

2.6.1 简单语句

每行只有一条语句

2.6.2 返回语句

不要用圆括号把返回语句都包括起来。

2.6.3 FOR/FOREACH语句

For语句格式：

for(语句1;语句2;语句3)

{

循环体；

}

for(n ti in IntLIist)

{

语句；

}

2.6.4 while/do-while语句和switch语句

格式在原来格式基础上参照for和foreach语句。

* 1. 命名规范

2.7.1大写风格

* **Pascal**风格

这种风格是每个单词首字母大写（例：TestCounter）

* 骆驼式风格

这种方式是除了第一个单词之外每个单词的首字母大写

* 大写字母命名规则

全大写标识符仅用在由一俩个字母组成的缩写情况下，由三个字母及以上的采用Pascal风格或者骆驼式风格。

2.8 命名指南

一般，在命名中使用下划线和以匈牙利命名规则命名变量被认为是不好的习惯。 匈牙利命名规则定义了一套前缀和后缀来命名变量，反映出变量的类型。这种命名方式在以前的windows编程中被广泛使用，但是现在却不被使用，或者不建议使用。如果你遵循本规范，那么你不能使用匈牙利命名规则。记住，一个好的变量名字是描述的语义而不是类型。

2.8.1类命名指南

* 类的名字必须为名词或者名词短语。
* 使用Pascal命名规则，参见8.1.1
* 不要使用类前缀

2.8.2变量命名

计数型变量倾向于使用i, j, k, l, m, n这几个字母，一般都用于常见的的循环计数中。（参见10.2中关于全局计数器等的更加智能的命名的例子。） 更喜欢在布尔型变量前加前缀，如Is, Has 或者 Can。一般你应该给布尔型变量命名能意味真或假的名字（例如：fileFound, done, success或者带is前缀：isFileFound, isDone, isSuccess，但是不要使用IsName这样根本没有意义的名字）。使用骆驼式命名规则，参见8.1.2

2.8.3 方法命名

* 用动词或者动词短语命名方法名字
* 使用Pascal命名规则

2.8.4 属性命名

* 使用名词或者名词短语命名属性的名字
* 使用Pascal命名规则
* 考虑使用属性类型的名字来命名属性

例子：

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

using System.Data.OleDb;

using Excel=Microsoft.Office.Interop.Excel;

namespace WindowsFormsApp1

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

public double DmsToRad(string s)

{

string[] ss = s.Split(new char[3] { '°', '′', '″' }, StringSplitOptions.RemoveEmptyEntries);//将角度按°，′，″分开存入字符串数组中

double[] d = new double[ss.Length];//创建一个数组来保存度分秒值

for (int i = 0; i < d.Length; i++)

d[i] = Convert.ToDouble(ss[i]);

double sign = d[0] >= 0.0 ? 1.0 : -1.0;//判断角度正负

double rad = 0;

if (d.Length == 1)

rad = Math.Abs(d[0]) \* Math.PI / 180;

else if (d.Length == 2)

rad = (Math.Abs(d[0]) + d[1] / 60) \* Math.PI / 180;

else rad = (Math.Abs(d[0]) + d[1] / 60 + d[2] / 3600) \* Math.PI / 180;

rad \*= sign;

return rad;

}

public string RadToDms(double rad)

{

double sign = rad >= 0.0 ? 1.0 : -1.0;//判断弧度正负

rad = Math.Abs(rad) \* 180 / Math.PI;//将弧度转成度

double[] d = new double[3];

d[0] = (int)rad;

d[1] = (int)((rad - d[0]) \* 60);

d[2] = (rad - d[0] - d[1] / 60) \* 3600;

d[2] = Math.Round(d[2], 2);

if (d[2] >= 60)

{

d[1] += 1; d[2] -= 60;

if (d[1] == 60)

{

d[0] += 1; d[1] -= 60;

}

}

d[0] \*= sign;

string s = Convert.ToString(d[0]) + "°" + Convert.ToString(d[1]) + "′" + Convert.ToString(d[2]) + "″";//将d数组按度分秒格式转成字符串输出

return s;//返回字符串的值

}

}