**Qt Coding Style**

This is a overview of the coding convention we use when writing Qt code.  
The data has been gathered by mining the Qt sources, discussion forums,  
email threads and through collaboration of the developers.

**Indentation**

* 4 spaces are used for indentation
* Spaces, not tabs!

**Declaring variables**

* Declare each variable on a separate line
* Avoid short (e.g. “a”, “rbarr”, “nughdeget”) names whenever possible
* Single character variable names are only okay for counters and temporaries, where the purpose of the variable is obvious
* Wait with declaring a variable until it is needed
* // Wrong
* int a, b;
* char \*c, \*d;
* // Correct
* int height;
* int width;
* char \*nameOfThis;
* char \*nameOfThat;
* Variables and functions start with a small letter. Each consecive word in a variable’s  
  name starts with a capital letter
* Avoid abbreviations
* // Wrong
* short Cntr;
* char ITEM\_DELIM = '\t';
* // Correct
* short counter;
* char itemDelimiter = '\t';
* Classes always start with a big letter. Public classes start with a ‘Q’ (QRgb). Public functions most often start with a ‘q’ (qRgb).

**Whitespace**

* Use blank lines to group statements together where suited
* Always use only one blank line
* Always use a single space after a keyword, and before a curly brace.
* // Wrong
* if(foo){
* }
* // Correct
* if (foo) {
* }
* For pointers or references, always use a single space between the type and ‘\*’ or ‘&’, but no space between the ‘\*’ or ‘&’ and the variable name.
* char \*x;
* const QString &myString;
* const char \* const y = "hello";
* No space after a cast.
* Avoid C-style casts when possible.
* // Wrong
* char\* blockOfMemory = (char\* ) malloc(data.size());
* // Correct
* char \*blockOfMemory = (char \*)malloc(data.size());
* char \*blockOfMemory = reinterpret\_cast<char \*>(malloc(data.size()));

**Braces**

* As a base rule, the left curly brace goes on the same line as the start of the statement:
* // Wrong
* if (codec)
* {
* }
* // Correct
* if (codec) {
* }
* Exception: Function implementations and class declarations always have the left brace on the start of a line:
* static void foo(int g)
* {
* qDebug("foo: %i", g);
* }
* class Moo
* {
* };
* Use curly braces when the body of a conditional statement contains more than one line,  
  and also if a single line statement is somewhat complex.
* // Wrong
* if (address.isEmpty()) {
* return false;
* }
* for (int i = 0; i < 10; ++i) {
* qDebug("%i", i);
* }
* // Correct
* if (address.isEmpty())
* return false;
* for (int i = 0; i < 10; ++i)
* qDebug("%i", i);
* Exception 1: Use braces also if the parent statement covers several lines / wraps
* // Correct
* if (address.isEmpty() || !isValid()
* || !codec) {
* return false;
* }
* Exception 2: Use braces also in if-then-else blocks where either the if-code or the else-code covers several lines
* // Wrong
* if (address.isEmpty())
* return false;
* else {
* qDebug("%s", qPrintable(address));
* ++it;
* }
* // Correct
* if (address.isEmpty()) {
* return false;
* } else {
* qDebug("%s", qPrintable(address));
* ++it;
* }
* // Wrong
* if (a)
* if (b)
* ...
* else
* ...
* // Correct
* if (a) {
* if (b)
* ...
* else
* ...
* }
* Use curly braces when the body of a conditional statement is empty
* // Wrong
* while (a);
* // Correct
* while (a) {}

**Parentheses**

* Use parentheses to group expressions:
* // Wrong
* if (a && b || c)
* // Correct
* if ((a && b) || c)
* // Wrong
* a + b & c
* // Correct
* (a + b) & c

**Switch statements**

* The case labels are on the same column as the switch
* Every case must have a break (or return) statement at the end or a comment to indicate that there’s intentionally no break
* switch (myEnum) {
* case Value1:
* doSomething();
* break;
* case Value2:
* doSomethingElse();
* // fall through
* default:
* defaultHandling();
* break;
* }

**Line breaks**

* Keep lines shorter than 100 characters; insert breaks if necessary.
* Commas go at the end of a broken line; operators start at the beginning of the new line. The operator is at the end of the line to avoid having to scroll if your editor is too narrow.
* // Correct
* if (longExpression
* + otherLongExpression
* + otherOtherLongExpression) {
* }
* // Wrong
* if (longExpression +
* otherLongExpression +
* otherOtherLongExpression) {
* }

**1：Add file and version information.**

At the beginning of header files and .cpp files, need to use comment make simple notes about Copyright, Function description ,Version, Author,ect.

／\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

＊　Copyright(c)2010 Teleca company.

＊　All rights reserved.

＊

＊　File name：filename.h/ filename.cpp

＊　Function description：Brief description files content and features.

＊

＊　Version No.：1.1

＊　Author：

＊　Date：

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**2 Fucntion comments: Features, Parameters and Return value**

Every program shoud begin with a brief comment to explain the features of function. And all of parameters and return values should add commentsto brief description too. E.g:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function name

\* Features description：Briefly describe the features of function

\* Parameters list：param1——Description；

\* param2——Description；

\* param3——Description；

\* Return value：Briefly describe the return value

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**3. Variables definiton and comment.**

The definition of variables use camel style(The compound words or phrases in which the elements are joined without spaces, the first letter is use low case and the words behind are start with upper case). Should use “Nonu” or “Adj. + Nonu”. E.g.

float value;

float oldValue;

float newValue;

Every static variable and global variable need comments. And important local variable alse need commmens.

**4.** The principle of variable and function names

1)Global and

全局（包括类中的）变量用长名字，局部变量用短名字。类成员变量前应加上 ｍ＿，静态变量加上S\_,全局变量加上 ｇ＿.

* 类的数据成员加前缀m\_（表示member），这样可以避免数据成员与成员函数的参数同名。

例如：

void Object::SetValue(int width, int height)

{

m\_width = width;

m\_height = height;

}

* 静态变量加前缀s\_（表示static）。

例如：

void Init(…)

{

static int s\_initValue; // 静态变量

…

}

* 如果不得已需要全局变量，则使全局变量加前缀g\_（表示global）。

例如：

int g\_howManyPeople; // 全局变量

int g\_howMuchMoney; // 全局变量

2)局部变量应尽量易懂简洁，使用常见的变量，如: nCount，i，j，k，len, pos, ect.

3)自定义信号以sig＿开头，自定义槽以slt＿开头。sig＿和slt＿后面第一个单词（或其缩写）要以小写字母开头，第二个以后的单词（或缩写）要以大写字母开头。

5)如果程序中出现了两个类名缩写一样的问题，如：QToolBar 和 QToolButton 的缩写都是tb，这时应将其中一个类的缩写进行变动，变动的准则以能够避免冲突，同时缩写能够表达类名为准。如这里可以将QToolBar 的缩写为tbar，QToolButton仍用tb，这样就可以避免命名冲突。

6) 类名用大写字母开头的单词组合而成，而函数名用骆驼风格。

例如：

class Node; // 类名

class LeafNode; // 类名

void drawRect(void); // 函数名

void setValue(int value); // 函数名

**5. 头文件结构及类声明体中的编排及**

1) 头文件由三部分内容组成：

<1>头文件开头处的版权和版本声明（参见示例1-1）。

<2>预处理块。

<3>函数和类结构声明等。

其中，头文件以内部包含卫哨开始（#define\*\*\*, #ifndef\*\*\*），以内部包含卫哨结束(#endif //\*\*\*)。

2)在类的声明体中，请按照：Q\_OBJECT、public：、siganls：、slots：、protected：、priavte:的顺序将成员变量和函数进行排列。如果需要在这个类（class）中声明某种数据类型（结构、枚举等），请将这个声明放在所有成员变量和成员函数的前面。如果在一种类型的声明中，即有成员变量，又有函数声明，请使用两个相同的类型声明将它们隔开，如：

private：

void function（）；

…

private：

int m\_num；

…

3)对一般的界面类，如果使用了信号／槽，则class声明的第一行必须是Q\_OBJECT。

4)建议避免使用protected 类型的成员函数或者成员变量，因为在 Qt 中，大量的事件处理函数（槽）均是以protected 类型出现的。

**6:界面布局原则**

　 制作Qt界面的时候，请尽量使用布局管理器(QlayOut)进行布局管理，避免使用绝对坐标，除非你能够绝对肯定这个界面的尺寸是完全不会变换的。如果某个区域出现了很多控件，在保证不会影响程序控制结构的前提下，请将这些控件放在一个可以包含其它窗口控件

的盒子（如 QWidget、QFrame、QGroupBox）中，然后再让这个盒子出现在界面上（即这些控件以整体形式出现）。书写界面上的英文长度的时候，请与对应的中文进行比较，确保汉化后中文能够被完全显示。

　 代码中如果出现了需要在界面上出现的文字，必须用英文表示，并且使用tr 函数将其包含。如：tr(“message！”),不能在程序中将这些文字直接用中文替换。

**7.调试信息原则**

代码中如果需要输出调试信息，请使用条件编译QT\_NO\_DEBUG\_OUTPUT宏将其包围，如：

#ifndef QT\_NO\_DEBUG\_OUTPUT

qDebug(“debug message”);

QMessageBox::warning(0, “warning”, “message”);

#endif // QT\_NO\_DEBUG\_OUTPUT