**1.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.

**2.Indentation**

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

**3.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).

**4.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()));

**5.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) {}

**6.Parentheses**

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

**7.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;
* }

**8.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) {
* }

**9：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：

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

**10. 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.**

1)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.

2)The usual controls variable, use controls abbreviation or “Verb + controls abbreviation”, “Adj + controls abbreviation”.

The usual controls abbreviation are in Appendix\_2.

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

1)Variable name use Hungarian notation, the detail of Hungarian notation please check Appendix\_1.

2)Global variable and class data members variable use long name, local variable use short name. The name of class data members beginning with m\_, the name of static variable beginning with s\_, the name of global variable beginning with g\_.

* The prefix of class data members start with m\_ can avoid confuse with another variable. E.g.

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

{

m\_width = width;

m\_height = height;

}

* The prefix of static variable is s\_(‘s’means static), E.g.

void Init(…)

{

static int s\_initValue; // static variable

…

}

* If we have to use global variable, the prefix of it is g\_ (means global), E.g.

int g\_howManyPeople; // global variable

int g\_howMuchMoney; // global variable

3)Local variable should simple and easy to understand, use common variable, E.g. nCount，i，j，k，len, pos, ect

4)In program, if two or more classes have same abbreviation, like QToolBar and QToolButton, both abbreviation are tb, then we need to change the abbreviation of one of them, the change principle is avoid conflict and can express the mean of class. E.g. The abbreviation of QToolBar is ‘tbar’, and QToolButton still use ‘tb’ for arrreviation.

5) Class name start with the combination of words that all start with upper case, but the function name use camel style. E.g.

class Node; // class name

class LeafNode; // class name

void drawRect(void); // function name

void setValue(int value); // function name

**12. Header files structure and class declaration arrangement.**

1) The header files consist with three parts.

<1>The version declaration at the beginning of header files.（Reference sample 1-1）。

<2> Preprocessing block。

<3>Function and class declaration, ect.

In it, the header files start with(#define\*\*\*, #ifndef\*\*\*)，end with(#endif //\*\*\*)。

2)In class declaration, the order is: Q\_OBJECT、public：、siganls：、slots：、protected：、priavte. If need to declaration some another data type(Structure,Enumerate,etc.), should put those declaration before the data members and function members. If in same class, have both data variable and function declaration, use same type declaration split both of them. E.g.

private：

void function（）；

…

private：

int m\_num；

…

3)At usual, if we use the Signals/Slots, the first sentence of class declaration is Q\_OBJECT.

4)Suggest avoid to use protected type function members or data members, because in Qt libraries, most of envent handle(slots) are use protected type.

**13:UI layout principle.**

When make Qt UI, use QLayout as you can for layout management, try to avoid using absolute coordinates, unless you are certain the the UI won’t changed forever. If one area have many controls, try to put those controls in a window box(e.g. QWidget, QFrame,QGroupBox,etc.), then put those window box in the UI.

**14.Debug information**

At first, in order to debug, we should add debug information in our codes. We should use the macro QT\_NO\_DEBUG\_OUTPUT control it, E.g.

#ifndef QT\_NO\_DEBUG\_OUTPUT

qDebug(“debug message”);

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

#endif // QT\_NO\_DEBUG\_OUTPUT

* 1. Appendix\_1: Hungarian notation 
  2. Appendix\_2: Contorls abbreviation 