**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(Can use Carbide and Source insighter present this)**

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

**Declaring variables（Not controllers variables defination）**

* 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) {
* }

**General exception**

* Feel free to break a rule if it makes your code look bad.