Qt Coding Convention

# General

This document is to define coding conventions when using Qt in IVC.

# Coding style

## Encoding

* All code is ACSII only (7-bit characters only)

## Indentation

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

## Declaring variables

* Declare each variable on a separate line
* Avoid short or meaningless names (e.g. "a", "rbarr", "nughdeget")
* Single character variable names are only okay for counters and temporaries, where the purpose of the variable is obvious
* Wait when 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 lower-case letter. Each consecutive word in a variable's name starts with an upper-case letter
* Avoid abbreviations

// Wrong

short Cntr;

char ITEM\_DELIM = ' ';

// Correct

short counter;

char itemDelimiter = ' ';

* Classes always start with an upper-case letter. Public classes start with a 'Q' (QRgb) followed by an upper case letter. Public functions most often start with a 'q' (qRgb).
* Acronyms are camel-cased (e.g. QXmlStreamReader, not QXMLStreamReader).

## 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";

* Surround binary operators with spaces
* No space after a cast
* Avoid C-style casts when possible

// Wrong

char\* blockOfMemory = (char\* ) malloc(data.size());

// Correct

char \*blockOfMemory = reinterpret\_cast<char \*>(malloc(data.size()));

* Do not put multiple statements on one line
* By extension, use a new line for the body of a control flow statement:

// Wrong

if (foo) bar();

// Correct

if (foo)

bar();

## Braces

* Use attached braces: The opening brace goes on the same line as the start of the statement. If the closing brace is followed by another keyword, it goes into the same line as well:

// Wrong

if (codec)

{

}

else

{

}

// Correct

if (codec) {

} else {

}

* 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 only when the body of a conditional statement contains more than one line:

// 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: Brace symmetry: Use braces also in if-then-else blocks where either the if-code or the else-code covers several lines:

// Wrong

if (address.isEmpty())

qDebug("empty!");

else {

qDebug("%s", qPrintable(address));

it;

}

// Correct

if (address.isEmpty()) {

qDebug("empty!");

} else {

qDebug("%s", qPrintable(address));

it;

}

// Wrong

if (a)

…

else

if (b)

…

// Correct

if (a) {

…

} else {

if (b)

…

}

* 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 in 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, unless another case follows immediately.

switch (myEnum) {

case Value1:

doSomething();

break;

case Value2:

case Value3:

doSomethingElse();

// fall through

default:

defaultHandling();

break;

}

## Jump statements (break, continue, return, and goto)

* Do not put 'else' after jump statements:

// Wrong

if (thisOrThat)

return;

else

somethingElse();

// Correct

if (thisOrThat)

return;

somethingElse();

* Exception: If the code is inherently symmetrical, use of 'else' is allowed to visualize that symmetry

## Line breaks

* Keep lines shorter than 100 characters; wrap if necessary
  + Comment/apidoc lines should be kept below 80 columns of actual text. Adjust to the surroundings, and try to flow the text in a way that avoids "jagged" paragraphs.
* Commas go at the end of wrapped lines; operators start at the beginning of the new lines. An operator at the end of the line is easy to miss if the editor is too narrow.

// Wrong

if (longExpression +

otherLongExpression +

otherOtherLongExpression) {

}

// Correct

if (longExpression

+ otherLongExpression

+ otherOtherLongExpression) {

}

## General exception

* When strictly following a rule makes your code look bad, feel free to break it

## Including headers

* In public header files, always use this form to include Qt headers: #include <QtCore/qwhatever.h>. The library prefix is neccessary for Mac OS X frameworks and is very convenient for non-qmake projects.
* In source files, include specialized headers first, then generic headers. Separate the categories with empty lines.

#include <qstring.h> // Qt class

#include <new> // STL stuff

#include <limits.h> // system stuff

* If you need to include qplatformdefs.h, always include it as the **first** header file.
* If you need to include private headers, be careful. Use the following syntax, irrespective of which module or directory whatever\_p.h is in.

#include <private/whatever\_p.h>

# Reference:

1. Coding Conventions: <http://wiki.qt.io/Coding_Conventions>
2. QML Coding Conventions: <http://doc.qt.io/qt-5/qml-codingconventions.html>
3. Qt Coding Style: <http://wiki.qt.io/Qt_Coding_Style>