CODING Standard

Für C++:

Formatierung

- Tabulator sollte nicht verwendet werden.
- Stattdessen Tabulator durch 4 Leerzeichen ersetzten.

Allgemeine Coding-Konventionen

- Der Codingstandard übernimmt zum Großteil die Standards des Qt-Frameworks (Info: qt-project)
- Camelcase-Notation
- Auf allgemeine Lesbarkeit/Verständlichkeit des Codes achten!
- C-Casts vermeiden, C++-Casts bevorzugen (static_cast, dynamic_cast, const_cast, reinterpret_cast) - Grund: Die Casts "reinterpret_cast" und der C-style Cast sind beide unsicher, aber "reinterpret_cast" entfernt wenigstens nicht den const-Modifizierer
- Konstruktoren verwenden um einfache Datentypen zu casten: int(myFloat) anstatt (int)myFloat - Grund: Der Compiler warnt, wenn der Cast unsicher werden würde (z.B. bei Änderung des Variablen-Datentyps)
- Templates vernünftig einsetzen, nicht nur, weil es möglich ist
- Zeiger nur mit gutem Grund einsetzen, eine Stack-basierte Umsetzung ist meistens vorzuziehen - Grund: keine uninitialisierten Zeiger, keine vergessenen deletes
- C-Zeiger sind grundsätzlich zu vermeiden. Sie können fast immer durch ein C++-Äquivalent ersetzt werden (std::unique_ptr, std::shared_ptr - Qt-Style: QScopedPointer, QSharedPointer)
- Zugriffsmodifizierer müssen vor jeder Member, Funktion oder Typ (Klasse) stehen
- Jede Datei beginnt mit dem nachfolgendem Header:

```
///------------------------------------------------------------------
/// Namespace:          -
/// Class:              Bsp.: Test
/// Description:        Short description of the class or file
/// Author:             Name of Author (z.B.: Max Mustermann)
/// Date:               Bsp.: Oct 2018
/// Notes:              -
/// Revision History:   Bsp.: First release
///------------------------------------------------------------------
```

Define / Makros

- Bei Define werden nur Großbuchstaben verwendet Bsp.:

```
#define TEST 10
```

- Defines wenn möglich vermeiden, static const (C++11 constexpr) stattdessen verwenden - Grund: Typsicherheit

Variablen

- Alle Variablen beginnen mit einem Kleinbuchstaben
- Jede Variable in einer separaten Zeile deklarieren
- Kurze oder bedeutungslose Variablennamen vermeiden (z.B. “a”, “rbarr”, “nughdeget”)
- Abkürzungen vermeiden

- Variablennamen bestehend aus nur einem Zeichen ausschließlich für Indizes oder temporäre Variablen verwenden, wenn die Bedeutung offensichtlich ist
- Variablen erst dann deklarieren, wenn sie verwendet werden
- typedef verwenden um Übersichtlichkeit zu gewährleisten (Vorsicht: keine Spaghetti typedefs!)

```
typedef std::vector<Class> ClassVector;
ClassVector halloWelt;
```

Funktionen

- Funktionsnamen beginnen immer mit Kleinbuchstaben
- Abkürzungen vermeiden
- Funktionen in der der Übersetzungseinheit (.cpp) implementieren, nicht in der Header-Datei! (*Ausnahme: Globale inline deklarierte Funktionen)

```
private void test() { };
```

- Die Übersetzungseinheit (.cpp) Datei hat den gleichen Namen wie die dazugehörige Header (.h) Datei
- Oberhalb jeder Funktionsdefinition befindet sich eine kurze Beschreibung

Klassen/Typen

- Selbst definierte Datentypen (z.B. Klasse) beginnen immer mit Großbuchstaben
- Klassennamen enthalten keine Unterstriche
- Datei hat den gleichen Namen wie die (Haupt-)Klasse
- Nur eine große Klasse pro File

```
public class Test { };
```

Klassenattribute

- Alle Member-Variablen einer Klasse beginnen mit dem Präfix m_
- Für statische Member-Variablen gilt das Präfix s_
- static const Attribute wie Defines in Großbuchstaben schreiben

```
private int m_TestTest;             //Lesbarkeit nach _ mit Großbuchstaben weiter!!!!
private int m_testTest;                     //Qt-Style bevorzugt nutzen
private static unsigned int s_TestTest;
private static unsigned int s_testTest;     //Qt-Style bevorzugt nutzen
private static const unsigned int TEST;
private List<Series> m_listSeries;
```

Get/Set Memberfunktionen

- Getter-Methoden die Member-Variablen zurückgeben beginnen mit „get“ Bsp.:

```
public int getWidth() const;      //bevorzugt nutzen
```

- (Qt-Style oder: Getter-Methoden bestehen nur aus dem Variablennamen Bsp.:

```
public int width() const;)
```

- Bei Flags (bool Datentyp) ist als Präfix „is“ zu verwenden Bsp.:

```
public bool isEnabled();
```

- Setter-Methoden die Member-Variablen setzen beginnen immer mit „set“ Bsp.:

```
public void setWidth(int width);
```

Leerzeichen

- Leerzeilen verwenden um zusammenhängende Ausdrücke zu gruppieren
- Nie mehr als eine Leerzeile verwenden

- Immer ein einzelnes Leerzeichen nach einem Keyword und vor einer geschwungenen Klammer verwenden
- Binäre Operatoren mit jeweils einem Leerzeichen davor und danach umschließen (Lesbarkeit!)
- Keine Leerzeichen nach einem Cast

Klammern

- Die linke geschwungene Klammer befindet sich **immer** am Zeilenanfang Bsp.:

```
public class Test
{
    //.....
};

private void test()
{
    //.....
    if (isTrue)
    {
        //.....
    }
    //.....
}
```

- Klammern verwenden um Ausdrücke zu gruppieren (z.B.: if ((a && b) || c)), auch wenn es durch die Operator-Prioritäten nicht unbedingt notwendig wäre (Lesbarkeit!)

Doxygen

- Dreifach-Slash Notation bevorzugen

```
int m_test;  ///< Testvariable
///\brief Kurze Beschreibung
```

- ASCII-Art vermeiden (z.B. Zeilenfüllende Sternchen **********)

Für qml:

- http://doc.qt.io/qt-5/qml-codingconventions.html
- https://material.io/design/