www.newtec.de

Creating safety.
With passion.

NewTec

QualityQuest
Coding Styleguide

1.1 Language

- Both source code and comments *shall* be in English.

1.2 Naming Conventions

- Names for modules, constants, variables, defines, functions, classes, methods and attributes *shall* be meaningful and easy to remember.
  - Natural language *should* be a model for naming.
- Loop varibles *shall not* be named „i“, „j“, „ii“ or similarly.

- Classes:
  - Noun, first letter upper case, rest lower caseRest
  - Whole words, concatenation by upper case letter („CamelCase“).

```
class Account;
class StandardTemplate;
```

- Methods:
  - Verb, imperative (demand), first letter lower case.
  - Reading/writing of attributes start with get/set prefix.
  - Requests/queries: bool result, no changes of values

```
bool checkAvailability();
void doMaintenance();
Date getDate();
```

```
bool isLarge() const;
bool hasFather() const;
```

- Attributes (C/C++ only):
  - Leading „m_“ prefix.

```
bool m_isAvailable;
Date m_date;
```

- Constants:
  - Upper case only.
  - Concatenation by „_“.
  - Standard prefixes are: `MIN_, MAX_, DEFAULT_`

```
#define DEFAULT_DURATION (200)
static const uint8_t MIN_WIDTH = 120;
static const uint8_t MAX_WIDTH = 360;
```

1.3 Indentation

- Functional blocks *shall* be indented.
- Each indentation depth *shall* be 4 spaces.
- Tabs *shall not* be used.
- Curly braces *shall* be the only content of their respective line.

```
if (isDegreeAcquired())
{
    throwParty();
    applyForJob();
}
```

1.4 Source Code Structure

- C/C++: Each class (C++) or module (C) *shall* have a .cpp file and a related .h file.
  - Exception: interfaces and templates
- C++/C#: Within a class, members *shall* be defined in the following order:
  - public
  - protected
  - internal
  - private

```
class Controller
{
public:
    ...

protected:
    ...

private:
    ...
};
```

1.5 Control Structures

- `if / else if / else`
  - Functional blocks *shall* be embraced by curly braces.
    - This also applies for blocks with a single statement.
  - After `if` or `else`, a space *shall* follow.

```
if (condition)
{
    singleCommand();
}
else if (condition)
{
    ...
}
else
{
    ...
}
```

- After `for` / `while` / `do while`, a space *shall* follow.

```
for (index = 0; index < 10; ++index)
{
    ...
}
```

```
while (conditon)
{
    ...
}
```

```
do
{
    ...

} while (condition);
```

- `switch / case` branching
  - After `switch`, a space *shall* follow.
  - `case/default` statements *shall* be at the same indentation depth as the `switch` statement.
  - Empty `case` statements *shall* be filled with a /* Fall through */ comment.

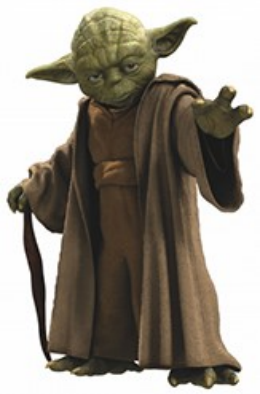
```
switch (condition)
{
case option1:
    ...
    break;

case option2:
    /* Fall through */

case option3:
    ...
    break;

default:
    ...
    break;
}
```

1.6 Equality Checks

- Literals *shall* be the first element of equality checks.

```
/** ... */
if (0 == counter)
{
    …
}
```

```
/** ... */
if (counter == 0)
{
    …
}
```



„Yoda condition“

Creating safety.
With passion.

NewTec

1.7 Comments

- The source code *shall* be documented by means of Doxygen and in Javadoc style.
- The following styles *may* be used for Doxygen relevant comments:
  - Forward comments, ie. the comment is related to the following element.
  - Backward comments, ie. the comment is related to the preceding element.

```
/** ... */
bool m_myMember1;

/**
 * ...
 * ...
 */
void myMethod();
```

```
bool m_myMember1; /**< ... */
```

- Doxygen comments of methods/functions *shall* include a description of the arguments.
- Doxygen comments of methods/functions *shall* include a description of the return value (if not `void`).

```
/**
 * This ...
 *
 * @param[in] myArgument The ...
 *
 * @return If it´s successful, it will return true, otherwise false.
 */
bool myMethod(uint8_t myArgument);
```