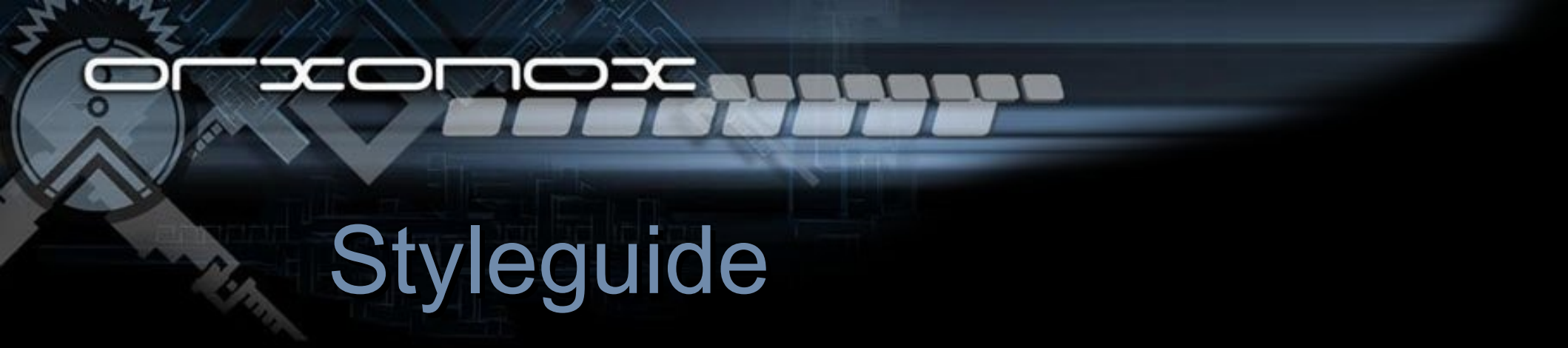


Styleguide

Styleguide

Was ist ein Styleguide?

- Ein Styleguide ist ein Set von Regeln, die definieren, wie unser Programmcode aussehen soll. Das beinhaltet die Formatierung, die Namensgebung, das Verwenden von speziellen Markierungen und das Einfügen von Kommentaren und Dokumentationen.

Styleguide

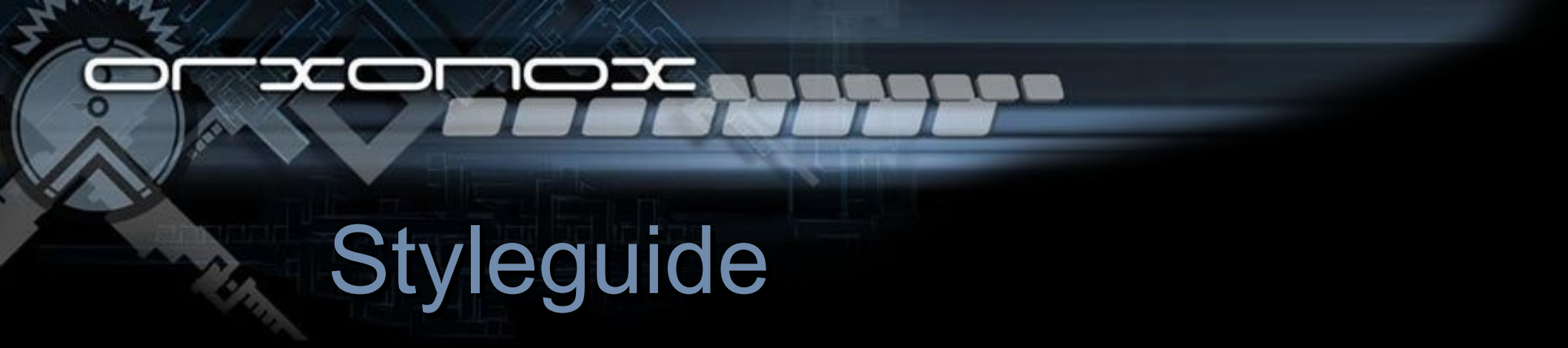
Warum ein Styleguide?

- Viele Entwickler arbeiten gemeinsam an Orxonox. Der Code ist umfangreich, keiner kennt jeden Teil. Oft muss man deshalb fremden Code betrachten. Dies fällt viel einfacher, wenn sich alle an die gleichen Regeln halten.

Styleguide

Was geschieht, wenn ich mich nicht daran halte?

- Im Prinzip nicht viel, aber Programmierer neigen dazu, das Rad neu zu erfinden. Wenn man deinen Code nicht versteht, wird er deswegen nutzlos und früher oder später ersetzt. Das wäre schade.



Styleguide

Styleguide

Whitespaces

- Nach jeder öffnenden geschweiften Klammer { wird der Code eine Stufe eingerückt.
- Die Einrückung beträgt 4 Leerzeichen.
- Wir verwenden keine Tabs, weil die Länge eines Tabs in jedem Editor anders ist und deswegen keine Formatierung möglich wäre.
- Jeder schlaue Editor kann beim drücken der Tabulator-Taste automatisch 4 Leerzeichen einfügen. Bitte konfiguriert euren Editor entsprechend.

Styleguide

Whitespaces

- Beispiel:

```
void myfunction()
{
    int limit = 5;
    for (int i = 0; i <= limit; i++)
    {
        COUT(0) << „Hello World!“ << std::endl;
    }
}
```

Styleguide

Klammern

- Geschweifte Klammern { und } stehen jeweils in einer separaten Zeile. So sieht man beim durchscrollen gleich wo ein Abschnitt beginnt und wo er wieder aufhört.

```
void myfunction()
{
    ....
}

if (i > limit)
{
    ...
    return;
}
```

Styleguide

Klammern

- Wenn innerhalb der geschweiften Klammern { und } nur eine Zeile steht, können die Klammern weggelassen werden. Dies betrifft allerdings nur if-else, for, while und ähnliche Aufrufe und NICHT Funktionen oder Klassen.

```
if (i <= limit)
    COUT(0) << „Hello World!“ << std::endl;
else
    return;
```

Styleguide

Klammern

- **Vorsicht**: Ungeklammerte Ausdrücke dürfen wirklich nur eine Zeile enthalten, alles andere ist ersten unschön und zweitens gefährlich:

```
if (i != 0)
    if (i > 0)
        COUT(0) << „i ist positiv“ << std::endl;
else
    COUT(0) << „i ist null“ << std::endl;
```

- Was geschieht hier?

Styleguide

Formatierung:

- Nach jedem Keyword folgt ein Leerzeichen. Operatoren werden mit Leerzeichen umschlossen:

```
int i = 1 + 2 + 3;

if (a + b < c)
```

- Ausnahmen sind erlaubt, wenn es die Gruppierung von Variabeln oder Zahlen verdeutlicht:

```
int c = sqrt(a*a + b*b);
```

Styleguide

Namensgebung:

- Klassen: CamelCase, Grossbuchstabe am Anfang:

```
class MyClass
```

- Funktionen: camelCase, Kleinbuchstabe am Anfang:

```
void myFunction()
```

- Files: CamelCase, gleich benannt wie die Klasse:

```
MyClass.cc und MyClass.h
```

Styleguide

Namensgebung:

- Variabeln: Normale Variabeln camelCase mit Kleinbuchstabe am Anfang, in Fällen, in denen es der Leserlichkeit dient, ist auch ein Underscore _ zur Formatierung möglich. Konstante Variabeln oder Schlüsselwerte werden komplett gross geschrieben:

```
int myVariable = 0;

int dies_ist_eine_variable = 0;

const int GLOBAL_MAXIMUM = 1000;
```

Styleguide

Namensgebung von Klassenvariabeln:

- Bei Klassenvariabeln wird ein Underscore _ angehängt. Somit ist auf einen Blick ersichtlich, welche Variabeln zur Klasse gehören und welche nur lokal existieren:

```
class MyClass
{
    void myFunction(int value);
    int member_;
}

void MyClass::myFunction(int value)
{
    this->member_ = value;
}
```

Styleguide

Namensgebung von Klassenvariabeln:

- Bei statischen Klassenvariabeln wird _s angehängt:

```
class MyClass
{
    void myFunction(int value);
    int member_;
    static int offset_s;
}

void MyClass::myFunction(int value)
{
    this->member_ = value + MyClass::offset_s;
}
```

Styleguide

Zugriff auf Klassenvariabeln:

- Auf Klassenvariabeln wird immer mit dem this-Pointer zugegriffen:

```
this->member_ = value;
```

- Auf **statische** Klassenvariabeln wird immer mit dem Klassenname zugegriffen:

```
MyClass::offset_s = value;
```

Styleguide

Headerfiles:

- In Headerfiles wird ein Includeguard verwendet, damit sich der Compiler nicht über mehrfache Definition beklagt wenn ein Headerfile an mehreren Stellen included wird:

```
#ifndef _MyClass_H__
#define _MyClass_H__

class MyClass
{
    ....
};

#endif /* _MyClass_H__ */
```

Styleguide

Headerfiles:

- In Headerfiles sollte nur das nötigste included werden, alles andere kommt erst im Sourcefile hinzu. Dies beschleunigt die Kompilation.
- Für Pointer wird beispielsweise kein Include benötigt, es reicht eine sogenannte Forward Declaration:

```
class MyClass; // Forward declaration
MyClass* pointer = 0;
```

Styleguide

Using Namespace:

- Die Anweisung `using xyz`, wobei xyz der Name eines Namespaces ist, sollte vermieden werden.
- `using` macht den Code zwar etwas kürzer und einfacher, allerdings öffnet es Tür und Tor für weitreichende Konflikte zwischen Orxonox und Libraries, sowie sogar zwischen verschiedenen externen Libraries oder gar der STL.
- Als Folge davon schreiben wir zum Beispiel immer `std::` vor einen Begriff aus der Standardlibrary:

```
std::cout << „Hello World!“ << std::endl;
```

Styleguide

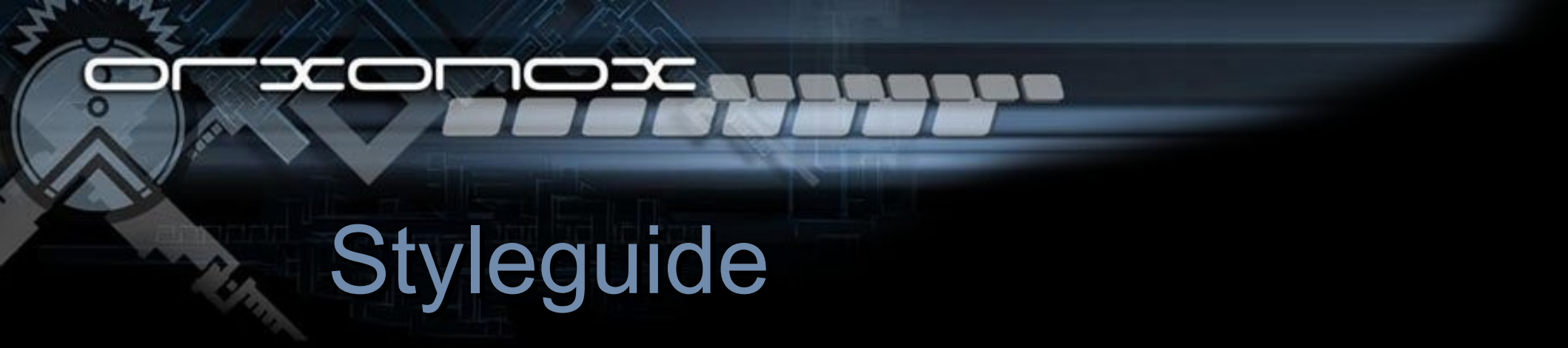
Kommentare:

- Codestellen die nicht auf ein oder zwei Blicke klar sind, sollten kommentiert werden:

```
for (int i = -10; i <= 10; i++)
    y += i*i * sgn(i) - x*(x = i); // Erklärung
```

```
if (i > 0)
{
    // Erklärung warum wir 10 ausschliessen
    if (i == 10)
        return;

    y = i;
}
```



Styleguide

Dokumentation:

- Variabeln, Funktionen, Klassen und Files können mit DoxyGen dokumentiert werden. Näheres steht auf der Wiki unter

`http://www.orxonox.net/wiki/Doxygen`

Styleguide

Mehr zum Styleguide:

- Den vollständigen Styleguide findet ihr unter

`http://www.orxonox.net/wiki/c++_styleguide`

- Oder über das Menü: Wiki → Development → Navigationsleiste → Programming: Styleguide