Inhaltsverzeichnis

1 Einleitung 3

2 Ordner- und Ablagestruktur 4

2.1. Einleitung 4

2.2. Richtlinien für die Ordner- und Ablagestruktur 4

2.3. Standardordnerstruktur für ein Projekt 5

3 Allgemeine Struktur einer Quellcodedatei 6

3.1. Einleitung 6

3.2. Richtlinien für die Strukturierung einer Quellcodedatei 6

3.3. Struktur einer Quellcode-Datei 7

4 Allgemeine Codierungsrichtlinien 8

4.1. Einleitung 8

4.2. Allgemeine Richtlinien 8

5 Namenskonventionen 9

5.1. Einleitung 9

5.2. Richtlinien für die Benennung (Allgemein) 9

5.3. Richtlinien für Benennung von Namensräumen 10

5.4. Richtlinien für Benennung von Klassen, Schnittstellen und Enumerationen 11

5.5. Richtlinien für Benennung von Methoden 11

5.6. Richtlinien für Benennung von Instanzvariablen 12

5.7. Richtlinien für Benennung von Properties 13

5.8. Richtlinien für die Benennung von Events und Delegates 13

5.9. Präfixe für die Benennung von Steuerelementen 14

6 Quellcode-Dokumentation 16

6.1. Einleitung 16

6.2. Richtlinien für die Quellcode-Dokumentation 16

6.3. Erzeugen der Quellcode API mit YUIDoc 21

7 Fehlerbehandlung 22

7.1. Einleitung 22

7.2. Richtlinien für die Fehlerbehandlung 22

8 Subversion 25

8.1. Einleitung 25

9 Abbildungsverzeichnis 26

# Einleitung

Die Entwicklungsrichtlinien bilden eine Grundlage zum vereinheitlichten Erstellen von Projekten. Sie legen unteranderem fest, wie Projektordner strukturiert, Quellcode zu codieren sind. Diese Konventionen sind umfassend in diesen Entwicklungsrichtlinien festgehalten und müssen angewendet werden.

Jedes Kapitel dieses Dokumentes ist nach dem gleichen Schema aufgebaut. Es beginnt mit einer kurzen Einleitung, welche die Motivation des jeweiligen Themas beschreibt. Anschließend werden die einzuhaltenen Richtlinen des Themas aufgeführt. In den nachfolgenden Abschnitten eines Kapitels werden diese Richtlinien erläutert und deren Umsetzung beschrieben.

Jede Richtlinie hat folgenden Aufbau:

1. **Nr.** – Jede Richtlinie erhält eine Nummer. In jedem Kapitel beginnt die Nummerierung bei 1.
2. **Name** – Kurze Beschreibung der Regel bzw. Richtlinie.
3. **Kategorie** – Kategorie zu der die Richtline gehört.
4. **Beschreibung** – Verständliche Beschreibung der Richtlinie.

|  |  |
| --- | --- |
| **§ 1** | <Kategorie> |
| *<Name der Richtlinie>* | |
| Erläuterung der Richtlinie und Ausführungsbestimmungen! | |

# Ordner- und Ablagestruktur

## Einleitung

Zur Reduzierung der Such- und Einarbeitungszeiten ist die Ordnerstruktur eines Projektes standardisiert. Dies gilt auch für die Projektordner und Projekte. Projektordner sind Ordner, die sich innerhalb eines nativen Projektes befinden z.B. Titanium-Projekt. Die Standardisierung der Ordnerstruktur bildet die Grundlage für die Arbeit mit Subversion (Haupt- und Nebenentwicklungszeigen) und für das Releasemanagement, sowie einen Ansatz für eine entsprechende Toolunterstützung.

## Richtlinien für die Ordner- und Ablagestruktur

|  |  |
| --- | --- |
| **§ 1** | Ordner- und Ablagestruktur |
| *Verwendung der Standardstruktur für Projektordner und Projekte* | |
| Für alle Projektordner und Projekte ist die definierte Ordnerstruktur einzuhalten. | |

|  |  |
| --- | --- |
| **§ 2** | Ordner- und Ablagestruktur |
| *Jedes native Projekt ist mindestens einem Projektordner zugeordnet* | |
| Jedes native Projekt ist Teil einem Projektordner. | |

|  |  |
| --- | --- |
| **§ 3** | Ordner- und Ablagestruktur |
| *Der Name eines SW-Elements muss eindeutig sein* | |
| Unterschiedliche SW-Elemente dürfen nicht denselben Namen haben, dies erhöht nicht nur das allgemeine Verständnis des Systems, sondern erlaubt auch eine eindeutige Zuordnung zwischen Dokumentation (Bereich: Systementwicklung) und Realisierung (Bereich: Implementation). | |

|  |  |
| --- | --- |
| **§ 4** | Ordner- und Ablagestruktur |
| *Es muss für jedes SW-Element ein Projektordner mit gleichen Namen geben* | |
| Für jedes SW-Element muss ein Projektordner existieren, welcher den gleichen Namen wie das SW-Element besitzt. Dies gilt auch für Prototypen, hier muss ein Projektordner mit dem Namen des Prototyps existieren. | |

|  |  |
| --- | --- |
| **§ 5** | Ordner- und Ablagestruktur |
| *Jeder Namensraum wird auf einen Ordner abgebildet* | |
| Innerhalb eines nativen Projektes wird jeder erzeugte Namensraum auf einen Ordner innerhalb des Ordners src abgebildet und die entsprechenden Quellcodedateien in diesem abgelegt. | |

## Standardordnerstruktur für ein Projekt

Im Nachfolgenden ist die Standardordnerstruktur für Projekte abgebildet. Die Bezeichnung Projekt ist bzgl. des allg. Projektbegriffes zu verstehen. Nachfolgend wird nur auf die Bereiche eingegangen, welche für einen Entwickler von besonderer Bedeutung sind.

| Ordner | Beschreibung |
| --- | --- |
| Dokumentation | Enthält Dokumentation z.B. Anleitungen, Technische - Dokumentation |
| Implementation | Enthält Prototype, Demoprojekte und Laufende Projekte |
| Management | Enthält Planung – und Steuerung Dokumente, Berichtswesen und Meetings |
| Releases | Enthält fertige Projekte, die zur Auslieferung bereit sind |
| Systementwicklung | Enthält Systemarchitektur und Standardprojekt |
| Tools | Enthält Tools, die zum Entwickeln benötigt werden |

# Allgemeine Struktur einer Quellcodedatei

## Einleitung

Um die Übersicht innerhalb einer Quellcode-Datei zu erhöhen, sollte jede Datei den gleichen Aufbau haben.

## Richtlinien für die Strukturierung einer Quellcodedatei

|  |  |
| --- | --- |
| **§ 1** | Struktur Quellcodedatei |
| *Jede Quellcodedatei ist gleich zu strukturieren.* | |
| Die Struktur muss der festgelegten Standardstruktur für eine Quellcodedatei entsprechen. | |

|  |  |
| --- | --- |
| **§ 2** | Struktur Quellcodedatei |
| *Jeder logisch zusammenhängende Block ist entsprechend seiner Umgebung einzurücken.* | |
| Eine Einrückebene besteht entweder aus einem Tab oder min. 2 bis max. 4 Leerzeichen. | |

|  |  |
| --- | --- |
| **§ 3** | Struktur Quellcodedatei |
| *Zwischen zwei Blöcken ist eine Leerzeile einzufügen.* | |
| Ein Block ist in diesem Zusammenhang folgendes: Methode, Property oder Konstruktor. | |

|  |  |
| --- | --- |
| **§ 4** | Struktur Quellcodedatei |
| *Pro Zeile steht nur eine Anweisung oder Zuweisung.* | |
| d.h. dass spätestens nach dem schließenden Semikolon ein Zeilenumbruch eingefügt wird. | |

## Struktur einer Quellcode-Datei

Im Nachfolgenden ist der schematische Aufbau einer Quellcode-Datei dargestellt.

* Requires
* Klassen-Dokumentation
* Klassen-Definition
  + Instanzvariablen
  + Konstruktoren
  + Properties
  + Public-Methoden
  + Private-Methoden

# Allgemeine Codierungsrichtlinien

## Einleitung

In diesem Kapitel werden allgemeine Richtlinien zur Codierung aufgeführt, welche grundsätzlich zu beachten sind.

## Allgemeine Richtlinien

|  |  |
| --- | --- |
| **§ 1** | Allgemein |
| *Vorgegebene Richtlinien sind einzuhalten!* | |
| Die für ein Projekt definierten Richtlinien für die Erzeugung von Quellcode, wie z.B. Namenskonventionen, ExceptionHandling, Logging, etc. sind einzuhalten. | |

|  |  |
| --- | --- |
| **§ 2** | Allgemein |
| *Für jede Klasse, Enumeration, Interface, etc. ist eine neue Quellcodedatei anzulegen, dessen Name identisch mit dem des enthaltenen Elements ist.* | |
| - | |

|  |  |
| --- | --- |
| **§ 3** | Allgemein |
| *Jede Quellcodedatei hat einen verantwortlichen Entwickler, den Autor.* | |
| - | |

# Namenskonventionen

## Einleitung

Dieser Abschnitt regelt die Benennungen von Variablen, Klassen, Methoden und sonstigen Elementen innerhalb einer Quellcodedatei. Die Namenskonventionen sollen die Einarbeitung in fremden Code erleichtern.

## Richtlinien für die Benennung (Allgemein)

|  |  |
| --- | --- |
| **§ 1** | Namenskonventionen (Allgemein) |
| *Kein Name darf mit einem Unterstrich beginnen.* | |
| - | |

|  |  |
| --- | --- |
| **§ 2** | Namenskonventionen (Allgemein) |
| *Bei zusammengesetzten Namen ist die Kamelnotation zu verwenden.* | |
| z.B. TimeSpan | |

|  |  |
| --- | --- |
| **§ 3** | Namenskonventionen (Allgemein) |
| *Aus Gründen der Interoperabilität ist die Groß- und Kleinschreibung als Unterscheidungsmerkmal von Variablen nicht ausreichend.* | |
| - | |

|  |  |
| --- | --- |
| **§ 4** | Namenskonventionen (Allgemein) |
| *Die Ungarische Notation <Präfix><Name> ist nur noch in der Oberflächenentwicklung zu verwenden.* | |
| - | |

|  |  |
| --- | --- |
| **§ 5** | Namenskonventionen (Allgemein) |
| *Umlaut und Akzente dürfen nicht verwendet werden. Ausnahme bei Kommentaren.* | |
| - | |

|  |  |
| --- | --- |
| **§ 6** | Namenskonventionen (Allgemein) |
| *Die Benennung erfolgt in englischer Sprache.* | |
| - | |

## Richtlinien für Benennung von Namensräumen

Für Namensräume gelten folgende Richtlinien:

|  |  |
| --- | --- |
| **§ 1** | Namenskonventionen (Namensräume) |
| *Jeder Namensraum beginnt mit einem Großbuchstaben.* | |
| - | |

|  |  |
| --- | --- |
| **§ 2** | Namenskonventionen (Namensräume) |
| *Der Name soll die darin enthaltenen Klassen und deren Funktionalitäten beschreiben.* | |
| Z.B. Util für Helper und Klassen einer Unterstützungsschicht | |

|  |  |
| --- | --- |
| **§ 3** | Namenskonventionen (Namensräume) |
| *Jede Projektmappe und somit jedes enthaltene Projekt hat seinen eigenen Namensraum (Stammnamensraum).* | |
| Dieser muss nicht eindeutig sein, da Systeme auf mehrere Projektmappen oder Projekte aufgeteilt werden. Für die Entwicklung von Bibliotheken gilt, dass der Stammnamensraum auch bei jeder enthaltenen Datei anzugeben ist. | |

|  |  |
| --- | --- |
| **§ 4** | Namenskonventionen (Namensräume) |
| *Jede Komponente, Teilsystem oder System hat seinen eigenen Namensraum.* | |
| Anwendungskomponenten besitzen immer den Namensraums des zugehörigen Systems. Der Namensraum des Systems ist immer der führende Namensraum, z.B. RASII.\*.  Ausgenommen sind technische oder Komponenten der Unterstützungsschicht, die in anderen Systemen oder Teilsystemen wieder verwendet werden. | |

## Richtlinien für Benennung von Klassen, Schnittstellen und Enumerationen

Für Klassen, Schnittstellen und Enumerationen gelten die folgenden Richtlinien:

|  |  |
| --- | --- |
| **§ 1** | Namenskonventionen (Klassen, Schnittstellen und Enumerationen) |
| *Klassen sind immer mit einem Substantiv zu benennen und beginnen mit einem Großbuchstaben.* | |
| - | |

|  |  |
| --- | --- |
| **§ 2** | Namenskonventionen (Klassen, Schnittstellen und Enumerationen) |
| *Schnittstellen sind immer mit einem Substantiv zu benennen und beginnen mit einem Großbuch­staben. Außerdem erhalten Schnittstellen den Postfix „IF“.* | |
| - | |

|  |  |
| --- | --- |
| **§ 3** | Namenskonventionen (Klassen, Schnittstellen und Enumerationen) |
| *Enumerationen sind immer mit einem Substantiv zu benennen und beginnen mit einem Großbuch­staben.* | |
| - | |

|  |  |
| --- | --- |
| **§ 4** | Namenskonventionen (Klassen, Schnittstellen und Enumerationen) |
| *Bei Vererbungen kann der Name der Oberklasse an die Unterklasse angehängt werden.* | |
| Bsp.: ApplicationException. | |

|  |  |
| --- | --- |
| **§ 5** | Namenskonventionen (Klassen, Schnittstellen und Enumerationen) |
| *Klassennamen sollten nicht das Schlüsselwort My enthalten.* | |
| - | |

## Richtlinien für Benennung von Methoden

Für die Benennung von Methoden gelten folgende Richtlinien:

|  |  |
| --- | --- |
| **§ 1** | Namenskonventionen (Methoden) |
| *Methoden enthalten ein Verb oder einen Imperativ.* | |
| - | |

|  |  |
| --- | --- |
| **§ 2** | Namenskonventionen (Methoden) |
| *Der erste Buchstabe einer Methode wird kleingeschrieben.* | |
| Allgemein werden Methoden in der .NET-Welt großgeschrieben und in der Java-Welt klein. Um den Umstieg zu erleichtern werden daher Methoden kleingeschrieben. | |

|  |  |
| --- | --- |
| **§ 3** | Namenskonventionen (Methoden) |
| *Parameter einer Methode werden klein geschrieben.* | |
| - | |

|  |  |
| --- | --- |
| **§ 4** | Namenskonventionen (Methoden) |
| *Getter- und Setter-Methoden* | |
| Sollte Getter- und Setter-Methoden notwendig sein, so erhalten diese den Präfix „get“ oder „set“. | |

## Richtlinien für Benennung von Instanzvariablen

In der Literatur werden Instanzvariablen auch als Member- oder Klassenvariablen bezeichnet. In .NET als Felder und in der UML als Attribute. Für die Benennung von Instanzvariablen gelten folgende Richtlinien:

|  |  |
| --- | --- |
| **§ 1** | Namenskonventionen (Instanzvariablen) |
| *Jede Instanzvariable hat die Sichtbarkeit private.* | |
| - | |

|  |  |
| --- | --- |
| **§ 2** | Namenskonventionen (Instanzvariablen) |
| *Instanzvariablen erhalten den Präfix „m\_“.* | |
| Dies soll die Lesbarkeit und das Verständnis des Quellcodes erhöhen, da durch den Präfix deutlich wird, dass es sich um eine Instanzvariable handelt. | |

|  |  |
| --- | --- |
| **§ 3** | Namenskonventionen (Instanzvariablen) |
| *Konstanten werden komplett in Großbuchstaben geschrieben.* | |
| Dies gilt auch für Werte eines Enums. | |

## Richtlinien für die Benennung von Events und Delegates

Für die Benennung von Events und Delegates gelten folgende Richtlinien:

|  |  |
| --- | --- |
| **§ 1** | Namenskonventionen (Events und Delegates) |
| *Delegates werden wie Methoden benannt, wobei die Pascal-Case Notation verwendet wird, d.h. der erste Buchstabe wird großgeschrieben.* | |
| - | |

|  |  |
| --- | --- |
| **§ 2** | Namenskonventionen (Events und Delegates) |
| *EventHandler erhalten den Namen des Events und als Postfix „EventHandler“.* | |
| Dies gilt nur für eigene Implementierungen von eigenen Handlern. | |

|  |  |
| --- | --- |
| **§ 3** | Namenskonventionen (Events und Delegates) |
| *Events werden immer mit einem Verb benannt und beginnen mit einem Großbuchstaben. Wenn es sich um ein Pre-Event handelt erhalten sie den Postfix „ing“ und wenn es sich um ein Post-Event handelt wird die Vergangenheitsform des Verbs verwendet.* | |
| Ansonsten erhält ein Event kein Prä- oder Postfix. | |

|  |  |
| --- | --- |
| **§ 4** | Namenskonventionen (Events und Delegates) |
| *Oberklassen, welche ein Event deklarieren, erhalten eine protected Methode mit dem Präfix „On“.* | |
| Erbt eine Klasse von einer anderen, welche ein Event deklariert, so sollte die Oberklasse eine protected Methode mit Präfix „On“ erhalten. Diese Methode kann dann in der Unterklasse entsprechend überschrieben werden und zum Auslösen des Events verwendet werden. | |

|  |  |
| --- | --- |
| **§ 5** | Namenskonventionen (Events und Delegates) |
| *Eine Klasse, die von EventArgs erbt, erhält den Postfix „EventArgs“.* | |
| - | |

## Präfixe für die Benennung von Steuerelementen

Bei der Benennung von Steuerelementen sind folgende Präfixe zu verwenden:

| Steuerelement | Präfix |
| --- | --- |
| Button | **cmd** |
| Canvas | **can** |
| CheckBox | **chk** |
| Fortschrittsleiste (ProgressBar) | **prg** |
| Kombinationsfeld (Dropdown-Listenfeld) | **cbo** |
| Kontrollkästchen (CheckBox) | **chk** |
| Label | **lbl** |
| Listenfeld (ListBox) | **lst** |
| Menü | **mnu** |
| Register (Tab) | **tab** |
| Schieberegler (Slider) | **sld** |
| ScrollView | **scr** |
| StackPanel | **stp** |
| Standard-Dialogfeld (CommonDialog) | **dlg** |
| Strukturansicht (TreeView) | **trv** |
| Symbolleiste | **tlb** |
| Tabelle (Grid) | **grd** |
| Textbox | **txb** |
| Textfeld | **txt** |
| UserControl (Benutzerdefiniertes Steuerelement) | *(Erhält keinen besonderen Präfix, sollte aber anhand des Namens deutlich werden)* |

# Quellcode-Dokumentation

## Einleitung

Eine der wesentlichen Hilfestellungen bei der Softwareentwicklung im Team ist die Quellcode-Doku­mentation. Diese erleichtert dem Entwickler die Verwendung von Klassen, Methoden etc., welche er nicht implementiert hat, da sie zum einen direkt im Code enthalten ist oder als API zur Verfügung steht.

## Richtlinien für die Quellcode-Dokumentation

## Erzeugen der Quellcode API mit YUIDoc

# Fehlerbehandlung

## Einleitung

## Richtlinien für die Fehlerbehandlung

# Subversion

## Einleitung

# Abbildungsverzeichnis