Vektorianische Namenskonvention

Version: 5.0

Date: Sonntag, 22. März 2015

Autor: Tobias Breiner

Inhaltsverzeichnis:

Sinn und Zweck der Namenskonventionen

1. Programmierende müssen sich nicht jedes Mal neu Gedanken über die Benennung ihrer Variablen und Klassen machen, denn die Benennung ist schließlich eindeutig definiert. Die Programmierarbeit wird dadurch zeitlich beschleunigt.

2. Die Variablennamen, die sich an die Namenskonventionen halten, sind in der Regel kürzer als selbst erdachte. Dies beschleunigt die Programmierarbeit und erleichtert das Lesen.

3. Auszüge des Codes sind trotz fehlenden Deklarationsteil bzw., Header trotzdem von ihrer Wirkungsweise her verständlich, das erleichtert die Didaktik und minimiert die Dokumentation.

4. Bei Teamarbeit kann es nicht mehr zu eventuellen Namenskonflikten kommen. Dies kann zwar auch durch das Konzept der „Namespaces“ geschehen, oftmals werden diese aber erst nachträglich eingesetzt, so dass Namenskonventionen einen zusätzlichen Schutz darstellen.

5. Programmierende finden sich schneller in einem fremden Projekt zurecht. Der Code ist monolithischer aufgebaut. Dies ist insbesondere bei Teamarbeit von entscheidendem Vorteil.

6. Programmierende durchschauen auch nach Jahren schneller ihren eigenen Code.

7. Die Namen sind meist selbstkommentierend. Daher sind weniger Kommentare für Variablendokumentationen in Source Code erforderlich.

8. Das Kopieren von Quellcode ist unproblematisch. Es sind keine anschließenden mühsamen Namensersetzungen mehr notwendig.

9. Das Suchen von Dateien/Ordnern/Variablen im Quellcode etc. geht wesentlich schneller von sich, da Programmierende nur nach einem einzigen fest definierten Namen suchen müssen, anstatt alle Möglichkeiten auszuloten. Dies ist besonders bei großen Projekten von entscheidendem Vorteil.

10. C++ Programmierfehler, die durch fehlendes Casting auftreten, werden durch die Typpräfixierung leichter entdeckt. Zwar gibt es dafür mittlerweile Programmierhilfen, diese sind aber nicht immer vorhanden (ältere IDEs, Zeitverzögerung beim Laden der Semantikdaten großer Projekte). Auch der Umgang mit Pointern wird erleichtert. Viele Pointerfehler können zudem nicht automatisch erkannt werden, zum Beispiel eine Zuweisung von Pointer-auf-Pointer-Variablen auf nur Pointer-Variablen. Der Code wird somit weniger fehleranfällig sein.

11. Redundante – und damit oftmals unsinnige – Codeteile werden sofort optisch ersichtlich, da der dazugehörige Quelltext identisch oder zumindest ähnlich lautet. Dies wäre bei unterschiedlichen Variablenbenennungen ohne Suffixe nicht der Fall. Dies minimiert den Code und begünstigt eine elegante Programmierung.

Bei den Namenskonventionen ist auf folgende Eigenschaften geachtet:
- Kürze
- Einfachheit
- Eineindeutigkeit
- Stringenz
- Redundanzvermeidung

Allgemeines zu den Namenskonventionen

Die allgemeinen Namenskonventionen gelten für alle Namen. Sie sind somit verbindlich für:

1. Dateinamen
2. Namen für Dateiordner
3. Variablennamen
4. Klassennamen
5. Define-Namen
6. Namen für Objekte der Szenegraphen

Sprache

Benennungen erfolgen ausschließlich in Englisch, denn es ist nicht auszuschließen, dass Quellcode in Zukunft auch von ausländischen Studierenden wiederverwendet wird, die oftmals nicht der deutschen Sprache mächtig sind. Auch sind englische Wörter meist kürzer als deutsche und haben keine Umlaute. Eventuelle Kommentare können durchaus auch in Deutsch formuliert werden, da sich durch die Möglichkeit der unbegrenzten Kompositabildung im Deutschen Dinge oftmals genauer beschreiben lassen und der Kommentar durch fehlende Redundanz einen echten Mehrwert zum Code bietet.

Beispiele

falsch	***richtig***
bAktiv	bActive
bAktuellerPlayer	bActualPlayer
// Shows the frame rate:	// Zeigt die Bildwiederholrate als Konsolentext in fps an:
ShowFramerate();	ShowFrameRate();

Sonderzeichen

Sonderzeichen wie Umlaute (Ä,Ö,Ü,ä,ö,ü), scharfes S (ß) oder Symbole (!?) sind in Namen zu vermeiden. Umlaute werden durch die entsprechenden Doppelvokalsalternativen ersetzt. (Ae, Oe, Ue, ae, oe, ue). Das scharfe „ß“ wird durch zwei normale „S“ ersetzt. Einziges erlaubtes Sonderzeichen ist in besonderen Fällen der Unterstrich.

Beispiele

falsch	***richtig***
MÖLLER.BMP	Moeller.bmp
RIGHT?.BAT	IsRight.bat

Polnische Notation

Jedes neue Wort in zusammengesetzten Wörtern sollte groß beginnen, um Verwechslungen zu vermeiden (auch als CamelCase bezeichnet). Ausnahme ist der Präfix, der den Variablentyp definiert. Er ist immer klein geschrieben. Im Zweifelsfall sollten die Teilwörter der Komposita auch mit Großbuchstaben beginnen:

Beispiele

falsch	***richtig***
Cnamelist	CNameList
bDrivinglicence	bDrivingLicence
piHighscorepage	piHighScorePage

Eindeutigkeitsgesetz

Jeder Namen sollte das Attribut bzw. das Objekt für das er steht, eindeutig und unmissverständlich beschreiben.

Beispiele

falsch	***richtig***
CChnls	CChannelList
iHomie	iCheckerInHome

Redundanzvermeidungsgesetz

Jegliche Tautologien innerhalb des Variablennamens sind zu vermeiden. „Weiße Schimmel“, „prollige Asoziale“ und „böse Bugs“ haben im Quellcode nichts zu suchen! Insbesondere gilt, dass im Corpus keine Information stehen sollte, die nicht schon durch den Präfix gegeben wurde. Häufige diesbezügliche Fehler sind x Bezeichnungen wie Counter, Nr., Integer in Variablennamen. Sie sind ein Anzeichen dafür, dass irgendetwas in der Benennung faul ist, da diese Eigenschaften schon durch das Präfix i gekennzeichnet sein sollte.

Beispiele

falsch	***richtig***
CBuildingClass	CBuilding
iCounter	i
iPersonNr	iPerson
iMaxNumbersOfPersons	iPersons
fMaxValueOfPressure	fMaxPressure
bBlinkerSwitchIsOn	bBlinkerOn

Abkürzungsgesetz

Abkürzungen sind nicht erlaubt, da sie einerseits zu Inkonsistenzen zwischen Abkürzungen und Vollwörtern führen und andererseits missverständlich sein können.
Ausnahmen sind folgende häufig auftretenden Abkürzungen. Diese sind aus Konsistenzgründen allerdings zwingend:

Abkürzungsliste	
Application	App
Boolean	Bool
Button	Btn
Color, Colour	Col
Dialog	Dlg
Directory	Dir
Identification Number	ID
Initialization file	Ini
Information	Info
Integer	Int
Floating Point Number	Float
Fraction Floating Point Number	Fract
Matrix	Mat
Maximum	Max
Memorize	Memo
Memory	Mem
Message	Msg
Middle	Mid
Minimum	Min
Previous	Prev
Source code	Src
Synchronize, synchronization	Sync
Temporary	Temp
Topapplication	Top

Spezielle Namenkonventionen

Dateinamen

Dateinamen sollen möglichst kurz und prägnant sein, plus ein dreistelliges Suffix hinter dem Punkt. Als Sonderzeichen ist einzig und allein der Unterstrich und der Bindestrich ist in Notfällen erlaubt, falls es ohne sie zu Verwechslungen kommen könnte.

Beispiele für Dateinamen

falsch	**richtig**
DR_HÄCK.EXE	DrHaeck.exe
MYFILE.BAT	MyFile.bat

Datumsangaben in Dateinamen

Sollte ein Datum in einem Dateinamen notwendig sein, so wird es nach der Beschreibung der Datei in CamelCase mit einem vorangestelltem Unterstrich eingeleitet, dann kommt erst das Jahr, in der Mitte der Monat und schließlich der Tag, jeweils mit Bindestrichen getrennt. Falls Tag oder Monat nicht bekannt sein sollten, werden sie weggelassen. Verschiedene Versionen innerhalb eines Tages werden alphabetisch mit Kleinbuchstaben nach dem Datum gekennzeichnet. Auf diese Weise werden Dateien automatisch alfabetisch richtig sortiert.

Beispiele für Dateinamen mit Datum

falsch	**richtig**
Oak-9.11.2009-Version1.jpg	Oak_2009-11-09a.jpg
Oak-2.Jan.2010-Version3.jpg	Oak_2010-01-02c.jpg
Leaf-April.2010-Version5.jpg	Oak_2010-04e.jpg

Dateinamen für C++ Source-Codes

Beim Source-Code sollte jede Klasse, die nicht in einer anderen Klasse gekapselt ist, in eine Datei gleichen Namens ohne vorausgehendes C abgespeichert werden.

Beispiele für Dateinamen für C++ Source Codes

Landscape.h	Header-Datei für die Klasse CLandscape
Sucker.cpp	Body-Datei für die Klasse CSucker

Namen für Dateiordner

Namen für Dateiordner bestehen aus maximal acht Zeichen(Großbuchstaben + Ziffern). Der Ordername sollte kein Suffix verwenden. Als Sonderzeichen ist einzig und allein der Unterstrich in Notfällen erlaubt, falls es ohne ihn zu Verwechslungen kommen könnte.

Beispiele für Dateiordner

falsch	***richtig***
TextureDir	Textures
DirectoryShader	

Klassennamen

Namen für C++ Klassen beginnen mit einem großen C. Danach folgt direkt anschließend in polnischer Notation das Objekt, für das die Klasse steht.

Besonderheiten bei Klassennamen

- Eine Klasse, die Hauptsächlich States verwaltet, ist mit dem Wort „Page“ am Schluß gekennzeichnet.
- Eine Klasse, die andere Objekte (Klassen) verwaltet hört mit dem Wort „Manager“ auf.

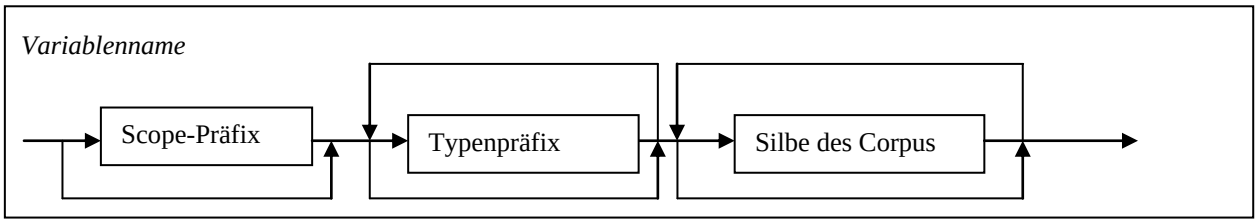
Beispiele für Klassennamen

falsch	***richtig***
CChannelInfo	CChannel
CCoordinateThing	CThingManager
cPlayerManager	CPlayerPage

Variablennamen

Jeder Variablenname besteht aus maximal Teilen
1. das Memberpräfix
2. das Typenpräfix
3. den Corpus

Skizze



Scope-Präfix
Das Scope-Präfix ist optional. Wenn es vorhanden ist, zeigt es an, ob es sich um eine globale Variable, eine Staticvariable oder eine Membervariable handelt. Es steht stets am Anfang und wird mit einem Unterstrich abgeschlossen.

Beispiele für Scope-Präfixe:

g_	Die Variable ist eine globale Variable (möglichst vermeiden)
m_	Die Variable ist im Header einer Klasse deklariert und ist eine Membervariable
s_	Die Variable ist eine Static-Variable in einer Routine.

Typenpräfix
Das Typenpräfix gibt den Variablentyp an.

Typenpräfix für elementare Variablen:

b	bool, BOOL
i	int
f	float
d	double
c	char
l	long
s	short
e	enum

Typenpräfix für spezielle Integer-Variablen:

i (without Corpus)	Integer-Laufvariable für einfache „for“-loops
j (without Corpus)	Integer-Laufvariable für einfache „for“-loops
k (without Corpus)	Integer-Laufvariable für einfache „for“-loops
id	Integer-ID
ix	integer coordinate auf X-Achse
iy	integer coordinate auf Y-Achse
iz	integer coordinate auf Z-Achse

Typenpräfix für spezielle Float-Variablen

fx	floating point number coordinate	(X-Wert bei Vektoren)
fy	floating point number coordinate	(Y-Wert bei Vektoren)
fz	floating point number coordinate	(Z-Wert bei Vektoren)
fr	fraction (0..1)	(Anteil von 0 bis 1)
fa	floating point angle in radians	(Winkel im Bogenmaß)

Typenpräfix für spezielle Double-Variablen

dx	double floating point number coordinate
dy	double floating point number coordinate
dz	double floating point number coordinate

Typenpräfix für Pointer

p	pointer to
pb	bool*, BOOL*
pi	int*
pf	float*
pd	double*
pc	char*
ps	short*
ptroll	Pointer auf die Klasse CTroll
phobbit	Pointer auf die Klasse CHobbit

Typenpräfix für Arrays

a	Array
aa	Zweidimensionales Array
aaa	Dreidimensionales Array
ai	array of integers
af	array of floats
ad	array of doubles
ac	array of chars (=Zeichenkette mit fixer maximaler Länge)
atroll	eindimensionales Array auf die Klasse CTroll
aai	Zweidimensionales Array of integer
aaaf	Dreidimensionales Array of floats
etc.	

Typenpräfix für spezielle Arrays

ac	char[index]

Typenpräfix für spezielle Typen

pt2	POINT oder rtPoint2
pt3	POINT3 oder rtPoint3
rect	RECT
h	HANDLE

Typenpräfix für spezielle Objekte

st	CString

Typenpräfix für Geometrische Szenegraph-Variablen

v	VECTOR, rtVector oder andere 3D-Vektoren mit 4 Skalarwerten
m	MAT, rtMatrix oder andere 4*4 Matrizen
q	für Quaternionen

Beispiele für die Benennung von Variablen:

```
int iTroll;                     // Trollnummer
int iTrolls;                    // Trollanzahl
int idTroll                     // Troll-ID
int* piTroll;                   // Pointer auf die Trollnummer
float fTrollStrength            // Trollstärke
double dTrollWeight             // Trollgewicht
CString stTrollNickName         // Spitzname des Trolls
int aiTroll[100];               // eindimensionales Array von Trollnummern
int aiTrolls[100];              // eindimensionales Array von Trollanzahlen
float afTrollStrength[100];     // eindimensionales Array von Trollstärken
CTroll atroll[100];             // eindimensionales Array von Trollen (= von CTroll)
```

Bei so viel Trollen, die sich hier tummeln, wäre es aber angebracht eine eigene Troll-Klassse zu erzeugen. Hier ein besseres Beispiel:

```
class CTroll
{
public:
        void CTrolls();
        void ~CTrolls();
        void Init();
        void Tick(float fTimeDelta);
        void Fini();


public:                                         // Objektorientiertheit, heißt nicht,
                                                // dass man alle Member-Variablen private sind
                                                // sie sind ja eh schon durch die Klasse gekapselt
        float GetStrength();
        void SetStrength(float fStrength);

        int m_id;                                       // Troll-ID
        float m_fStrength;                              // Troll-Stärke
        double m_dWeight;                               // Troll-Gewicht
        CString m_stNickName;                           // Troll-Spitzname
        CString m_stFamilyName;                         // Troll-Familienname
        CHVector m_vHead;                               // Troll-Kopf-Koordinaten
        CHVector m_vTail;                               // Troll-Schwanz-Koordinaten
        enum {sNormal, sAngry, sExited, sStoned,
                        sFearfull, sSad, sLucky}eMood;  // Troll-Laune
private:
        CString m_stSecretName  ;                       // Troll-Geheimname
        rtID m_rp;                                      // Placement des Trolls
        rtID m_rs;                                      // Shape des Trolls;
}

class CTrolls
{
public:
        void CTrolls();
        void ~CTrolls();

        CTroll m_atroll[100];                           // Array of Trolls
        int m_iMax;                                     // Anzahl der Trolle
}
```

CPP

```
Int CTroll::SetStrenth(float fStrenth)
{
        m_fStrength = fStrength;
}
```

Noch mehr Beispiele

```
BOOL aabSquare[8][8];                   // zweidimensionales Array (8*8) von BOOLschen Werten
enum {sRed, sYellow, Green}eTrafficLightColor;   // Ampelstati
HANDLE hWindow;                         // Handle auf Window
HANDLE hGameThread;                     // Handle auf den Game Thread
```

Spezielle Variablennamen

acIni complete path of a initialization file

Typennamen
Typennamen haben das Präfix „t_“. Danach folgt die Typenbenennung analog zur Benennung von Variablennamen.

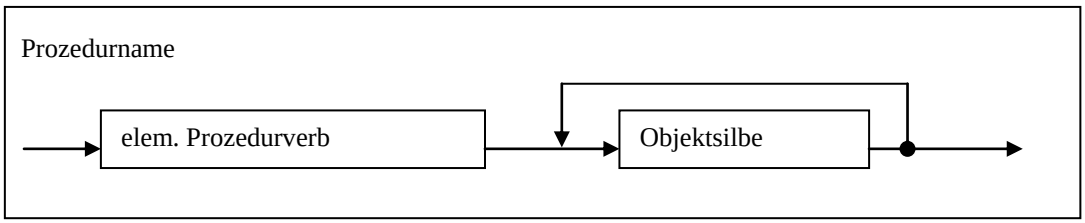
Beispiele:

t_aai	Typ auf ein zweidimensionales Integerarray
t_hWindow	Typ auf ein Windowhandle
t_apxyzTree	Typ auf ein Array von Pointern auf ein Punkt im dreidimensionalen Raum, welches die Position eines Baumes im euklidischen Raum kennzeichnet

Prozedurnamen

Ein Prozedurname besteht aus einem Verb mit anschließendem optionalem Objekt. Der Prozedurname ist ausschließlich in CamelCase geschrieben. Das Verb sollte wenn möglich einer der folgenden elementaren Prozedurverben sein.

Skizze



Elementare Prozedurverben

- Get — Prozeduren, welche Werte ermitteln, oder geschützte Variablen ausgeben
- Set — Prozeduren, die einen Status bzw. eine geschützte Variable auf einen Wert setzen.
- Add — Prozeduren, die Objekte zu einer Liste hinzufügen
- Sub — Prozeduren, die Objekte aus einer Liste herausnehmen
- Reset — Prozeduren, welche Variablen bzw. Stati auf ihre Ursprungswerte zurücksetzen
- Do — Prozeduren, die einen Vorgang durchführen
- Undo — Prozeduren, die einen Vorgang rückgängig machen
- Start — Prozeduren, welche einen Vorgang starten
- Stop — Prozeduren, welche einen Vorgang anhalten, und die Aktivität weitergeben
- Pause — Prozeduren, welche einen Vorgang anhalten
- Continue — Prozeduren, welche einen angehaltenen Vorgang fortsetzen
- Make — Prozeduren, die fremde Objekte erzeugen
- Kill — Prozeduren, die fremde Objekte zerstören
- Create — Prozeduren, die eigene Objekte erzeugen und Speicher allokieren
- Release — Prozeduren, die eigene Objekte erzeugen und Speicher deallokieren
- Draw — Prozeduren, die Objekte auf den Bildschirm zeichnen
- Redraw — Prozeduren, die Objekte auf dem Bildschirm entweder neu zeichnen oder wegradieren.
- Show — Prozeduren, die Objekte sichtbar machen
- Hide — Prozeduren, die Objekte verstecken
- Push — Prozeduren, die irgendwas auf einen Stapel legen
- Pop — Prozeduren, die irgendwas von einem Stapel holen
- Calc — Prozeduren, welche komplexe Rechnungen durchführen und ein Ergebnis liefern
- Memo — Prozeduren, welche sich bestimmte Werte oder Zustände merken
- Remember — Prozeduren, welche sich gemerkte Werte zurückrufen
- Check — Prozeduren, welche eine Klasse, bzw. eine Variable auf ihre Richtigkeit überprüft
- Save — Prozeduren, welche irgendwas auf Festplatte o.ä. abspeichern
- Load — Prozeduren, welche irgendwas von der Festplatte, ö.ä. laden
- Put — Prozeduren, welche Werte von einer Klasse in eine andere schaufeln
- Enable — Prozeduren, welche einen Vorgang erlauben (z.B. Buttonklick)
- Disable — Prozeduren, welche einen Vorgang verbieten

- Init Prozeduren, welche die Werte einer Klasse initialisieren
- Tick Prozeduren, welche pro Schleifendurchgang Simulationswerte neu berechnen
- Fini Prozeduren, welche die Werte einer Klasse deinitialisieren
- Is Prozeduren, die eine ISA-Frage beantworten und einen Boolschen Wert zurückliefern
- Has Prozeduren, die eine ISP-Frage beantworten und einen Boolschen Wert zurückliefern

Ersetzen nichtelementarer Prozedurverben
Der Programmierer sollte, wenn möglich, auf diese elementaren Verben bei der Benennung von Prozedurnamen zurückgreifen. Dabei sollte er nichtelementare Prozedurverben durch elementare ersetzen.

Beispiele für die Ersetzung nichtelementarerer Prozedurverben durch elemenetare
Begin -> Start
End, Halt ->Stop
Sleep, Halt -> Pause
GoOn, Go, Walk, Awake -> Continue
Build, Create -> Make
Refresh -> Reset
Paint -> Draw
Destroy, Terminate -> Kill
Put -> Push
Allow -> Enable
Forbit -> Disable
Store -> Save

Komplementäre Prozedurverben
Sollte zu einer bereits existierenden Funktion eine Gegenfunktion programmiert werden, stehen die komplementären Prozedurverben zur Verfügung, die aus folgender Liste ersichtlich sind:

Init <-> Fini
Get <-> Set
Add <-> Sub
Do <->Undo
Start <-> Stop
Pause <-> Continue
Make <-> Kill
Create <-> Release
Draw <-> Redraw
Show <-> Hide
Push <-> Pop
Memo <-> Remember
Save <-> Load
Enable <-> Disable
Create <-> Release
Is <-> !Is
Has <-> !Has

Benennung von Listen-Prozeduren
Listen-Prozeduren sollten folgende Namen haben:

GetNext	Gibt nächstes Element der Liste
GetPrev	Gibt vorheriges Element der Liste
GetHead	Gibt erstes Element der Liste
GetTail	Gibt letztes Element der Liste
Add	Hängt ein Element an die Liste an
Sub	Löscht ein Element aus der Liste

Redundanzvermeidung mit der eingebundenen Klasse
Prozedurnamen enthalten die Information, welche durch die übergeordnete Klasse gegeben ist, nicht mehr.

Beispiele für die Benennung von Prozedurnamen

falsch	***richtig***
int CChannel::GetChannelNr()	int CChannel::GetNr()
void CButSprite::DrawButSprite()	void CButSprite::Draw()
void CHouse::StoreHouse()	void CHouse::Store()

Define-Konstantennamen

Konstanten, die mit „#define“ definiert wurden, werden ausschließlich mit Großbuchstaben, Ziffern und Underscores geschrieben. Die Einzelnen Wörter werden durch die Underscores getrennt.

Der Define-Konstantenname beginnt mit einem Kürzel mit einem Underscore, welches das Projekt eindeutig definiert. Darauf folgt die Bezeichnung.

Fixe Integer-Konstanten beginnen mit I_

Beispiele für die Benennung von Defines

```
#define I_BUTS
#define GEN_PLAYER 1                  // 3D-Generation-Projekt, Player
#define GEN_COMPUTER 0                // 3D-Generation-Projekt, Computer
#define SB_ PIXELS_PER_TILE_EDGE      // slaughtybartfas-Projekt,  maximale Pixelanzahl auf einer
                                      Tile-Kante
```

Szenegraph-Namen für Knotenobjekte

Die Klassen in Vektoria, die Knotenobjekte repräsentieren, fangen (fast) alle mit einem anderen Buchstaben an. Es gibt zurzeit folgende Buchstaben:
A: CAudio
B: CBackground
C: CCamera
D: CDevice
E: CEmitter
F: CFrame
G: CGeo
H: CHardware
I: CImage
J: *(frei)*
K: CKeyframe *(angedacht)*
L: CLight *(=> LL: ParallelLight, LP: PointLight, LS: SpotLight)*
M: CMaterial
N: CNode
O: COverlay
P: CPlacement
Q: *(frei)*
R: CRoot
S: CScene
T: *(frei)*
U: CUnion *(angedacht)*
V: CViewport
W: CWriting, CWritingChar, CWribel
X: *(frei)*
Y: *(frei)*
Z: *(frei)*
Die Buchstabenunterschiedlichkeit in Vektoria ist kein Zufall. Damit lässt sich gezielt Schreibarbeit durch leicht zu merkende mnemotechnische Abkürzungen vermindern. Zum Beispiel können die Präfixe für instanziierte Variablen in Vektoria immer mit einem „z“ und dem nachfolgenden Anfangsbuchstaben des Knotenobjektes abgekürzt werden:
za: CAudio; zas: CAudios
zb: CBackground; zbs: CBackgrounds
zc: CCamera; zcs: CCameras
zd: CDevice; zds: CDevices
zf: CFrame; zfs: CFrames
etc.

Für die mathematischen Klassen gibt es auch Präfixe:
v: CHVector; vs: CVHVectors
m: CHMat; ms: CVHMats
q: CQuaternion; qs: CQuaternions
r: CRay; rs: CRays

Typenpräfix für Szenegraph Variablen
Szenegraphenvariablen beginnen stets mit einem „z“ und einem zusätzlichen Buchstaben für das Szenenkontenobjekt.

Präfix	***Vektoria***	***Ogre***
zr	CRoot	
zh	CHardware	
zv	CViewport	
zd	CDevice	
zs	CScene	
zp	CPlacement	Entity
zg	CGeo	StaticGeometry
zgq	CGeoQuad	
zge	CGeoEllipoid	
zgc	CGeoCube	
zgt	CGeoTetraeder	
zgi	CGeoIkosaeder	
zt	CTexture	
zm	CMaterial	
zi	CImage	
zl	CLight	Light
zc	CCamera	
ze	CEmitter	

Beispiele:
```
CEmitter m_zeFire;      // Emitter für ein Lagerfeuer
CViewport m_zvLeft;     // Linker Viewport
CCamera m_zcBird;       // Vogelperspektivkamera
```

Parameterreihenfolge

szIni kommt immer zuletzt
szSection kommt vor szKey