Clean Code Policy

Version 1.0

April, 2007

Author: J. Blauth, blauth@pracma.de

Inhalt

1	Preface	3
2	VB-based programming	4
2.1	Nameskonventionen	4
2.2	Variablen, Prozeduren und Konstanten	6
2.2.1	Variablen	6
2.2.2	Prozeduren	7
2.2.3	Konstanten	8
2.3	Writing Code	9
2.4	Coding rules	9
2.4.1	Kommentare	11
2.4.2	Code-Formatierung	12
2.5	Best Practises	13
3	SQL	14
3.1	Namenskonventionen und Styles	14
3.2	Best Practises	17
3.2.1	Database Design und Architektur	17
3.2.2	Coding	18
3.2.3	Queries und Optimierung	19
4	Boilerplates	20
4.1	Errorhandling	20
5	History	21

1 Preface

Die Clean Code Policy der PRACMA Ltd. stellt einen Leitfaden zur einheitlichen Programmierung dar. Auch wenn die Clean Code Policy in den meisten Fällen und in der Regel anwendbar ist, wird es während der Realisation einer Programmierung immer wieder Situationen geben, in denen eine strikte Umsetzung der Clean Code Policy keinen Sinn macht – diese Fälle sind im Code zu vermerken (siehe hierzu das Kapitel „Kommentare“).

Grundsätzlich versucht die Clean Code Policy nicht nur die Vereinheitlichung von geschriebenem Code innerhalb des Unternehmens zu optimieren, sondern auch eine Grundlage dafür zu bilden, dass Code immer den Leitsätzen

- easy to read
- easy to maintain
- reusable
- performing as expected

entsprechen kann (und soll!) und somit auch die Qualität des geschrieben Codes zu verbessern.

Erweiterungen können gerne über das **weboffice** zur Diskussion gestellt werden.

J. Blauth

2 VB-based programming

2.1 Namenskonventionen

2.1.1 Standardsprache für Identifier, Comments etc. ist Englisch.

2.1.2 Prozeduren sind in „upper Camel-case“ zu benennen. Dabei ist der Name deskriptiv und enthält in der Regel keine Abkürzungen, außer diese sind allgemein gebräuchlich oder mindestens projektweit geläufig.

2.1.3 Variablennamen beschreiben die Nutzung und nicht den Typ, sind in „upper CamelCase“, haben Hungarian Präfixe und entsprechen dem regulären Ausdruck ([a-zA-Z][a-zA-Z0-9]+)

2.1.4 Typ-Präfixe sind

Datentyp	**Präfix**	**EXAMPLE**
Boolean	`b`	`bIsValid`
Byte	`byt`	`bytAge`
Currency	`cur`	`curPrice`
Date and Time	`dat`	`datStart`
Double	`d`	`dScale`
Integer	`i`	`iCounter`
Long	`l`	`lHighCounter`
Single floating Point	`f`	`fSize`
String	`s`	`sName`
Variant	`v`	`vUnknown`
Object	`o`	`oAccess`
Collection	`col`	`colCustomer`
User defined Type	`t`	`tPoint`
Enum	`e`	`eGender`

2.1.5 Variablennamen sind immer in der Einzahl

2.1.6 Boolean Variablen immer mit „Can“, „Has“, “Is” oder ähnlichem benennen `bIsValid`, `bHasChanged`

2.1.7 Arrays haben kein speziellen Präfix, sondern tragen den Präfix aus dem Datentyp. Arrays tragen einen Namen in der Mehrzahl und können zusätzlich das Postfix „Array“ führen.
`sNames`, `sNamesArray`

2.1.8 Abkürzungen als Identifier sind zu vermeiden, ausser diese sind projektweit bekannt und/oder der ausgeschriebene Name würde zu lang werden. Abkürzungen von mehr als 5 Buchstaben sind zu vermeiden. Sind Abkürzungen max 3 Buchstaben lang, werden alle Buchstaben groß geschrieben, bei mehr Buchstaben wir nur der erste Buchstabe groß geschrieben.

2.1.9 Enumerations und Members einer Numeration wird das `pra` Firmen-Präfix vorangestellt um sie von VB-Enums zu unterscheiden.

```
Public Enum praGender
    praMale
    praFemale
End Enum
```

2.1.10 Es werden folgende Scope-Präfixe für verwendet

2.1.10.1Variablen

Scope	Präfix	EXAMPLE
Funktionsvariablen		bIsValid
Classvariables/ Modulvariables	c_ [1]	c_bIsValid
Global	g_	g_bIsValid

2.1.10.2Konstanten

Scope	Präfix	EXAMPLE
Funktionsvariablen		CONSTANT
Classvariables/ Modulvariables	c [2]	cCONSTANT
Global	g	gCONSTANT

2.1.11 Konstanten immer in Großbuchstaben, einzelne Wörter sind mit „_“ getrennt. Ein Typpräfix wird nicht verwendet. Scope-Präfix wird verwendet
cIS_VALID, gIS_VALID, IS_VALID

2.1.12 Modul- und Element-Präfixe

Modul	Präfix	EXAMPLE
Module	m	mTools
Classes	c	cDatahandler
Forms (Instanz)	frm	frmMain
Interfaces	I	IMenu
Elemente (Access)	**Präfix**	**EXAMPLE**
Formulare	frm_	
Abfragen	qry_	
Berichte	rpt_	
Tabellen	tbl_	
Textbox	txt_	
Listbox	lb_	
Kombobox	cb_	
Control, allg.	ctr_	
Button	cmd_	
ActiveX-Komponente	Ax_	
Radiobutton	rb_	

[1] Wenn eine Variable mit „_“ beginnen kann (bsw in VB.net), kann das „c“ weggelassen werden. In einem Projekt wird entweder die Schreibweise mit oder ohne „c“ benutzt.
[2] Wenn eine Variable mit „_“ beginnen kann (bsw in VB.net), kann das „c“ konform zu 2.1.10.1 durch ein „_“ ersetzt werden.

2.2 Variablen, Prozeduren und Konstanten

2.2.1 Variablen

2.2.1.1 Immer `Option Explicit`

2.2.1.2 Dimensionierung immer in Verbindung mit explizitem Typ

2.2.1.3 Es werden ausschließlich C#-native Datentypen und keine CTS-Datentypen verwendet (nur .net)

```
'BAD
Dim iVar As Int32
```

2.2.1.4 Objektvariable werden nicht mit `As New` deklariert. Erst wenn die Objektinstanz benötigt wird, wird das Objekt entweder mit `Set = New` oder mit `CreateObject` deklariert.

```
'BAD
Dim oAccess As New Access.Application

'GOOD
Dim oAccess As Access.Application
Set oAccess = New Access.Application

'GOOD
Dim oAccess As Object
Set oAccess = CreateObject("Access.Application")
```

2.2.1.5 Variablen werden immer on-top dimensioniert.

2.2.1.6 In Prozeduren mit Rückgabewert, wird der Rückgabewert als letzte Variable dimensioniert. Die Rückgabevariable heißt immer Return mit entsprechendem Typpräfix. Der Rückgabewert wird immer direkt nach der Dimensionierung mit einem Default-Wert initialisiert

```
'EXAMPLE
Dim vSomeVariable As Variant
Dim sReturn As String
sReturn = „<kein Eintrag>“
```

2.2.1.7 Variablen werden Nutzungsabhängig deklariert. D.h. deklariere nicht einmal eine Variable die für mehrere verschiedene Dinge verwendet wird.

2.2.1.8 String-Konkatenationen werden mit `&` und nicht mit `+` gebildet.

2.2.1.9 Boolesche Variable werden positiv benannt.

```
'GOOD
bIsValid

'BAD
bIsNotValid
```

2.2.2 Prozeduren

2.2.2.1 Alle Prozeduren haben einen klar definierten Scope (`Public`, `Private`, `Friend`).

2.2.2.2 Prozeduren haben möglichst nur einen Austrittspunkt.
`Exit Sub/Function/Property` sind zu vermeiden, außer es verbessert die Lesbarkeit und/oder Performance.

2.2.2.3 Prozeduren haben „robust exit code“, d.h. das bsw. Objekt Referenzen explizit aufgelöst werden.

2.2.2.4 Parameter von Funktionen haben immer einen Datentyp.

2.2.2.5 Eine Function ohne Rückgabewert gibt es nicht.

2.2.2.6 `ByVal` und `ByRef` werden immer explizit angegeben.

2.2.3 Konstanten

2.2.3.1 Konstanten sind in der Regel immer sogenannten „magic numbers“ (Werten, die keine explizite Deklaration besitzen) vorzuziehen. Ausnahmen bilden nur bsw. Initialisierungen von Variablen auf 0 oder 1 (bsw. Counter in Loops)

```
'BAD
for i=0 to 12
    (...)

'GOOD
for i=0 to MY_MAX_NUMBER
    (...)
```

2.3 Writing code

2.3.1 Coding Rules

2.3.1.1 Jede nicht triviale Prozedur enthält ein Errorhandling nach Error-Boilerplate.

2.3.1.2 Es werden nur Head-based Schleifen verwendet.

```
'GOOD
do while true
    (...)
loop

'BAD
do
    (...)
loop while true
```

2.3.1.3 While… Wend wird nicht genutzt.

2.3.1.4 select case-Strukturen haben immer einen default case (case else).

2.3.1.5 Vergleiche boolesche Variablen niemals gegen true oder false.

```
'GOOD
if bIsValid then (...)

'BAD
if bIsValid = true then (…)
```

2.3.1.6 Programmiere nach “Keep-Simple-Style“.

2.3.1.6.1 Keine Wertzuweisungen in Conditional Statements

```
'BAD
If (i=2) = 2 then
```

2.3.1.6.2 „:“ Verkettungen von Statements nur wenn es die Lesbarkeit verbessert und wenn es logisch gruppierbare Elemente verkettet.

2.3.1.7 „Veiled declaring“ vermeiden

```
'BAD
Dim iMyVar As Integer

Sub DoSomething
    iMyVar = 2
End Sub

Sub DoSomethingElse
    DoSomething
    If iMyVar > 2 Then
       (…)
    End If
End Sub

'GOOD
Function DoSomething() As Integer
    DoSomething = 2
End Sub

iMyVar = DoSomething
```

2.3.1.8 Benenne Methoden mit “verb-object pair”

```
'EXAMPLE
ShowDialog()
```

2.3.1.9 Benutze nicht den Klassennamen in Property-Deklarationen

```
'BAD
Customer.CustomerName

'GOOD
Customer.Name
```

2.3.1.10 Properties haben niemals „Get“ oder „Set“ im Identifier.

2.3.1.11 Code „defensiv“, d.h. vergebe nur die Rechte, die nötig sind (Code Defensely Policy).

2.3.1.12 Niemals aus Kontrollstrukturen Returns zurückgeben

```
'BAD
Public Function MyFunction() As Boolean
    If true then
       MyFunction = false
    End If
End Function

'GOOD
Public Function MyFunction() As Boolean
    Dim bReturn As Boolean
    bReturn = true

    If true then
       bReturn = false
    End If

    MyFunction = bReturn
End Function
```

2.3.1.13 If then-else Konstrukte zum einfachen setzen von Variablenwerten sollte auf ein if-then Konstrukt mit Default minimiert werden

```
'EXAMPLE
Dim bIsValid As Boolean
bIsValid = false

If SomeStatement Then
    bIsValid = true
End If
```

2.3.1.14 Hänge Qualifier wie „Average“, „Count“, „Sum“, „Min“, „Max“ etc. an Identifier an, wenn sinnvoll

2.3.1.15 Prozeduren mit Rückgabewert sollten einen Prozedurnamen haben, der den Rückgabewert beschreibt

```
'EXAMPLE
GetObjectState()
```

2.3.1.16 Niemals Standardnamen für Elemente benutzen

2.3.2 Kommentare

2.3.2.1 Code soll selbsterklärend sein.

2.3.2.2 “Comment wisely”. Nutze Kommentare nur für nicht-triviale Codestellen

2.3.2.3 Kommentare müssen einer Rechtschreibprüfung standhalten können.

2.3.2.4 Vermeide „Flowerboxing”

```
'BAD
'******************
'*  Some Comment  *
'******************
```

2.3.2.5 Nutze Task-List Keywords als Flags in Kommentaren.
Keywords sind

Keyword	Nutzung	Description	Mit Recipient?
todo	Offene Tasks		Ja
verbose	Debugging-Ausgaben		Nein
help	Fragen	Offene Fragen, Unklarheiten	Ja
change	Änderungen gegenüber der vorgänger-Version		Nein
info	Informationen	Bsw. non-CCP-konformer Code	Nein

2.3.2.6 Benennung von Tasks-Comments sind immer in der Form
`TaskKeyword [Author] ([@Recipient]) („#duedate“+DueDate {yymmdd}) Comment`
wobei Author und der optional Recipient das Namenskürzel ist (bsw. jbl, uit)

```
'EXAMPLE
'todo [jbl] [@uit] #duedate:070918 Some Comment
```

2.3.2.6.1 Todos mit Duedate werden nach Bearbeitung wie folgt geflagt

```
'EXAMPLE
'todo [jbl] [@uit] #duedate:070918, #donedate:070918 New Comment
```

2.3.2.7 Code-Stellen die nicht konform zur CCP sind, müssen mit einem Kommentar versehen werden.

2.3.2.8 Kommentare werden wie die darauf folgende Code-Zeile eingerückt.
Ausnahmen können reine Debugging-Comments sein.

2.3.2.9 Eine Kommentarzeile beginnt mit einem Apostroph, niemals mit `REM`

2.3.3 Code-Formatierung

2.3.3.1 Code-Einrückung (identation) mit Schrittweite 4 Leerzeichen.

2.3.3.2 Einrückung und Leerzeilen im Code sollen Code lesbarer machen (ähnlich wie Paragraphen in Büchern).

2.3.3.3 in einer Zeile Code steht maximal ein Statement. Insbesondere wird pro Zeile in der Regel nur eine Variable deklariert, außer die Lesbarkeit wird spürbar erhöht.

2.3.3.4 Code wird jeweils eine Schrittweite weiter eingerückt zwischen:
`If` und `End`
`If` und `Else`
`If` und `ElseIf`
`Else` und `End`
`ElseIf` und `Else`/ `End`
`Select` und `End Select`
Zwischen allen `Case`-Statements
`Do` und `Loop`
`For` und `Next`
`With` und `End With`
`Edit`/ `AddNew`-Methode und deses `Update`/`CancelUpdate`
Start und Ende einer Transaktion
Bei allen Deklarationen die untergeordnete Elemente hat, bsw. Types mit Typ-Deklarationen.

2.4 Best Practises

2.4.1 Deklarationsreihenfolge ist `Dim`, `Enum`, `Type`, `Property`, `Constructor`, `Destructor`, `Events` (`Form` [`Load`,`Open`,`Close`,`Unload`,weitere], `Element`), `private Function`/ `Sub`, `public Function`/ `Sub`

2.4.2 Schreibe “Structured Code”

2.4.2.1 Nutze Klassen und nutze sie ihrer Bestimmung entsprechend – denke Objekt-Orientiert.

2.4.2.2 Trenne Front- und Backend-Code

2.4.2.3 Gruppiere Funktionen ausserhalb von Classes thematisch sinnvoll in Modulen

2.4.2.4 Nutze Properties

2.4.3 Niemals Strings hardcoden, die sich bsw durch Deployment ändern können. Benutze hierfür Config-Dateien

2.4.4 Resourcen werden so eng wie möglich abgefangen

2.4.4.1 Das Anbieten von public oder protected member Variablen ist zu vermeiden. Stattdessen mit Properties arbeiten.

```
'BAD
Class SomeClass
    public SomeValue as String
End Class

Dim oClass As SomeClass
oClass.SomeValue = “value"

'GOOD
Class SomeClass
    Public Property SomeValue (…)
End Class

Dim oClass As SomeClass
oClass.SomeValue = “value"
```

2.4.4.2 Als „Rule of thumb“ gilt: Methoden sollten nicht mehr als 25 Zeilen haben.

3 T-SQL

3.1 Namenskonventionen und Styles

3.1.1 Benutze Großbuchstaben für alle T-SQL Konstrukte außer Typen
```
SELECT MAX(myField) FROM tbl_myTable
```

3.1.2 Benutze Kleinbuchstaben für alle T-SQL Typen und Usernamen
```
DECLARE @myVariable int
```

3.1.3 Benutze lower Camel-Case für alle UDOs (User defined objects)
```
CREATE TABLE dbo.tbl_myTable
(
    myField int
)
```

3.1.4 Vermeide Abkürzungen und Variablennamen mit nur einem Buchstaben
```
--GOOD
DECLARE @counter int

--BAD
DECLARE @c int
```

3.1.5 UDO-Bennenungen müssen dem Regulären Ausdruck ([a-zA-Z][a-zA-Z0-9]+) entsprechen.
```
--BAD
CREATE TABLE dbo.[Some Field]
```

3.1.6 Benennungssprache ist Standardmäßig Englisch.

3.1.7 Folgende Präfixe werden bei den Benennung von Objekten benutzt
`usp_` - User-created Stored Procedures
`qry_` - Queries and Views
`udf_` - User defined Functions
`tbl_` - Tabellen
`udt_` - User defined Datatypen

3.1.8 Tabellen in der Einzahl benennen
```
--GOOD
CREATE TABLE dbo.tbl_adress

--BAD
CREATE TBALE dbo.tbl_adresses
```

3.1.9 Tabellen werden hierarchisch benannt und einzelne Hierarchieebenen werden mit „_“ getrennt.
```
--EXAMPLE
tbl_customer
tbl_customer_adress
```

3.1.10 Native Felder einer Tabelle haben kein Präfix. SQL-Keywords dürfen nicht zur alleinigen Benennung von Tabellenfeldern benutzt werden. Bei SELECT-Aufrufen von Tabellenfeldern muss immer der Tabellenname angegeben werden. Das gilt auch bei „*“-Aufrufen.
```
--EXAMPLE
```

Tabelle	Native Felder	Referenzierung bei SELECT
tbl_customer	firstname	dbo.tbl_customer.firstname
	dateOfBirth	dbo.tbl_customer.dateOfBirth

3.1.11 Jede Tabelle führt ein eindeutiges Index-Feld namens „id“. Bei Tabellen ohne Replikation kann das ein inkrementeller Autowert sein, ansonsten ein eindeutiger Indexwert.

3.1.12 Zufallszahlen als Indexfelder in replizierten Tabellen sind zu vermeiden.

3.1.13 Fremdschlüssel werden in Tabellen durch das präfix „id_“ gekennzeichnet, gefolgt vom Fremd-Tabellen-Identifier (ohne Objektpräfix)

```
--EXAMPLE
tbl_customer_adress
id
id_customer
...
```

3.1.14 Bei CREATE-Anweisungen auf MSSQL immer das Schema angeben

```
--GOOD
CREATE TABLE dbo.tbl_customer

--BAD
CREATE TABLE tbl_customer
```

3.1.15 Bei einer Objektreferenzierung auf MSSQL immer das Schema mit angeben

```
--GOOD
SELECT * FROM dbo.tbl_customer

--BAD
SELECT * FROM tbl_customer
```

3.1.16 Statements werden immer sauber formatiert. Dabei nutzen wir striktes „multi-line with identation“

```
--EXAMPLE
SELECT
    *
FROM
    dbo.tbl_customer_adress
WHERE
    dbo.tbl_customer_adress.id_customer IN
        (
            SELECT
                *
            FROM
                dbo.tbl_customer
        )
    AND dbo.tbl_customer_adress.someField > 0
```

3.1.17 Einrückung bei lokalen Scopes

```
--EXAMPLE
BEGIN
    (...)
END
```

3.1.18 Wenn Klammerung eingesetzt wird, dann werden die Klammern in eigene Zeilen geschrieben und der eingeschlossene Block eingerückt

```
--GOOD
RETURN
(
    ...
)
```

3.1.19 IF/ ELSE-Statements öffnen immer einen BEGIN-Scope

```
--EXAMPLE
IF (1=2)
BEGIN
    (…)
END
```

3.1.20 CREATE PROCEDURE öffnet immer einen BEGIN-Scope

```
--EXAMPLE
CREATE PROCEDURE dbo.usp_myProcedure
```

```
AS
BEGIN
    (…)
END
```

3.1.21 JOIN immer im FROM-clause, nicht im WHERE clause (ANSI-Syntax)

```
--BAD
SELECT
    ...
FROM
    dbo.tbl_customer,
    dbo.tbl_customer_adress
WHERE
    dbo.tbl_customer.id = dbo.tbl_customer_adress.id_adress
```

3.1.22 Im JOIN-clause steht der referenzierte Schlüsselwert immer rechts

```
--EXAMPLE
...
dbo.tbl_customer
JOIN dbo.tbl_customer_adress ON dbo.tbl_customer_adress.id_customer =
dbo.tbl_customer.id
```

3.1.23 Definiere immer ein „Test Harness“ wenn Prozeduren oder Funktionen definiert werden. Deklariere hierin alle Parameter als Variablen für einfacheres Testen.

```
--EXAMPLE
CREATE PROCEDURE dbo.usp_myFunction
(
    @myParameter int
)
AS
/*
    TEST HARNESS
        DECLARE @myParameter int
        SET @myParameter = 1
        SELECT * FROM dbo.usp_myFunction(@MyParameter)
    /TEST HARNESS
*/
(…)
```

3.1.24 Definiert eine Procedure mehr als nur ein simples INSERT/UPDATE/DELETE, überlege ein erweitertes Test Harness mit einer Roll-Back-Transaktion

```
--EXAMPLE
CREATE PROCEDURE dbo.usp_myFunction
(
    @myParameter int
)
AS
/*
    TEST HARNESS
        DECLARE @myParameter int
        SET @myParameter = 1
        BEGIN TRAN
                SELECT * FROM dbo.tbl_myTable
                EXEC dbo.usp_myFunction(@myParameter)
                SELECT * FROM dbo.tbl_myTable
        ROLLBACK TRAN
    /TEST HARNESS
*/
(…)
```

3.1.25 “Comment wisely”. Nutze Kommentare nur für nicht-triviale Codestellen

3.2 Best Practises

3.2.1 Database Design und Architektur

3.2.1.1 Entscheide zwischen einem guid oder integer als Primary keys. Nutze Guids in Replikations-Szenarios

3.2.1.2 Jede Tabelle besitzt einen Primary Key. Ausnahmen können Tabellen aus den folgenden Kategorien sein
- many-to-many Relationstabelle
- Temporäre Tabelle
- Importtabelle

3.2.1.3 FK Identifier im MSSQL-Server haben maximal 30 Zeichen.

3.2.1.4 Wenn es Enumerations gibt, die auf die Datenbank beschränkt sind, können diese in einer Tabelle abgelegt werden und der Primary Key kann ein anderer sein als eine guid oder ein integer. Wenn der Primary Key ein anderer ist als eine Guid der ein Integer, so wird dieser mit „code“ benannt

```
--EXAMPLE
--GOOD
tbl_language            EXAMPLE data
id (PK)
languageCode            DE, EN, ...
languageDescription

--GOOD
tbl_language            EXAMPLE data
languageCode (PK)       DE, EN, ...
languageDescription
```

3.2.1.5 Entscheide ob ein id-Field oder ein code-Feld in Enumeration-Tabellen als Primary Key benutzt wird. Wenn id und code-Feld vorhanden ist, mixe niemals den Identifier. Insbesondere nutze in einer Datenbank nicht id und code-Feld bsw. in unterschiedlichen UDOs als Parameter

3.2.1.6 Als Collation wird in der Regel ein `SQL_Latin1_General` oder, falls verfügbar, das spezielle `SQL_Latin1_General_CP1_CI_AS` (Case insensitive/ Accent sensitive) benutzt.

3.2.1.7 Informative Tabellenfelder (last access, last update,...) warden an das Ende einer Tabelle gesetzt.

3.2.2 Coding

3.2.2.1 Vermeide „GOTO“ – Anweisungen

3.2.2.2 Vermeide umfangreichere SQL-Statements in allem außer in Views, Procedures und Functions.

3.2.2.3 Benutze CURSOR so selten wie möglich

3.2.2.4 Benutze immer individuelle Feldnamen in INSERT und UPDATE-Statements. Niemals nutze hier „*“.

```
--GOOD
INSERT INTO dbo.tbl_customer (firstname, lastname)
VALUES (@firstName, @lastName)

--BAD
INSERT INTO dbo.tbl_customer VALUES (@firstName, @lastName)
```

3.2.2.5 Setze NOCOUNT als erstes Statement nach BEGIN in allen Procedures in denen nicht explizit der Count zurückgegeben werden soll

3.2.2.6 Verwende immer CONVERT statt CAST.

3.2.3 Queries und Optimierung

3.2.3.1 Vermeide `SELECT *`

3.2.3.2 Bei großen Tabellen sollte die „driving table“ in einer Query (die Tabelle im `FROM`) diejenige Tabelle sein, die prozentual die wenigsten Rows zurückgibt.

3.2.3.3 Komplette Strings werden mit „=“ gefiltert.

3.2.3.4 Benutze CURSOR beim deklarieren ohne „(...)“ im FOR-clause. Benutze zudem standardmäßig „TOP“-Angaben.

```
-- GOOD
DECLARE crs CURSOR
    FOR
        SELECT TOP 100 PERCENT
        …

-- BAD
DECLARE crs CURSOR
    FOR
        (
            SELECT
                …
        )
```

4 Boilerplates

4.1 Error-Handling

```
Public/Private Function/Sub ProcedureName([Parameter]) [As Datatype]
    On Error Goto f_err

       [...]

f_exit:
    [ProcedureName = vReturn]
    Exit Function/Sub

f_err:
    [vReturn = DefaultValue]
    [Handle Error]
    Goto f_exit
End Function/Sub
```

5 History

Date	Author	Changes
July, 2006	J. Blauth	Release CCP Version 0.1
October, 2006	J. Blauth	Release CCP Version 0.2
November, 2006	J. Blauth	Release CCP Version 0.3 (online)
January, 2007	J. Blauth	Release CCP Version 0.4
April, 2007	J. Blauth	Release CCP Version 1.0
May, 2007	J. Blauth	Policy added: 3.2.3.4
May, 2007	J. Blauth	Policy changed: 3.1.10 Policies adapted: 3.2.1.4, 3.2.2.4
June, 2007	J. Blauth	Policy changed: 3.1.7 Policies adapted: 3.1.20, 3.1.23, 3.1.24