Software Implementation

Fehler! Unbekannter Name für Dokument-Eigenschaft.

Programmierrichtlinie für Microsoft Visual Basic.

Key Words: Code, Richtlinie, VB, Visual Basic

Dokument Nummer:	8'000'031'812 (SEM-0057)
Dokument Index:	A
Ausgabe Datum:	14-Mar-2003
Dokument Status:	akzeptiert/freigegeben
Autor:	Andreas Fehr, 6513
Firma:	Siemens Building Technologies AG / Building Automation
Klassifizierung:	Nur für internen Gebrauch
verantwortliche Stelle:	Software Fachgruppe
Dokumentverwaltung:	SEM, Metaphase

Dokument Freigabe(n)

Andreas Fehr	Donat Hutter
(Autor, Fachverantwortlicher)	(Software Process Management)

Änderungen

Rev	Date	Author	Remarks
A	14-Mar-2003	Donat Hutter	Aufnahme ins SEM

V48	`+ Dictionary hinzugefügt`
	`! kleine grammatikalische Änderungen`
	`+ Node (in TreeView) hinzugefügt`		

V47	`+ Abschnitt über 'Probleme beim Errorhandling'`		
V46	`+ Controlname lvi für ListViewItem hinzugefügt` `! Funktionsdokumentation 'Type' in 'Function Type' geändert`		

Table of Contents

1. Einführung, Allgemeines

1.1 Zweck des Dokumentes

Vorgaben zur Erreichung eines einheitlichen Codierungsstils für Microsoft Visual Basic Code.

1.2 Geltungsbereich

Entwicklungsprojekte innerhalb Siemens Building Technologies, Building Automation.

2. Namenskonventionen

Abweichungen von folgenden Konventionen müssen schriftlich (im Code) begründet werden (mit der Deklaration)!

Für die folgenden Kapitel gelten diese Präfixe:

<Prefix>	***Typ***	***Variablen***	***Parameter***	***Konstanten***	***Bemerkungen***
b	Boolean (bit)	x	x	B	(1)
byt	Byte	x	x	BYT	
col	Collection	x	x		(5)
cur	Currency	x	x	CUR	(2)
dic	Dictionary	X	X		(5)
dtm	Date/Time	x	x	DTM	(2)
dbl	Double	x	x	DBL	(3)
enm	Enum		x		(5)
int	Integer	x	x	INT	
lng	Long	x	x	LNG	
obj	Object (7)	x	x		(5)
sng	Single	x	x	SNG	(3)
str	String	x	x	STR	
var	Variant	x	x		(1)(4)(5)
rs	Recordset	x	x		(6)
db	Database	x	x		(6)
ebo	Object (8)	x	x		(5)
stc	Structure	x	(x)		(5)

(1) Abweichung zu Microsoft
(2) kommt in RXT nicht vor
(3) kommt in RXT nur in 'paint' Funktionen vor
(4) darf nur für Arrays verwendet werden
(5) keine Konstante möglich
(6) möglich als read-only
(7) bei Objekten die zur Laufzeit gelinkt werden
(8) bei Objekten die schon zur Entwicklungszeit gelinkt werden (Instanzen von Klassen) (early bound object)

2.1 Variablen

Alle *Variabelnamen* haben folgenden Aufbau:

`<scope><prefix><body>`

`<scope>`

g_	für public Variablen	
	Wo:	Nur in Modulen (*.bas)
	Wie:	Public g_ ...
	Wann:	So wenig wie möglich verwenden!
m_	für private variablen	
	Wo:	In Klassen, Forms, Controlls (*.cls, *.frm, *.ctl)
	Wie:	Private m_ ...
	Wann:	Falls Variable nicht nur lokal verwendet werden soll
-	für lokale Variablen (keine scope Angabe)	
	Wo:	In Funktionen, Subs, Properties
	Wie:	Dim ...
	Wann:	Wenn eine Variable nur lokal verwendet wird

`<prefix>`: (siehe Tabelle oben)

`<body>`

- Erster Buchstabe des Bodies muss gross sein (caps)
- Worte werden durch Grossbuchstaben optisch abgetrennt (auch Abkürzungen werden dadurch klein geschrieben)

2.2 Funktionsparameter

Alle *Parameter* einer Funktion haben folgende Aufbau:

`<pass><scope><prefix><body>`

`<pass>`

ByVal	Alle Parameter, die in der Funktion nicht verändert werden
ByRef	Alle Parameter, die in der Funktion verändert werden und alle Objekte

`<scope>`

p_	für Parameter	
	Wo:	Nur in Funktionen
	Wie:	p_ ...

`<prefix>`: (siehe Tabelle oben)

`<body>`

- Erster Buchstabe des Bodies muss gross sein (caps)
- Worte werden durch Grossbuchstaben optisch abgetrennt (auch Abkürzungen werden dadurch klein geschrieben)

2.3 Konstanten Namen

Alle *Konstanten Namen* haben folgenden Aufbau:

`<scope><prefix><body>`

Allgemein:

- ●im Unterschied zu den Variablen werden die Konstanten gross geschrieben.
- ●es dürfen keine expliziten Konstanten verwendet werden (z.B. 2)
- ●es dürfen nur implizite Konstanten verwendet werden (z.B. Const INT_MAX_KEY = 42: If p_intNewKey > INT_MAX_KEY Then ...)

<scope> (fea: bin nicht mehr sicher, ob ich mir das richtig notiert habe, brauchen wir da einen scope??)

G_ für public Konstante
Wo: Nur in Modulen (*.bas)
Wie: Public Const G_ ...
Wann: So wenig wie möglich verwenden!

M_ für private Konstanten
Wo: In Klassen, Forms, Controlls (*.cls, *.frm, *.ctl)
Wie: Private Const M_ ...
Wann: Falls Konstante nicht nur lokal verwendet werden soll

- für lokale Konstanten (keine scope Angabe)
Wo: In Funktionen, Subs, Properties
Wie: Const ...
Wann: So wenig wie möglich (gibt es da einen Fall?)

<prefix>: (siehe Tabelle oben)

<body>

- ●Alle Buchstaben des Bodies müssen gross sein (caps)
- ●Worte werden durch '_' optisch abgetrennt

2.4 Controllnamen

Alle *Controllnamen* haben folgenden Aufbau:

```
<prefix><body>
```

<prefix>:

<Prefix>	***Typ***
chk	Check box
cbo	Combo box, drop-down list box
cmd	Command button
dlg	Common dialog
ctl	Control (in Funktionen, wenn der Typ unbekannt ist)
dir	Directory list box
drv	Drive list box
fil	File list box
frm	Form
fra	Frame
hsb	Horizontal scroll bar
img	Image
icb	Image combo
ils	ImageList
lbl	Label
lin	Line

lst	List box
lv	ListView
lvi	ListViewItem
mnu	Menu
nde	Node (in Treeview)
opt	Option button
pic	Picture box
clp	Picture clip
prg	ProgressBar
rtf	RichTextBox
shp	Shape
sld	Slider
spn	Spin
sta	StatusBar
tab	TabStrip
txt	Text box
tmr	Timer
tlb	Toolbar
tv	TreeView
vsb	Vertical scroll bar

<body>:

- Erster Buchstabe des Bodies muss gross sein (caps)
- Worte werden durch Grossbuchstaben optisch abgetrennt (auch Abkürzungen werden dadurch klein geschrieben)

2.5 Dateinamen

Alle *Dateinamen* haben folgenden Aufbau:

```
<prefix><body>
```

<prefix>

frm	für Forms
bas	für Module
cls	für Klassen (Classe Module)
ctl	für Controls (User Control)

<body>

- Erster Buchstabe des Bodies muss gross sein (caps)
- Worte werden durch Grossbuchstaben optisch abgetrennt (auch Abkürzungen werden dadurch klein geschrieben)

Die Dateinamen der Dateien haben folgenden Aufbau:

- <body>.<prefix>
- Der <prefix> wird von VB selber vergeben.

2.6 Funktionsnamen

Alle *Funktionsnamen* haben folgenden Aufbau:

```
<object><predicate>
```

Hinweis:

- mit Funktionen sind immer auch Sub, Properties

Hinweis:

- sollten viele Funktionsnamen mit dem gleichen Objekt beginnen, so kann das ein Hinweis sein, dass aus diesem Objekt eine Klasse gemacht werden sollte.

2.7 Enums

Alle *Enums* haben folgende Aufbau:

```
<prefix><body>
```

Achtung: Enum-Deklarationen können die Schreibweise betreffend Gross/Kleinschreibung ändern, VB übernimmt immer die letzte Version aus dem Code.

`<prefix>`

Enum für Enums

`<body>`

- Erster Buchstabe des Bodies muss gross sein (caps)
- Worte werden durch Grossbuchstaben optisch abgetrennt (auch Abkürzungen werden dadurch klein geschrieben)

Hinweis:

- Enums müssen explizit deklariert werden (siehe Beispiel)
- die Werte der Enums müssen mit den Grossbuchstaben aus <prefix> und <body> und einem '_' beginnen (siehe Beispiel)

Beispiel:

```
Enum EnumFuerEtwasCode  ' => EFEC_ als <prefix> der Enumwerte
    EFEC_DieserWert = 1 ' explizit den Wert setzen
    EFEC_SelbigerWert = 2
    EFEC_AndererWert = 3
End Enum
```

3. Dokumentation

Hinweise:

- Kommentar ist prinzipiell in Englisch zu halten.
- Wichtig ist *wieso etwas gemacht wird*, nicht *was* gemacht wird.
- Kommentar darf nicht mit Fortsetzungszeichen ' _' fortgesetzt werden. Jede Zeile muss neu mit ' begonnen werden.

3.1 Fehler

Ein Fehler, der in Code vorgenommen wird, der schon freigegeben wurde, wird wie folgt markiert:

<Kurzzeichen>@yyyymmdd: CR<CR-Nummer>

z.B. fea@20011205: CR12864

Zusätzlich zu diesem Kommentar gehört natürlich noch die Beschreibung des geänderten Codes.

Hinweise:

- Der Kommentar wird gleich eingerückt wie der Code

3.2 Zeilen

Hinweise:

- Der Kommentar wird gleich eingerückt wie der Code
- Kommentar zu Zeilen wird immer oberhalb der Zeilen geschrieben. Ausnahme: 'If-Verschachtelungen' sollten wie folgt dokumentiert werden:

```
' why this if
If (blah) Then
    ' here we do this
    dothis
Else      ' not (blah) here
    ' else we do that
    dothat
End If ' end (blah)
```

3.3 Blöcke

Blöcke ermöglichen eine Strukturierung des Codes, wenn keine Funktionen geschrieben werden. Es muss eine der folgenden Varianten verwendet werden:

```
''''''''''''''''''''''''''''''''''''''''
' Titel des Kommentars
'
' Kurze Beschreibung, was hier gemacht wird
'
```

oder

```
''''''''''''''''''''''''''''''''''''''''
' Titel des Kommentars
' ----------------------------------
' Kurze Beschreibung, was hier gemacht wird
'
```

3.4 Funktionen

Funktionen werden wie folgt dokumentiert:

```
''''''''''''''''''''''''''''''''''''''''
' <Scope><Function Type>Funktionsname
'
' Argument      Type    Range    Description
' --------      ----    -----    -----------
' p_strName     String           Applicationname
' p_intObjCount Integer 1..255   Number of objects in application
```

```
'
' Return Value: Long    0        Failed
'                       1..255   Number of objects used
'
' Description
' Kurze Beschreibung, was hier gemacht wird
'
' History
' -------
' 20011130  Arthur Schweri (sar)
'           Changed to use the NV number from the database, not
'           from the XIF file (there are 'hidden' NVs in the XIF)
'
Private Function AddObjects(ByVal p_StrName As String, ByVal
p_intObjectCount As Integer) As Long
...
```

<Scope>: Private, Friend, Public

<Function Type>: Sub, Function, Property Get, Property Set, Property Let

Hinweis:

- Properties, die nur zwei drei Zeilen Code enthalten können auch etwas einfacher dokumentiert werden.
- Der Range kann zusätzlich noch mit `Debug.Assert` kontrolliert werden.

3.5 Dateiheader

Dateien werden wie folgt dokumentiert:

```
' --------------------------------
' Name:         frmCommSupView
' File:         CommSupView.frm
' --------------------------------
' Copyright 1997-2002, SIEMENS Building Technologies
' --------------------------------
' OS:           Win32
' Lang:         VisualBasic 6.0
' --------------------------------
' Project:      DESIGO RXT
' Responsible   Arthur Schweri (sar)
' --------------------------------
' Description:
'   This form is the commissioning main view, where the user
'   selects groups, datapoints and viewing category
'
' History:
'   19980523    G. Pusch (Gesytec)
'               Original version
'
'   20011206    A. Schweri (sar)
'               modification for new commissioning support V2.1
' --------------------------------
```

```
Options Explicit
```

Der Eintrag `Options Explicit` ist zwingend!

4. Codierung

Die folgende Liste von Tipps sind verbindlich. Wird etwas nicht so gelöst wie vorgeschlagen, dann muss im Sourcecode der Grund angegeben werden.

4.1 Vorschriften

4.1.1 Wiederverwendbarkeit

Code darf nur einmal geschrieben werden. Ist eine Ähnlichkeit mit bestehendem Code vorhanden, so muss der Code zusammengefasst werden. Gegebenenfalls muss der Code in ein neues Modul ausgelagert werden.

Geht dies aus irgendeinem Grund nicht, so muss bei beiden Codestellen eine Referenz auf die jeweils andere Stelle gemacht werden. So wird bei einer Änderung erkannt, dass beide Codestellen aktualisiert werden müssen. Solche Fälle müssen aber mit einem zweiten Entwickler diskutiert werden!

4.1.2 Arbeiten mit Fenster (Forms)

1. Forms müssen wie Variablen deklariert werden.
2. Die globale Form darf nicht verwendet werden.
3. Das hat zur Folge, dass in einer Form keine globalen Variablen verwendet werden dürfen.

Beispiel:

```
(1)Dim frmSpecialForm As frmProperties
(2)Set frmSpecialForm = New frmProperties
(3)frmSpecialForm.SomeParameter = m_eboThisParameter
(4)frmSpecialForm.Show vbModal
(5)(Unload frmSpecialForm)
(6)Set frmSpecialForm = Nothing
```

zu (3): Da auf keine globalen Variablen zugegriffen werden kann, müssen Variablen z.B. als Properties weitergegeben werden. Leider kann dem Konstruktor (Zeile (2), New) kein Parameter übergeben werden.

zu (5): Wenn die Form über Hide verlassen wird, dann muss im aufrufenden Code die Form mit '`Unload frmSpecialForm`' aus dem Speicher entfernt werden.

zu (6): Diese Zuweisung '`Set frmSpecialForm = Nothing`' wird benötigt, da sonst private Variablen in der Form nicht freigegeben werden, falls frmSpecialForm wieder verwendet wird (ohne 'out of scope' zu gehen).

4.1.3 Objekte

Werden verschachtelte Objekte mehr als zweimal verwendet, so müssen diese referenziert werden.

z.B.

```
For intObjIdx = 1 To _
        objTomObject.AppDevice.Interface.LonMarkObjects.Count
    For intCptidx = 1 To _

objTomObject.AppDevice.Interface.LonMarkObjects(intObjIdx).ConfigPro
perties.Count
        strCptVal =
m_objTomObject.AppDevice.Interface.LonMarkObjects(intObjIdx).ConfigP
roperties(intCptIdx).Value
        If strCptVal = "bla" Then
            strCptVal = "blo"

m_objTomObject.AppDevice.Interface.LonMarkObjects(intObjIdx).ConfigP
roperties(intCptIdx).Value = strCptVal
        End If
    Next intCptIdx
Next intObjIdx
```

besser so:

```
Set objLmObjects = objTomObject.AppDevice.Interface.LonMarkObjects
For intObjIdx = 1 To objLmObjects.Count
    Set objCpts = objLmObjects(intObjIdx).ConfigProperties
    For intCptIdx = 1 To objCpts.Count
        strCptVal = objCpts(intCptIdx).Value
        If strCptVal = "bla" Then
            strCptVal = "blo"
            objCpts(intCptIdx).Value = strCptVal
        End If
    Next intCptIdx
Next intObjCount
```

Jedes Auflösen über ein verschachteltes Objekt dauert ein kleine Ewigkeit.

4.1.4 For Index ... Next Index

`For`-Schlaufen müssen mit `Next` + Index abgeschlossen werden. VB erlaubt zwar das weglassen des Index bei `Next`, dies ist aber verboten.

4.1.5 Symmetrischer Code

Codeblöcke wie '`if`', '`do`', '`while`' , '`loop`', '`case`' etc. müssen mit der Bedingung kommentiert werden, wenn jene mehr als 5 Zeilen lang sind.

z.B.

```
If CheckIfValidString(strValue) Then
...
... 'mehr als 5 Zeilen
...
End If ' CheckIfValidString(strValue)
```

4.1.6 Funktionsnamen

Funktionsnamen sind so zu wählen, dass Funktionen zu ähnlichen Aufgaben in der DropDownListe beeinanderliegen (Objekte vor den Verben):

```
DeviceBindingsInstall()
DeviceSettingsInstall()
DeviceSettingsUpload()
DeviceApplicationLoad()
DeviceApplicationVersionCheck()
```

Dies entspricht auch ein wenig dem Klassengedanke.

4.1.7 Events

Werden in einem Event mehr als 5 Zeilen Code geschrieben müssen diese in einer separaten Funktion ausgelagert werden.

4.1.8 Zeilenumbrüche

Zeilen müssen logisch umbrochen werden.

z.B.

```
If ThisIsSomeDummyFunction() And ThisIsSomeOtherFunction _
   And bValidValue Then
    ' do something
End If
```

oder

```
bOrValue = IsValidParameter(strCurrentValue, m_strDefaultvalue) _
           Or IsActiveObject(m_objSelectedObject)
```

4.2 Empfehlungen

4.2.1 Klassen Entwicklung

In neu entworfenen Klassen den Aufruf 'Debug.Print “Terminate KlassenName“' in die Terminate Funktion einfügen. So kann einfach geprüft werden, wann die Klasse aus dem Speicher geladen und nicht mehr referenziert wird.

4.2.2 Kreisreferenzen

Sollte VB einmal nicht mehr vom Run-Modus in den Design-Modus wechseln, dann liegt sicher eine 'Kreisreferenz' vor. Mehrere Objekte sind im Kreis aufeinander angewiesen. Keines wird freigegeben, die Software läuft noch.

A -> B -> C -> A

z.B. A creates B, B.Parent = A, B creates C, C.Parameter = A. A löscht B, B wird nicht freigegeben, da C noch verwendet wird, da C nicht freigegeben wird, weil in C noch A verwendet wird.

Abhilfe kann der Tipp (Klassen Entwicklung) schaffen.

4.2.3 Funktionslängen

Funktionen sollten nicht länger als 80 Zeilen sein (= eine Bildschirmhöhe oder A4 beim Ausdruck).

4.2.4 Parameter

Funktionen sollten nicht mehr als 5 Parameter haben.

4.2.5 Debug.Print

Strategische Debug.Print Anweisungen können mit Debug_Print ausgeführt werden. Mit dieser Funktion ist es möglich auch zur Laufzeit den Programmfluss zu beobachten.

4.2.6 Verschachtelungen

Die Verschachtelungstiefe in Funktionen sollte < 5 sein. (If, Case, etc...)

4.2.7 Zeilenlänge

Zeilen sollten nicht länger als 80 Zeichen sein.

4.3 Dont's

4.3.1 SourceSave

```
Debug.Assert False
```

Darf nicht engechecked werden.

Bei COM-Komponenten darf die Kompatibilität zu alten Versionen nur bis zum Meilenstein F2 gebrochen werden.

4.3.2 Funktionen die nie aufgerufen werden dürfen:

```
chdir
```

Hilfefunktionen laufen danach nicht mehr, wenn der Pfad nicht mehr stimmt.

```
exit
```

Dies beendet eine Funktion. Debugging und Codeänderungen werden dadurch unnötig erschwert. Ausnahme: Errorhandling, siehe dort.

```
end
```

Das Programm wird dadurch abgebrochen. Keine weitere Funktion wird aufgerufen.

```
Debug.Assert Function()
Debug.Assert Property
```

In der Anweisung `Debug.Assert` darf *keine Funktion* ausgeführt werden.

```
goto
```

Darf nur in Errorhandlern verwendet werden (siehe dort).

4.3.3 Schlechter Code

- Arrays anstelle von Klassen verwenden (Verdacht bei Zugriff auf fixe Indizes im Array) (vor allem bei Gesytec ein Problem).
- Collections anstelle von Klassen verwenden (Verdacht bei vielen Zugriffen über Namen)
- return values als Variablen verwenden:

```
Private Function blah() as Integer
    blah = 3
    blah = 2 * blah
End Function
```

4.3.4 Strukturen

Es dürfen keine Strukturen verwendet werden. Diese müssen als Objekte implementiert werden. (Ausnahme: Win32 API Aufrufe).

4.3.5 Pragma #if

Das Progmakonstrukt von VB #if, #else, #end if darf nicht verwendet werden.

4.4 Errorhandling

Das Fehlerhandling in VB wird über `On Error Goto` gelöst. Vor einem Funktionsblock wird das Errorhandling eingeschaltet. Am Ende der Funktion wird ein ErrorLabel eingefügt. In der gleichen Funktion können mehrere Labels vorhanden sein.

Beispiel:

```
Private Function Blah(Byval p_intCounter As Integer) As Long
    On Error Goto BlahErrorDivide
Dim intNumber As Integer, intResult As Integer
    intResult = p_intCounter / intNumber

    On Error Goto BlahError2
    Blah = intResult * G_LNG_CONST
BlahExit:
    Exit Function
BlahError1:
    Select Case Err
        Case 11
            ' division by zero, intNumber was wrong...
            intNumber = 1
            Resume
        Case Else
            ' Unknown Error occured, just display the error text
            MsgBox Error$, VbInformation, "AppName"
    End Select
    Resume BlahExit
BlahError2:
    Select Case Err
        Case 11
            ' division by zero not possible, we multiply
            Resume BlahExit
        Case Else
            ' Unknown Error occured, just display the error text
            MsgBox Error$, VbInformation, "AppName"
    End Select
    Resume BlahExit
    ' just for debugging purpose, if the code is checked in,
    ' we can jump to where the error occured
    Resume
End Function
```

4.4.1 Problem beim Errorhandling

- Verschachtelte Errorhandler müssen über Funktionen gelöst werden.
- Es darf nie ein if/while/loop etc. einen Error auslösen, da das weiterfahren mit 'Resume Next' nicht klar ist. Im Moment wird zwar die nächste Codezeile ausgeführt (bei 'Resume Next') aber

dies könnte mit einer nächsten Version von VB ändern. Also immer zuerst das Statement in einem Boolean speichern und auf diesen testen.

5. Entwicklungsumgebung

●Als Entwicklungsumgebung gilt VB6, SP5, Englisch

Einstellungen die bei allen Entwicklungsumgebungen gleich sein müssen (Tools, Options).

Bemerkungen:

[ ] Diese Checkbox ist nicht ausgewählt

[x] Diese Checkbox ist ausgewählt

5.1 Vorschriften

Diese Optionen müssen so eingestellt werden, damit Fehler vermieden werden.

5.1.1 Editor

●[x] Require Variable Declaration

●[x] Auto Indent

●Tab Width: 4

5.1.2 General

●Grid Width: 60

●Grid Height: 60

●[x] Align Controls to Grid

●(x) Error Trapping, Break in Class Module

●[ ] Compile On Demand

5.1.3 Environment

●(x) Save Changes

5.2 Empfehlungen

Jeder so wie er will, diese haben sich bewährt.

5.2.1 Editor

●[ ] Auto Syntax Check

●[ ] Auto List Member

●[x] Default to Full Module View

5.2.2 General

●[x] Show Grid

●[ ] Collapse Proj. Hides Windows

5.2.3 Advanced

●[x] SDI Development Environment

6. Userinterface Richtlinien

Allgemein sollte darauf geachtet werden, dass Userinterfaces grosszügig designed werden. Lieber einen Dialog mehr, als zwei, die nicht übersichtlich sind.

Achtung, deutsch Texte sind normalerweise länger als englische. Es muss darauf geachtet werden, das die Grösse von Controls auch für Beschriftungen in andere Sprachen ausreicht.

6.1 Controls

Es dürfen nur folgende Controls verwendet werden:

●Microsoft Common Dialog Control 6.0 (SP3)
Standarddialoge wie File Open, Save, Print etc.

●Microsoft Windows Common Controls 6.0 (SP4)
Elemente wie Listview, Treeview, Tabs, Image List etc

Andere gemäss Abmachungen in einem Projekt (z.B. LNS, SpinText etc.)

6.2 Layout

6.2.1 Allgemein

●Controls werden 60 Einheiten vom Rand entfernt platziert (ein Gridpunkt ist noch sichtbar)

●Dialoge mit BorderStyle = 2 - Sizable müssen die Grösse aller Controls zur Laufzeit neu berechnen. Normalerweise werden die Controls am unteren und rechten Rand verschoben, die Controls oben und links bleiben, die in der Mitte werden vergrössert.

6.2.2 Buttons

●Buttons haben eine Höhe von 315 twips

●sind mehrere Buttons auf einer Form, so sind diese zueinander auszurichten

●sind mehrere logisch verknüpfte Buttons auf einer Form, so haben diese die gleiche Breite (Save, Load oder OK, Cancel etc.)

●Buttons müssen gruppiert werden