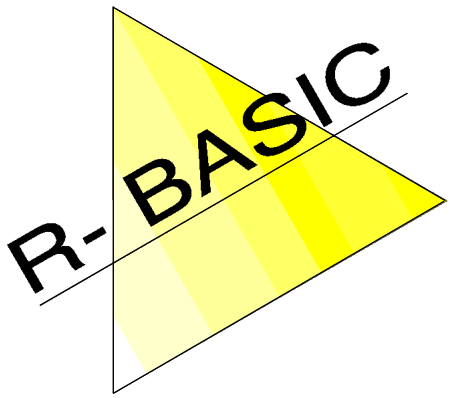
R-BASIC

Einfach unter PC/GEOS programmieren



Objekt-Handbuch

Volume 2
Die GenericClass

Version 1.0

(Leerseite)

(Leerseite)

3 Die Generic Class

Die meisten Objekte in R-BASIC stammen von der GenericClass ab. In diesem Abschnitt finden Sie die Instance-Variablen, die allen diesen Objekten gemeinsam sind.

3.1 Caption: Die Objekt-Beschriftung

Unter "Caption" (engl. caption = Überschrift, Titel) versteht man in R-BASIC die "Beschriftung" eines Objekts. Das kann ein Text oder eine kleine (!) Grafik sein. Im Primary befindet er sich der Caption-Text in der Titelzeile, bei Buttons ist es die Aufschrift und bei anderen Objekten ist er über oder neben dem Objekt angeordnet um die Funktion oder Bedeutung des Objekts zu beschreiben. Text-Captions weisen Sie mit der Instancevariablen **Caption$** zu. Für grafische Captions stehen Ihnen - je nachdem, wo die Grafik herkommt, die Methoden **CaptionIcon**, **CaptionPicture**, **CaptionImage** und **CaptionGString** zur Verfügung. Mit der Instancevariablen **justifyCaption** können Sie in vielen Fällen festlegen, wie die Caption relativ zum Objekt positioniert wird.

Variable	Syntax im UI-Code	Im BASIC-Code
Caption$	Caption$ = **"Text"** [, **n**]	lesen, schreiben
CaptionIcon	CaptionIcon = **"tchr"** , **manufID** [, **flags**]	nur schreiben
CaptionPicture	CaptionPicture = **"PictureName"**	nur schreiben
CaptionImage	CaptionImage =[**stdPath**,] "**File**" [, **num**]	nur schreiben
CaptionGS	—	nur schreiben
justifyCaption	justifyCaption = **numWert**	lesen, schreiben
kbdShortcut	kbdShortcut = **numWert**	lesen, schreiben
kbdSearchPath	kbdSearchPath = TRUE I FALSE	lesen, schreiben

Bitte beachten Sie, dass Captions im gleichen Speicherblock gespeichert werden, wie das Objekt selbst. Speicherblöcke können unter GEOS nicht größer als 64 kByte werden, meistens gibt es schon viel früher Probleme ("Hauptspeicher voll"). Grafische Captions sollten deshalb nicht größer als 4 kByte sein. Bei Text-Captions (Caption$) Captions aus der TokenDatabase (CaptionIcon) und GString-Captions (CaptionGString) ist das im Allgemeinen erfüllt. Normale Grafikbefehle wie Line, Rectangle, FillEllipse usw. erfordern jeweils 10 bis 15 Byte. Texte erfordern pro Zeichen 1 Byte.

Problematisch können Captions sein, die eine Bitmap enthalten und mit zur Laufzeit zugewiesen werden, da dies der Compiler nicht prüfen kann. Bei der Zuweisung im UI-Code führt R-BASIC bei Bitmaps eine Größenkontrolle aus und warnt bei einem Speicherbedarf von mehr als 4 kByte. Captions mit mehr als 12 kByte lassen sich nicht zuweisen.

Der Speicherbedarf einer Bitmap setzt sich aus den Bitmapdaten und einer eventuell vorhandenen Maske (Transparenz) zusammen. Für die Bitmapdaten gilt

die Formel "Breite x Höhe x Farbtiefe (in Bit pro Pixel) / 8". Für die Maske kommen je Zeile noch "Breite/8" Bytes hinzu, wobei jeweils auf ganze Bytes aufgerundet (!) werden muss.

Beispiele (jeweils eine Transparenzmaske vorausgesetzt)

Abmessungen (Pixel)	Farbtiefe (Bit pro Pixel)	Speicherbedarf (Byte)
48 x 30	4	900
32 x 32	8	1152
32 x 32	24	3200
64 x 64	8	4608
128 x 128	8	51200

Falls Sie vorhaben, zur Laufzeit grafische Captions zuzuweisen, die deutlich größer sind als die zur Compilezeit zugewiesenen, sollten Sie der Verteilung der Objekte auf die Objektblöcke Aufmerksamkeit widmen. Details dazu finden Sie im Kapitel 2.1.4 (Beeinflussung der Objektblöcke im UI-Code).

3.1.1 Text-Captions

Caption$

Caption$ ist der Text auf oder neben dem Objekt. Im Gegensatz zu den grafischen Captions kann Caption$ auch gelesen werden und es kann ein "Navigationsbuchstabe" definiert werden, der eine Tastaturnavigation durch die Menüs ermöglicht.

Syntax UI-Code: **Caption$** = **"Text"** [, **n**]
"Text" : Aufschrift
n: Nummer des hervorgehobenen Buchstaben für Tastatur-Navigation
0 = 1. Buchstabe, 1 = zweiter Buchstabe usw.

Lesen: **<stringVar>** = **<obj>**. **Caption$**
Liefert den Text. Der Navigationsbuchstabe kann nicht gelesen werden.

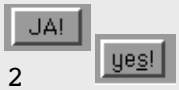
Schreiben: **<obj>**. **Caption$** = **"Text"** [, **n**]

Beispiele UI Code:

```
Button  OKButton
  Caption$ = " OK "
  ActionHandler = ....
  END Object

Button  MyButton
  Caption$ = "Press mich!", 0
  ActionHandler = ....
  END Object
```

Beispiel BASIC-Code

```
DIM c$
  c$ = MyButton.Caption$
  OKButton.Caption$ = " JA! "
  MyButton.Caption$ = "yes!", 2
```



3.1.2 Grafische Captions

CaptionIcon

Weist dem Objekt ein Token aus der Token-Database als Caption zu. CaptionIcon kann im UI Code und im BASIC Code (Schreiben) verwendet werden. Lesen im BASIC-Code ist nicht möglich.

Syntax UI Code:	**CaptionIcon = "tchr" , manufID [, flags]**
Syntax BASIC:	**<obj>.CaptionIcon = "tchr" , manufID [, flags]**
"tchr":	Tokenchars des Icons. Genau 4 Zeichen
manufID:	ManufacturerID des Icons. Datentyp WORD
flags:	Icon-Flags. Siehe unten.

Das passende Bild aus der Tokendatabase Datei wird beim Aufruf von CaptionIcon in den Speicherblock des Objekts kopiert.

Gültige Werte für "flags":

Konstante	Wert	Bedeutung
TOOL_ICON	1	Tool-Icon (15 x 15 Pixel) verwenden.
TINY_ICON	1	Synonym für TOOL_ICON
SMALL_ICON	2	Kleineres Icon verwenden (oft 32x20 Pixel)
BIG_ICON	4	Größeres Icon verwenden (oft 64x40 Pixel)
GRAY_ICON	8	Schwarz-Weiß Icon verwenden
RGB_ICON	16	True-Color Icon verwenden

Wird keines der Flags angegeben wird das "Standard" Icon (meist 48 x 30 Pixel, 16 Farben oder 256 Farben) verwendet.

Hinweise:

- Ist die entsprechend den Flagbits angeforderte Kombination nicht vorhanden sucht das System ein "möglichst passendes" Icon aus. Das Flag "TOOL_ICON" hat dabei Vorrang vor allen anderen Flags.
- Sollte zum gegebenen Token ("TCHR", manufID) kein grafisches Icon vorhanden sein wird ein Text verwendet.
- Findet sich das Token nicht in der TokenDB zeigt R-BASIC ein Ersatzbild ("unbekanntes Icon").
- R-BASIC Icons enthalten nur zwei Bilder: ein Standard- und ein Tool-Icon.

Beispiel: Das GeoWrite-Datei-Icon "WDAT, 0" enthält Normal, Small und Big Icons

```
Button Button1
  CaptionIcon = "WDAT", 0, TOOL_ICON
End OBJECT

Button Button2
  CaptionIcon = "WDAT", 0, SMALL_ICON
End OBJECT

Button Button3
  CaptionIcon = "WDAT", 0
End OBJECT

Button Button4
  CaptionIcon = "WDAT", 0, BIG_ICON
End OBJECT
```

CaptionPicture

CaptionPicture weist einem Objekt eine grafische "Aufschrift" zu. Die Grafik steht in der Picture-List des Programms (oder der Library).
Die Picture-List enthält Grafiken, die über ihren Namen angesprochen werden und in der Code-Datei selbst gespeichert sind. Sie kann über das Menü "Extras" -> "Picture-List" verwaltet werden. Details dazu finden Sie im Kapitel 2.8.6.2 (Verwendung der Picture-List) des R-BASIC Programmierhandbuchs.

Syntax UI Code: **CaptionPicture = "PictureName"**
Syntax BASIC: **<obj>.CaptionPicture = "PictureName"**
"PictureName": Name der Grafik in der Picture-List

Das Bild wird beim Aufruf von CaptionPicture in den Speicherblock des Objekts kopiert. Beachten Sie den Hinweis am Anfang des Kapitels 3.1. Zu große Caption-Bilder können zum Systemabsturz führen!

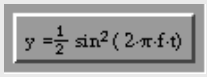
Hinweise:
- Wird CaptionPicture im BASIC-Code gerufen setzt es die globale Variable fileError - entweder auf Null (das Bild wurde gefunden) oder auf einen Fehlerwert (das Bild wurde nicht gefunden).
- Wenn CaptionPicture im Code einer Library gerufen wird bezieht sich der Name des Bildes auf die Picture-List der Library. Das ermöglicht es unter anderem Bilder in die Picture-List von Libraries auszulagern.

Beispiel: In der Picture-List befinden sich zwei kleine Bilder mit dem Namen "Radioactive" und "Formel". Die Grafik "Formel" wurde mit GeoDraw erstellt und dann über die Zwischenablage in die Picture-List aufgenommen.

```
Button MyButton
  CaptionPicture = "Radioactive"
End OBJECT
```

Die Zuweisung der zweiten Grafik kann z.B. im Basic Code erfolgen.

```
MyButton.CaptionPicture = "Formel"
```



CaptionImage

CaptionImage weist einem Objekt eine grafische "Aufschrift" zu die aus einer externen Datei gelesen wird. Sollte die Datei mehr als ein Bild enthalten (z.B. *.GIF, *.ICO) können Sie mit dem Parameter pictNum bestimmen, welches Bild ausgelesen wird. Das erste Bild hat immer die Nummer Null.

Syntax UI Code:	**CaptionImage =** [**stdPath**,] **"Path+File"** [, **pictNum**]	
Syntax BASIC:	**<obj>.CaptionImage** = [**stdPath**,] "**Path+File**" [, **pictNum**]	
	stdPath:	Optional: Standardpfad Konstante, z.B. SP_TOP
	"Path+File":	Dateiname, Pfade sind zulässig
	pictNum:	Optional: Nummer des Bildes in der Datei

CaptionImage öffnet und schließt die Datei automatisch. Das Bild wird in den Speicherblock des Objekts kopiert. Beachten Sie den Hinweis am Anfang des Kapitels 3.1. Zu große Caption-Bilder können zum Systemabsturz führen!

Folgende Dateiformate werden unterstützt: JPG, BMP, ICO, PCX, GIF, TGA, RLE, DIB, SCR , FLC, FLI und GEOS Hintergrunddateien.

Wird CaptionImage im UI-Code verwendet so gilt:
- Wird kein Standardpfad angegeben wird die Datei im Ordner "USERDATA\R-BASIC\IMAGES" gesucht.
- Das Bild wird in die Code-Datei kopiert, d.h. die externe Datei muss nicht in das R-App Paket aufgenommen werden.

Wird CaptionImage im BASIC-Code verwendet so gilt:
- Wird kein Standardpfad angegeben wird die Datei im aktuellen Verzeichnis gesucht.
- Die externe Datei wird zur Laufzeit geöffnet, d.h. sie **muss unbedingt** in das R-App Paket aufgenommen werden oder es muss auf andere Weise sichergestellt sein, dass sie existiert.
- Wird pictNum nicht angegeben so wird immer das erste Bild ausgelesen.
- Die globale Variable fileError wird gesetzt - entweder auf Null (das Bild wurde gefunden) oder auf einen Fehlerwert (die Datei wurde nicht gefunden oder sie enthält kein Bild).

Beispiel: Das PC/GEOS Icon "GWICON5.ICO" verwenden, dass sich im PC/GEOS Hauptverzeichnis befindet.

```
Button MyButton
  CaptionImage = SP_TOP, "GWICON5.ICO"
End OBJECT
```

CaptionGString

CaptionGString weist dem Objekt eine grafische Caption zu, die vorher in einen GString geschrieben wurde. Die Verwendung von CaptionGString ist eine der Möglichkeiten für ein Objekt (häufig ein Button oder eine Group) zur Laufzeit die grafische Caption zu ändern. CaptionGString wird verwendet, wenn die Grafik nicht als vordefiniertes Bild vorliegt, sondern zur Laufzeit des Programms gezeichnet werden muss (oder kann). Eine komplette Beschreibung der Arbeit mit GStrings finden Sie im Kapitel 2.8.5 (Arbeit mit Graphic Strings) des R-BASIC Programmierhandbuchs.

Syntax BASIC: **<obj>.CaptionGString = <gsHandle>**
<gsHandle>: Handle auf einen Graphic String.

Der GString wird beim Aufruf von CaptionGString in den Speicherblock des Objekts kopiert. Daher sollten Sie nicht vergessen den GString nach Verwendung wieder freizugeben. Beachten Sie den Hinweis am Anfang des Kapitels 3.1. Zu große Caption-Bilder können zum Systemabsturz führen!

Beispiel:

```
SUB SetNewCaption ( )
DIM gsHan AS HANDLE

  gsHan = StartRecordGS ( )
  FillEllipse 0, 0, 32, 32, LIGHT_RED      ' Farbig
  Ellipse 0, 0, 32, 32                     ' Schwarz
  FillRect 9, 8, 24, 24, YELLOW
  EndRecordGS gsHan

  MyObj.CaptionGString = gsHan

  FreeGS gsHan                             ' Nicht vergessen!
End SUB
```

3.1.3 Die Caption-Ausrichtung

justifyCaption

Die Instance-Variable **justifyCaption** bestimmt, wo bzw. wie die Beschriftung (Caption~) des Objekts angeordnet wird. Die folgende Tabelle enthält die zulässigen Werte. Häufig verwendet wird die Kombination J_TOP + J_CENTER, bei Buttons auch J_CENTER alleine.

Konstante	Wert	Bedeutung
J_CENTER	1	Caption zentrieren
J_LEFT	2	Caption nach links
J_RIGHT	4	Caption nach rechts
J_TOP	8	Caption nach oben

Wird **justifyCaption** nicht gesetzt, ist es dem Objekt überlassen, wo die Beschriftung angeordnet wird. In vielen Fällen entspricht dies J_LEFT. Einige Objekte akzeptieren nicht jeden Wert, da justifyCaption als Hint implementiert ist; z.B. ignorieren Buttons alles außer J_CENTER. Hier hilft nur ausprobieren.
Beachten Sie, dass nur die in der Tabelle oben aufgeführten Konstanten für justifyCaption akzeptiert werden. Andere J_-Konstanten, die z.B. für justifyChildren (siehe Kapitel Geometriemanagement) definiert sind, werden ignoriert.

Beispiel

```
View       MyView
  Caption$ = "Vorschau"
  justifyCaption = J_TOP + J_CENTER    ' Mittig über dem
  Objekt
  < .. weitere instances .. >
  END Object
```

3.1.4 Keyboard Shortcuts

Keyboard Shortcuts sind Tastenkürzel, die auch dann wirken, wenn das entsprechende Menü nicht offen ist. Typische Fälle sind "Strg-C" für Kopieren und "Strg-P" für Drucken. Um einen Shortcut zu definieren müssen Sie außer dem ASCII-Code der Taste auch die "Modifier"-Taste (Strg, Shift oder Alt) angeben, die gedrückt sein soll. Der ASCII-Code kann auch ein "erweiterter" ASCII-Code sein, z.B. für die Cursortasten oder F12.

Obwohl die Instancevariable **kbdShortcut** für alle GenericClass Objekte definiert ist wird sie hautsächlich für Buttons benutzt. Die im Folgenden für Buttons getroffenen Aussagen sind sinngemäß auf alle andere GenericClass Objekte übertragbar. Bitte beachten Sie bei der Wahl der Tastenkombination für kbdShortcut, dass diese vom "R-BASIC Translator" nicht geändert werden kann.

kbdShortcut

Die Instancevariable kbdShortcut enthält einen WORD-Wert, der ein Tastenkürzel beschreibt. Drückt der Nutzer die entsprechende Tastenkombination (z.B. Strg + Z) wird der Button aktiviert, genau so, als sei er direkt angeklickt worden.

Syntax UI-Code: **kbdShortcut** = **numVal**
Lesen: **<numVar>** = **<obj>**. **kbdShortcut**
Schreiben: **<obj>**. **kbdShortcut** = **numVal**

Die niederwertigen 8 Bit (Bit 0 bis Bit 7) enthalten den ASCII-Code der Taste bzw. den Steuercode der Steuertaste. Die Bits 8 bis 11 sind gesetzt, wenn es sich um eine Steuer- oder Funktionstaste handelt, ansonsten sind sie Null. Die vier höchstwertigen Bits (Bit 12 bis Bit 15) enthalten "Modifier"-Tasten, die gleichzeitig gedrückt sein müssen, damit das Kürzel aktiviert wird. Dafür sind die folgenden Konstanten definiert (KSM = Keyboard Shortcut Modifier):

Konstante	Wert	hex.	Bedeutung
KSM_SHIFT	4096	&h1000	Shift Taste muss gedrückt sein
KSM_CTRL	8192	&h2000	Strg Taste muss gedrückt sein
KSM_ALT	16384	&h4000	Alt Taste muss gedrückt sein
KSM_PHYSICAL	32768	&h8000	Die Taste ist gemeint, nicht das Zeichen. Das heißt im Wesentlichen, dass der Shift-Lock- und der NumLock-Status ignoriert werden.

Hinweise:
- Den ASCII-Code der gewünschten Taste können Sie über die Funktion ASC( ) erhalten, die auch in numerischen Ausdrücken erlaubt ist.
- Tipp für Fortgeschrittene: KSM_PHYSICAL bedeutet auch, dass der Scancode der Taste ausgewertet wird. Für den seltenen Fall, das Sie ein solches Objekt

mit einem simulierten Tastaturereignis aktivieren wollen, müssen Sie die Methode KbdEventWithScancode (anstelle von KbdEvent) verwenden. Details dazu finden Sie im Handbuch Themen, Kapitel 14.3 (Simulieren von Tastaturereignissen).

Um die gewünschte Tasten-Kombination für den Shortcut zu definieren ist manchmal etwas Experimentieren erforderlich. Die folgenden Beispiele demonstrieren die typischen Fälle.

Beispiel: Ausschneiden (Ctrl-X) und Kopieren (Ctrl-C).
Die Verwendung von KSM_PYHSICAL stellt sicher, dass die Kürzel auch funktionieren, wenn die Shift-Lock Taste eingerastet ist. Beachten Sie, dass die Codes der Kleinbuchstaben angegeben werden.

```
Button CutButton
  Caption$ = "Ausschneiden", 0
  ActionHandler = DoCut ' ButtonAction
  kbdShortcut = KSM_CTRL + KSM_PHYSICAL + ASC("x")
End OBJECT

Button CopyButton
  Caption$ = "Kopieren" , 0
  ActionHandler = DoCopy ' ButtonAction
  kbdShortcut = KSM_CTRL + KSM_PHYSICAL + ASC("c")
End OBJECT
```

Beispiel: Umsch Ctrl A
Um die Umschalttaste in einen Shortcut aufzunehmen können Sie entweder den ASCII-Code eines Großbuchstaben angeben oder einen Kleinbuchstaben mit dem Flag KSM_SHIFT kombinieren. In beiden Fällen müssen sie zusätzlich das Flag KSM_PHYSICAL setzen, weil der Tastaturtreiber die Shift-Taste bereits beim Erzeugen des ASCII-Codes verarbeitet.

```
  kbdShortcut = KSM_CTRL + KSM_PHYSICAL + ASC("A")
  kbdShortcut = KSM_CTRL + KSM_SHIFT + KSM_PHYSICAL  +
  ASC("a")
```

Beispiel F12
Wenn Sie eine Steuertaste oder eine F-Taste als Shortcut setzen wollen benötigen Sie den entsprechenden "erweiterten" ASCII-Code. Diese Codes sind in der Library "KeyCodes" definiert. Suchen Sie im Wizzard der Library unter "GetKey: Steuertasten" den passenden Code heraus. Sie müssen aber unbedingt beachten, dass die Codes alle 8 höherwertigen Bits gesetzt haben. Deswegen müssen Sie die 4 höchstwertigen Bit mit der Operation "code AND &hFFF" ausblenden.

In der Library "KeyCodes" ist folgendes definiert:

```
  CONST KEY_F12 = &hFF8B
```

UI Code

```
Include "KeyCodes"

Button MyButton
  < ... >
  kbdShortcut = KEY_F12 AND &hFFF
End OBJECT
```

Alternativ können Sie auch direkt den Code, aber ohne die höherwertigen 4 Bit, angeben:

```
  kbdShortcut = &hF8B
```

Um Strg-F12 als Shortcut zu setzen verwenden Sie eine der folgenden Zeilen. Die Klammern sind wichtig!

```
  kbdShortcut = (KEY_F12 AND &hFFF) + KSM_CTRL
bzw.
  kbdShortcut = &hF8B + KSM_CTRL
```

Beispiel: ESC

Ein häufiger Fall ist das Verwenden der ESC-Taste für "Abbrechen" oder "Beenden". Die ESC-Taste ist eine erweiterte Taste und hat den Code &hFF1B. Um die ESC-Taste einem Button als Tastenkürzel zuzuweisen verwenden Sie eine der folgenden Zeilen:

```
  kbdShortcut = KEY_ESC AND &hFFF
bzw.
  kbdShortcut = &hF1B
```

kbdSearchPath

Damit die Keyboard Shortcuts arbeiten können müssen die entsprechenden Objekte im "Suchpfad" für Keyboard Shortcuts sein. Primaries, Menüs und Button sind per Default im Suchpfad. Aus Effizienzgründen ist das für die meisten anderen Objekte nicht der Fall. Wenn Ihre Keyboard Shortcuts nicht arbeiten müssen Sie das Objekt, sein Parent, dessen Parent usw. in den Suchpfad aufnehmen indem Sie die Instancevariable **kbdSearchPath** auf TRUE setzen.

Syntax	UI-Code:	**kbdSearchPath** = TRUE I FALSE
	Lesen:	**<numVar>** = **<obj>**. **kbdSearchPath**
	Schreiben:	**<obj>**. **kbdSearchPath** = TRUE I FALSE

3.2 Objekt States

Variable	Syntax im UI-Code	Im BASIC-Code
visible	visible = TRUE I FALSE	lesen, schreiben
fullyVisible	—	nur lesen
enabled	enabled = TRUE I FALSE	lesen, schreiben
fullyEnabled	—	nur lesen
readOnly	readOnly = TRUE I FALSE	lesen, schreiben

Methode	Aufgabe
HideDelayed	visible = FALSE mit verzögertem Bildschirmupdate
ShowDelayed	visible = TRUE mit verzögertem Bildschirmupdate

visible, fullyVisible,

Die Instance-Variable **visible** (engl.: sichtbar) bestimmt, ob das Objekt und seine Children auf dem Schirm erscheinen oder nicht. Wenn Sie ein Objekt auf nicht sichtbar (visible = FALSE) setzen, so wird es einschließlich seiner Children vom Schirm verschwinden. Das bedeutet im Umkehrschluss, dass ein Objekt verborgen sein kann, auch wenn es auf visible gesetzt ist. Um wirklich sichtbar zu sein, muss ein vollständiger Pfad von sichtbaren (visible = TRUE) Objekten bis zum (ebenfalls sichtbaren) Application-Objekt führen. Dieser Zustand heißt **fullyVisible** (= vollständig sichtbar) und kann im BASIC-Code abgefragt werden.
Jedes Objekt, das nicht explizit auf visible = FALSE gesetzt ist, ist per Default visible.

Syntax	UI-Code:	**visible** = TRUE I FALSE
	Lesen:	**<numVar>** = **<obj>**. **visible**
	Schreiben:	**<obj>**.**visible** = TRUE I FALSE
Syntax	Lesen:	**<numVar>** = **<obj>**. **fullyVisible**

Beispiele: siehe unten.

HideDelayed, ShowDelayed

Ändert man zur Laufzeit den visible-Status eines Objekts, so verschwindet das Objekt sofort bzw. erscheint sofort. Das kann zu unerwünschtem Flackern des Bildschirms führen, z.B. wenn Sie mehrere Änderungen vornehmen oder ein Objekt nur kurzzeitig auf visible = FALSE setzen wollen, um bestimmte Änderungen vorzunehmen. HideDelayed setzt das Objekt auf visible = FLASE, ohne es sofort vom Schirm zu nehmen. ShowDelayed setzt entsprechend das Objekt auf visible = TRUE, ohne es sofort neu zu zeichnen.

In jedem Fall wird das Objekt spätestens dann neu dargestellt (oder vom Schirm genommen), wenn der Action-Handler, in dem die ~Delayed-Methode aufgerufen wurde, beendet ist.
Sie können HideDelayed und ShowDelayed beliebig mit dem direkten Belegen der Instancevariable visible mischen. Die häufigste Variante ist das Aufrufen von HideDelayed, danach z.B. das Hinzufügen oder entfernen von Children (bis hier passiert auf dem Bildschirm nichts) und abschließend setzen von visible = TRUE. Erst jetzt stellen sich die Objekte in der neuen Konfiguration neu dar.

Syntax:	**<obj>**. **HideDelayed**
	<obj>. **ShowDelayed**

enabled, fullyEnabled

Ein Objekt ist **enabled** (engl.: aktiviert), wenn der Nutzer damit interagieren kann. Objekte, die nicht enabled sind, zeichnen sich üblicherweise in grau. Wenn Sie ein Objekt auf nicht enabled (enabled = FALSE) setzen, so werden auch alle seiner Children nicht enabled erscheinen, auch wenn ihre eigene Instance-Variable enabled auf TRUE steht. Um wirklich enabled zu sein, muss ein vollständiger Pfad von enabled Objekten bis zum (ebenfalls enabled) Application-Objekt führen. Dieser Zustand heißt **fullyEnabled** (= vollständig enabled) und kann im BASIC-Code abgefragt werden.
Jedes Objekt, das nicht explizit auf enabled = FALSE gesetzt ist, ist per Default enabled.

Syntax	UI-Code:	**enabled** = TRUE I FALSE
	Lesen:	**<numVar>** = **<obj>**. **enabled**
	Schreiben:	**<obj>**.**enabled** = TRUE I FALSE
Syntax	Lesen:	**<numVar>** = **<obj>**. **fullyEnabled**

Beispiele: siehe unten.

readOnly

Ein readOnly (engl.: nur lesen) Objekt ignoriert Texteingaben vom Nutzer. Das ist nur für Objekte, die prinzipiell Texteingaben entgegennehmen können, von Bedeutung. Andere Objekte, wie Buttons, ignorieren den readOnly Wert. Das heißt konkret, dass das Setzten einer Group oder eines ähnlichen Objekts auf readOnly **nicht** dazu führt, dass deren Children (z.B. Texte) readOnly werden.
ReadOnly Objekte ignorieren nur Eingaben vom Nutzer, es ist daher trotzdem möglich alle Instance-Werte eines readOnly-Objekts vom BASIC-Code aus zu ändern.

Syntax	UI-Code:	**readOnly** = TRUE I FALSE
	Lesen:	**<numVar>** = **<obj>**. **readOnly**
	Schreiben:	**<obj>**.**readOnly** = TRUE I FALSE

Beispiele

Der UI-Code definiert eine Gruppe, die ein readOnly Text Objekt und ein auf "nicht enabled" gesetztes Value Objekt enthält.

```
Group   MyInputGroup
  Children = MyText, MyValue
  < .. weitere instances.. >
  End Object

Memo    MyText
  readOnly = TRUE
  text$ = "Hallo Welt!"
  < .. weitere instances.. >
  End Object

Value  MyValue
  enabled = FALSE
  < .. weitere instances.. >
  End Object
```

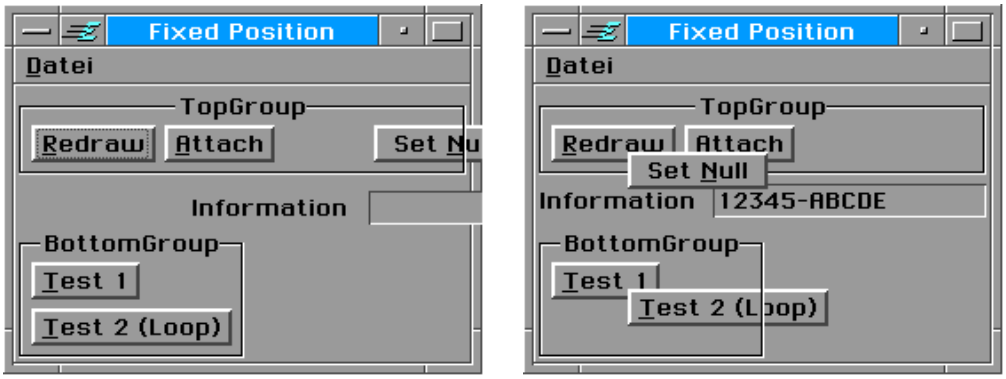
Der folgende BASIC-Code ändert einige Dinge. Es wird vorausgesetzt, dass die Buttons, deren ActionHandler hier implementiert werden, irgendwo definiert sind.

```
ButtonAction  VersteckeGroup
  MyGroup.visible = FALSE
  End Action

ButtonAction  MacheTextEditierbar
  MyText.readOnly = FALSE
  END Action

ButtonAction  SchalteValueEnabeldStateUm
  If    MyValue.enabled = FALSE   THEN
    MyValue.enabled = TRUE
  ELSE
    MyValue.enabled = FALSE
  END IF
END Action

ButtonAction  SchreibeEtwasText
  IF    MyText.readOnly   THEN
    MyText.text$ = "Der Text ist read only."
  ELSE
    MyText.text$ = "Sie können den Text ändern."
  END IF
  END Action
```

3.3 Geometriemanagement

3.3.1 Überblick

Die vernünftige Positionierung der UI-Objekte auf dem Schirm ist eine der aufwändigsten Aufgaben in einer grafischen Programmierumgebung. Der Einsteiger wird annehmen, dass es zweckmäßig sei, für jedes Objekt die Position und Größe explizit anzugeben. Auch wenn dies unter GEOS möglich ist, so ist es für generische Objekte jedoch eine sehr schlechte Idee. Neben dem hohen damit verbundenen Aufwand kommt es schnell zu unschönen Resultaten, wenn der User eine andere Bildschirmauflösung, einen anderen Textfont oder Größe für den Standard-Menü-Text in der GEOS.INI eingestellt hat. Die folgenden Bilder zeigen einige schlechte Beispiele.

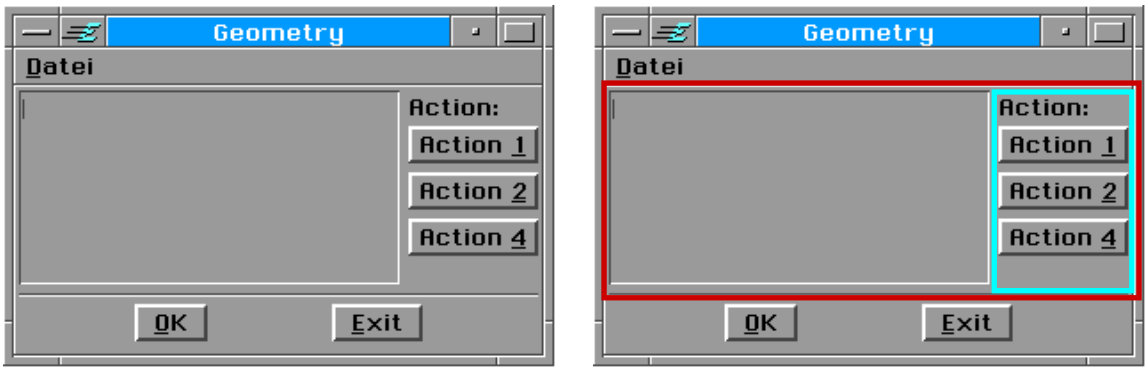


Im linken Bild haben der Text und der Button rechts oben eine feste Position, das Primary-Objekt wurde jedoch mit der Maus "zu klein" eingestellt. Das rechte Bild zeigt manuell deutlich fehlplatzierte Objekte.

Diese Bilder demonstrieren die Wirkung einer festen Größe in Pixel für die oberen Buttons. Im rechten Bild wurde eine vergrößerte Systemschrift verwendet.

Die Lösung für diese Probleme heißt GEOS-Geometrie-Manager. Der Geometrie-manager berechnet automatisch die Größe und Position der Objekte, so dass sie vollständig und vernünftig auf dem Schirm angeordnet werden. Das passiert zur Laufzeit des Programms, so dass unterschiedliche Textfonts und Größen sowie andere Randbedingungen automatisch berücksichtigt werden. Anstatt festzulegen, wo sich die Objekte befinden sollen und wie groß sie sind, geben Sie dem Geometriemanager nur Hilfen der Form "Ordne die Objekte nebeneinander an", "Mache das Objekt so breit wie möglich", "Zentriere die Objekte horizontal" usw. Dieses Konzept erfordert anfangs etwas Eingewöhnung, aber Sie werden schnell

merken, dass es sehr systematisch und intuitiv zu benutzen ist. Sehr hilfreich ist es dabei, die Objekte in Gruppen anzuordnen und die Positionierung der Objekte innerhalb dieser Gruppen auszuführen.



Diese Anwendung beispielsweise besteht aus einer Group (rot, oben) und einer Replybar (unten), die vertikal (untereinander) angeordnet sind. Die rot markierte Gruppe besteht aus einem Text-Objekt und einer weiteren Group, hier blau markiert. Beide Objekte sind horizontal angeordnet. Die rechte (blaue) Group enthält letztlich die Action-Buttons, die untereinander angeordnet sind. Sollte Sie z.B. die Aufschrift des zweiten Buttons verändern, so dass er breiter wird, dann wird der Geometriemanager automatisch alle betroffenen Objekte (beide Group's, das Hauptfenster und die ReplyBar) verbreitern. Eine explizite Größenangabe ist da nur störend.

Anweisungen an den Geometriemanager sind als "Hints" - das heißt "Hilfen" - für den Geometriemanager organisiert. Das bedeutet, Sie zwingen den Geometriemanager nicht, etwas zu tun, sondern Sie bitten ihn. Der Geometrie-manager wird versuchen, Ihre Anweisungen so gut wie möglich umzusetzen. Manchmal kann ihm das jedoch nicht gelingen. Beispielsweise könnten Sie widersprüchliche Forderungen aufgestellt haben. Das ist einer der häufigsten Fehler bei diesem Konzept. Es kann aber auch sein, dass bestimmte Objekte einige Eigenschaften oder Fähigkeiten einfach nicht unterstützen. So ist es bei Buttons nicht möglich, die Aufschrift (Caption$) oberhalb des Buttons anzuordnen, die meisten anderen Objekte unterstützen dies aber. Oder Primary-Objekte (die Hauptfenster eines Programms) ignorieren die Vorgabe einer festen Größe mit "fixedSize", aber sie akzeptieren "WindowSizeFromParent". Gelegentlich kann es daher etwas mit Probieren verbunden sein, den Geometriemanager zu überreden, das zu tun, was man will.

Die Organisation der Geometrie-Eigenschaften als "Hints" hat noch eine andere Konsequenz. Jede Eigenschaft, die Sie vorgeben, kostet einige (wenige) Bytes. Eigenschaften, die nicht explizit angegeben werden, sind auch nicht im Objekt gespeichert. Der Geometriemanager nimmt dann den für das Objekt gültigen Vorgabewert. So ordnen Groups ihre Children untereinander linksbündig an, wenn kein Wert für "orientChildren" und "justifyChildren" vorgegeben wird. Intern ist das so organisiert, dass den eigentlichen Objekt-Daten (Instance-Daten) eine Tabelle variabler Länger folgt, in die die Geometrie-Instance-Daten (Geometrie-Hints) abgelegt sind.

Wird z.B. kein Eintrag für "fixedSize" vorgenommen, so sind die entsprechenden Daten einfach nicht da, das Objekt kommt gar nicht auf die Idee, sich eine feste Größe zu leisten.
Bei vielen der Geometrie-Instance-Daten kann man auch abfragen, ob die Daten präsent sind oder nicht.

3.3.2 Angabe von Größen, Positionen und Abständen

Größen, Positionen oder Abstände werden häufig in Pixeln angegeben. Manchmal ist das aber gar nicht sinnvoll, es ist z.B. oft viel besser zu sagen "Mache den Text 5 Textzeilen hoch". Um so etwas zu kennzeichnen wird zum eigentlichen Zahlenwert (hier 5) eine große Konstante addiert, so dass GEOS weiß, das nicht Pixel sondern Textzeilen gemeint sind. Dabei stehen die folgenden Werte zur Verfügung:

Konstante	Wert	Werte gemessen in:
ST_PIXELS	0	Pixel
ST_AVG_CHAR_WIDTH	4096	mittlere Zeichenbreite
ST_MAX_CHAR_WIDTH	5120	maximale Zeichenbreite
ST_LINES_OF_TEXT	6144	Textzeilen Höhe
ST_PCT_OF_SCREEN_WIDTH	2048	Prozent der Bildschirmbreite
ST_PCT_OF_SCREEN_HEIGHT	3072	Prozent der Bildschirmhöhe
	1024	(Reserviert, nicht benutzen!)

Grundsätzlich gilt: Größenangaben können im Bereich von 0 bis 1023 (jeweils einschließlich) liegen. Ausnahme bilden die _PCT_ (engl. percent: Prozent) - Konstanten. Hier liegen die Werte sinnvollerweise von 0 bis 100.

ST_PIXELS
Dies ist der Standard. Da der Wert Null ist, muss er nicht addiert werden. **Achtung!** Sie können mit ST_PIXELS nur Größen bis maximal 1023 Pixel spezifizieren, da der Wert 1024 vom System reserviert ist!

ST_AVG_CHAR_WIDTH
Der Wert wird mit der mittleren Breite eines Text-Zeichens multipliziert. GEOS verwendet üblicherweise Proportionalfonts, in dem jedes Zeichen eine andere Breite hat. Die erzeugte Größe hängt also vom verwendeten Font ab.

ST_MAX_CHAR_WIDTH
Der Wert wird mit der Breite des breitesten Text-Zeichens multipliziert. Die erzeugte Größe hängt also hängt vom verwendeten Font ab.

ST_LINES_OF_TEXT
Der Wert wird mit der Höhe einer einzelnen Text-Zeile multipliziert. Die erzeugte Größe hängt also von der verwendeten Textgröße ab.

Anmerkung: Die Bezeichnung der Konstante wurde geändert. Die alte Bezeichnung ST_TEXT_LINES kann weiterhin verwendet werden.

ST_PCT_OF_SCREEN_WIDTH
Größe entspricht einem Prozentsatz der Bildschirmbreite. Das ist z.B. nützlich für Dialogboxen.

ST_PCT_OF_SCREEN_HEIGHT
Größe entspricht einem Prozentsatz der Bildschirmhöhe. Das ist z.B. nützlich für Dialogboxen.

Beispiel: feste Größe für ein Textobjekt vorgeben

```
Memo    MyText
  fixedSize = 45 + ST_AVG_CHAR_WIDTH,  8 + ST_LINES_OF_TEXT
  <.. weitere Instance-Variablen.. >
END object
```

Hinweis: Für Fenster-Objekte (Primary, Dialog, Display) gibt es spezielle Hints zum Festlegen der Größe, die sogenannten Window-Hints. Sie sind im Kapitel 3.3.7 beschrieben.

3.3.3 Anordnung der Objekte

Die Anweisungen zum Anordnen der Objekte sind die wohl am häufigsten genutzten Anweisungen im Geometriemanagement. Dabei wird dem Parent-Objekt mitgeteilt, wie es seine Children zu organisieren hat. Sie können z.B. festlegen ob die Children neben oder untereinander angeordnet werden und wie sie ausgerichtet werden sollen (linksbündig, rechtsbündig, zentriert usw.). Die folgenden Hints stehen zur Verfügung:

Hint	Syntax im UI-Code	Im BASIC-Code
orientChildren	orientChildren = **numWert**	lesen, schreiben
justifyChildren	justifyChildren = **numWert**	lesen, schreiben
childSpacing	childSpacing = **numWert**	lesen, schreiben
MinimzeChildSpacing	MinimzeChildSpacing	——
IncludeEndsInChildSpacing	IncludeEndsInChildSpacing	——
wrapAfterChild	wrapAfterChild = **numWert**	lesen, schreiben
DivideHeightEqually	DivideHeightEqually	——
DivideWidthEqually	DivideWidthEqually	——

3.3.3.1 Orientierung und Ausrichtung

orientChildren

OrientChildren legt fest, ob die Children-Objekte nebeneinander oder untereinander angeordnet werden.

Syntax	UI- Code:	**orientChildren** = **numWert**
	Lesen:	**<numVar>** = **<obj>** . **orientChildren**
	Schreiben:	**<obj>**.**orientChildren** = **numWert**

Dabei stehen folgende Konstanten zur Verfügung. Sie können jeweils nur eine Konstante angeben, eine Kombination (mit +) ist nicht zulässig.

Konstante	Wert	Bedeutung
ORIENT_HORIZONTALLY	1	nebeneinander
ORIENT_VERTICALLY	2	untereinander
ORIENT_SAME_AS_PARENT	4	genauso, wie das Parent-Objekt der Children
ORIENT_ON_LARGER_DIMENSION	8	nebeneinander, wenn der Bildschirm breiter als hoch ist, sonst untereinander

Beispiele: siehe unten

justifyChildren

JustifyChildren legt fest, wie die Children-Objekt relativ zueinander angeordnet werden.

Syntax	UI- Code:	**justifyChildren** = **numWert**
	Lesen:	**<numVar>** = **<obj>** . **justifyChildren**
	Schreiben:	**<obj>.justifyChildren** = **numWert**

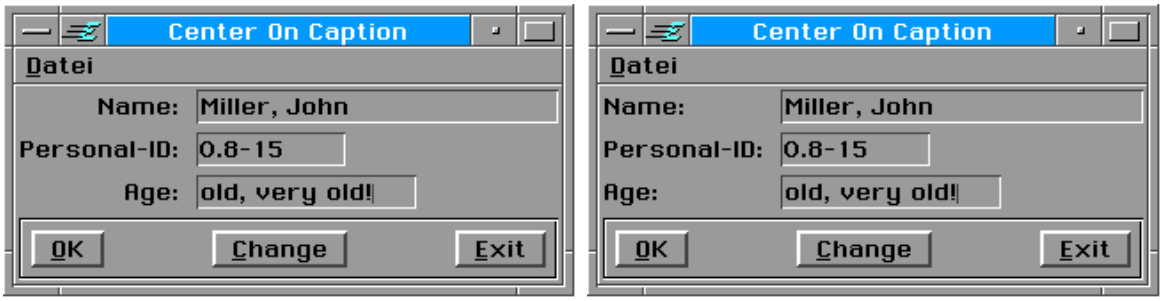
Dabei stehen folgende Konstanten zur Verfügung. Sie können mehrere Konstanten kombinieren (mit +), solange dies sinnvoll ist. Ungültige Werte werden ignoriert, bei widersprüchlichen Werten entscheidet der Geometriemanager.

Konstante	Wert	Bedeutung
J_LEFT	2	linksbündig
J_RIGHT	4	rechtsbündig
J_TOP	8	am oberen Rand
J_BOTTOM	16	am unteren Rand
J_FULL_H	64	volle Breite, horizontal
J_FULL_V	128	volle Höhe, vertikal
J_CENTER_H	256	horizontal zentriert
J_CENTER_V	512	vertikal zentriert
J_CENTER_ON_CAPTION	1024	am Caption$ ausgerichtet
J_CENTER_ON_CAPTION_LEFT	2048	am Caption$ ausgerichtet
J_CENTER	1	wie J_CENTER_H und J_CENTER_V gleichzeitig
J_FULL	32	wie J_FULL_H und J_FULL_V gleichzeitig

Bemerkungen zu einigen der Konstanten:

J_FULL_H, J_FULL_V

Die Children sollen die volle Breite/Höhe des Parent-Objekts ausschöpfen und den freien Platz gleichmäßig zwischen sich aufteilen. Ein Beispiel finden Sie bei den drei unteren Buttons im nächsten Bild. Das ist analog zum Blocksatz bei einer Textverarbeitung. Dazu muss das Parent-Objekt aber größer sein, als die Children erfordern. Oft müssen Sie dazu die Hints **ExpandWidth** bzw. **ExpandHeight** (siehe nächster Abschnitt) setzen.



J_CENTER_ON_CAPTION, J_CENTER_ON_CAPTION_LEFT

Die Texte in den beiden Bildern oben sind "ON_CAPTION" ausgerichtet. Der UI-Code für die obere Group im linken Bild ist:

```
Group TopGroup
  Children = Text1, Text2, Text3
  orientChildren = ORIENT_VERTICALLY
  justifyChildren = J_CENTER_ON_CAPTION
END Object
```

Im rechten Bild dagegen ist:

```
  justifyChildren = J_CENTER_ON_CAPTION_LEFT
```

J_CENTER, J_FULL

Diese Werte werden von R-BASIC intern in die beiden _H und _V Werte umgerechnet, d.h. statt 1 wird 256+512 und statt 32 wird 64+128 in den Objekt-Daten gespeichert. Das ist wichtig, falls Sie den Wert für justifyChildren auslesen und auf bestimmte Konstanten testen wollen.

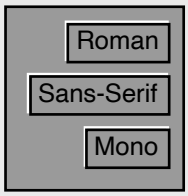
Beispiele für ausgerichtete Objekte

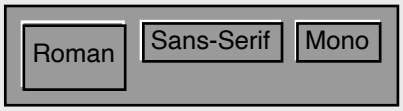
Wir verwenden folgende Buttons:

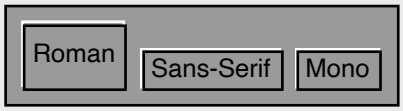
```
Button RomanButton
  Caption$ = "Roman"
  END Object
Button SansButton
  Caption$ = "Sans-Serif"
  END Object
Button MonoButton
  Caption$ = "Mono"
  END Object
```

Gruppen zur Ausrichtung der Objekte

```
Group   Group1
  Children = RomanButton, SansButton, MonoButton
  orientChildren = ORIENT_VERTICALLY
  justifyChildren = J_LEFT
  END Object

Group   Group2
  Children = RomanButton, SansButton, MonoButton
  orientChildren = ORIENT_VERTICALLY
  justifyChildren = J_RIGHT
  END Object


Group   Group3
  Children = RomanButton, SansButton, MonoButton
  orientChildren = ORIENT_HORIZONTALLY
  justifyChildren = J_TOP
  END Object

Group   Group4
  Children = RomanButton, SansButton, MonoButton
  orientChildren = ORIENT_HORIZONTALLY
  justifyChildren = J_BOTTOM
  END Object
```

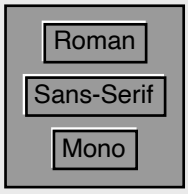


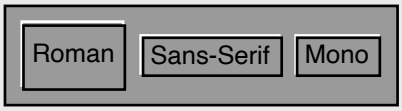


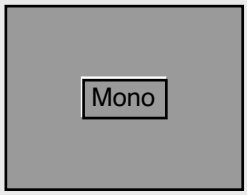


Beispiel zur Objekt-Zentrierung

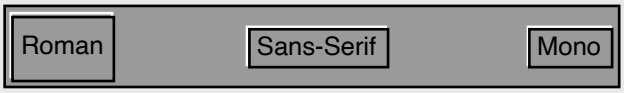
```
Group   Group1
  Children = RomanButton, SansButton, MonoButton
  orientChildren = ORIENT_VERTICALLY
  justifyChildren = J_CENTER_H
  END Object

Group   Group2
  Children = RomanButton, SansButton, MonoButton
  orientChildren = ORIENT_HORIZONTALLY
  justifyChildren = J_CENTER_V
  END Object

Group   Group3
  Children = MonoButton
  justifyChildren = J_CENTER_H + J_CENTER_V
  END Object
```







```
Group   Group4
  Children = RomanButton, SansButton, MonoButton
  orientChildren = ORIENT_HORIZONTALLY
  justifyChildren = J_CENTER_V + J_FULL_H
        ' H und V beachten!
  END Object
```



3.3.3.2 Child Spacing

Normalerweise legt der Geometriemanager die Abstände zwischen benachbarten Objekten fest. Sie können durch das Setzen bestimmter Geometrie-Hints auf diesen Prozess Einfluss nehmen.

childSpacing

ChildSpacing (engl. space: Platz, Abstand) legt einen Wert für den Abstand zwischen benachbarten Children fest. Üblicherweise wird der Wert in Pixeln angegeben, Sie können aber auch die Werte ST_AVG_CHAR_WIDTH, ST_MAX_CHAR_WIDTH bzw. ST_LINES_OF_TEXT verwenden (Details dazu siehe Kapitel 3.3.2). Per Default ist kein childSpacing Wert gesetzt.

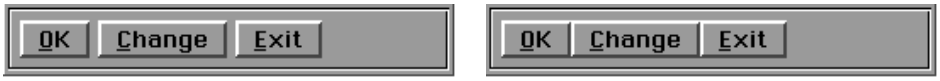
Syntax	UI- Code:	**childSpacing** = **numWert**
	Lesen:	**<numVar>** = **<obj>** . **childSpacing**
	Schreiben:	**<obj>**.**childSpacing** = **numWert**
		Spezialfall: numWert = –1: Löschen eines vorher gesetzten **childSpacing** Hints aus den Objekt-Daten

MinimizeChildSpacing

MinimizeChildSpacing weist den Geometriemanager an, die Children so eng wie möglich anzuordnen, auch wenn das bedeutet, dass sie sich berühren.

Syntax	UI- Code:	**MinimizeChildSpacing**

Beispiel



Eine Group mit 3 Buttons. Links normal, Rechts mit **MinimizeChildSpacing**.

IncludeEndsInChildSpacing

IncludeEndsInChildSpacing bewirkt, dass der Platz links und rechts der Children bei der Positionsberechnung berücksichtigt wird, das linke Child also nicht ganz an den Rand gesetzt wird.

Syntax UI- Code: **IncludeEndsInChildSpacing**

Beispiel
Verschiedene Varianten von childSpacing-Werten

```
Group Group1
  Children = OKButton, ChangeButton, ExitButton
  orientChildren = ORIENT_HORIZONTALLY
  ExpandWidth
  DrawInBox
  childSpacing = 4 + ST_AVG_CHAR_WIDTH
  END Object

Group Group2
  Children = OKButton, ChangeButton, ExitButton
  orientChildren = ORIENT_HORIZONTALLY
  ExpandWidth
  DrawInBox
  childSpacing = 12  ' Pixels
  IncludeEndsInChildSpacing
  END Object

Group Group3
  Children = OKButton, ChangeButton, ExitButton
  orientChildren = ORIENT_HORIZONTALLY
  ExpandWidth
  DrawInBox
  justifyChildren = J_FULL_H
  IncludeEndsInChildSpacing
  END Object
```

3.3.3.3 Automatischer Umbruch

wrapAfterChild

Standardmäßig ordnen alle Groups, deren Children nebeneinander angeordnet sind (dazu zählen auch Reply Bars), alle Children in einer Reihe an. Der Hint **wrapAfterChild** erlaubt es, die Children in mehreren Reihen anzuordnen. Damit dies ordentlich funktioniert ist es oft nötig, der Group und ggf. ihren Parents auch den Hint **ExpandWidth** zu geben.

Syntax	UI- Code:	**wrapAfterChild** = **numWert**
	Lesen:	**<numVar>** = **<obj>**. **wrapAfterChild**
	Schreiben:	**<obj>**.**wrapAfterChild** = **numWert**
		numWert = – 1: Automatischer Modus
		numWert = 0: Hint aus Objektdaten löschen
		numWert = 1 .. N : Umbrechen nach N Children

Automatischer Modus:
Es werden so viele Children nebeneinander angeordnet, wie Platz ist. Gegebenenfalls werden weitere Reihen eröffnet.

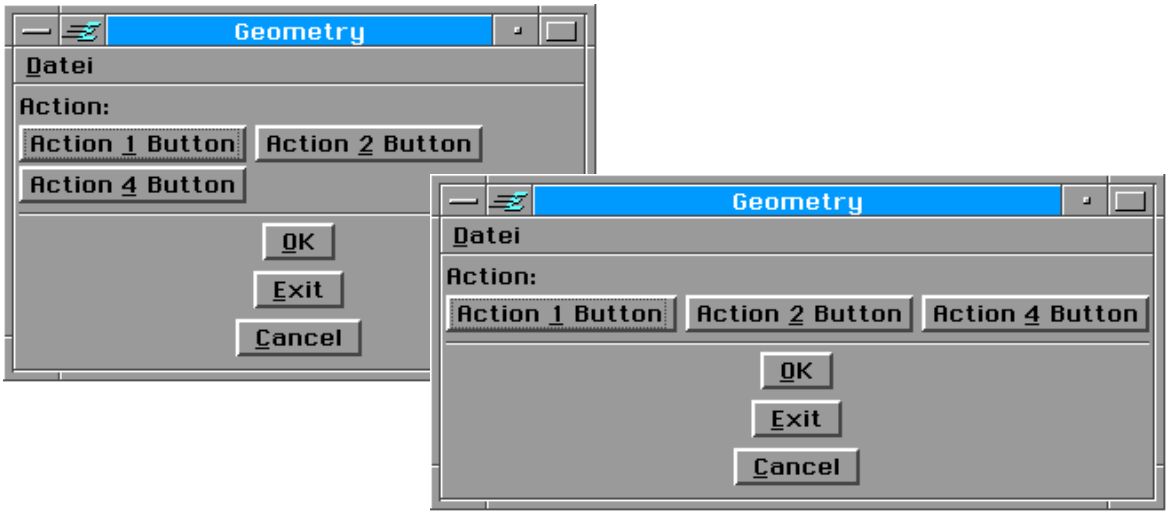
Umbrechen nach N Children:
Achtung! Verhält sich für normale Groups und für ReplyBars unterschiedlich!
Normale Groups: Der erste Umbruch erfolgt frühestens nach N Children. Ist ausreichend Platz erfolgt der erste Umbruch nach Bedarf. Die weiteren Umbrüche folgen nach Bedarf.
Reply Bars: Der Umbruch erfolgt immer nach N children. Auch die nächsten Umbrüche erfolgen nach jeweils N children.

Hinweis: Der Hint ist nicht auf Primaries, DisplayGroups und Displays anwendbar. Verwenden Sie ggf. eine Group als Zwischenobjekt.

Beispiel UI- Code:

```
Group AGroup
  Caption$ = "Action:"
  justifyCaption = J_TOP
  Children =  button1, button2, button3
  orientChildren = orient_horizontally
  wrapAfterChild = –1
  ExpandWidth
  END Object

Group AReplyBar
  MakeReplyBar
  Children = button4, button5, button6
  wrapAfterChild = 1
  END Object
```

Nehmen wir an, die Groups befinden sich in einem Primary-Objekt. Zoomt man dieses erhält man die folgenden Bilder:



3.3.3.4 Vorhandenen Platz gleichmäßig verteilen

DivideWidthEqually

Dieser Hint legt fest, dass sich die Children die vorhandene Breite gleichmäßig untereinander aufteilen sollen. Damit das funktioniert, müssen **alle Children** den Hint **ExpandWidth** gesetzt haben. Auch das Objekt selber sollte **ExpandWidth** gesetzt haben.

Syntax UI- Code:	**DivideWidthEqually**

DivideHeightEqually

Dieser Hint legt fest, dass sich die Children die Höhe gleichmäßig untereinander aufteilen sollen. Damit das funktioniert, müssen **alle Children** den Hint **ExpandHeight** gesetzt haben. Auch das Objekt selber sollte **ExpandHeight** gesetzt haben.

Syntax UI- Code:	**DivideHeightEqually**

Beispiel:

Die Group befinde sich in einem größenveränderlichen Primary-Objekt. Im rechten Bild wurde nur die Breite des Primary Objekts mit der Maus verändert. Hier ist der UI-Code dazu:

```
Group BottomGroup
  Children = OKButton, ChangeButton, ExitButton
  orientChildren = ORIENT_HORIZONTALLY
  DivideWidthEqually
  ExpandWidth
  DrawInBox
  END Object

Button OKButton
  Caption$ = " OK ", 1
  justifyCaption = J_CENTER
  ExpandWidth
  END Object

Button ChangeButton
  Caption$ = " Change ", 1
  justifyCaption = J_CENTER
  ExpandWidth
  END Object

Button ExitButton
  Caption$ = " Exit ", 1
  justifyCaption = J_CENTER
  ExpandWidth
  END Object
```

3.3.4 Objektgröße

Das Vorschreiben einer bestimmten Objektgröße kann sehr oft hilfreich sein, um das Aussehen Ihres Programms den Erfordernissen anpassen. Gerade bei Listen, Text-Objekten und View's kann eine Größenvorgabe erforderlich sein. Bedenken Sie aber, dass der Nutzer Ihres Programms eine andere Bildschirmauflösung oder eine andere Größe für die Systemtexte eingestellt haben kann. Die Verwendung von Größenangaben, die das berücksichtigen (z.B. ST_LINES_OF_TEXT, ST_AVG_CHAR_WIDTH, siehe Kapitel 3.3.2) ist daher oft eine gute Idee. Beachten Sie, dass es für Fenster-Objekte, wie z.B. Primaries, weitere Geometrie-Hints gibt, die "Window-Hints" wie z.B.

SizeWindowAsDesired

Grundsätzlich gilt, dass Sie die Größe eines generischen Objekts nur dann vorschreiben sollten, wenn es unbedingt erforderlich ist. Dabei stehen Ihnen die folgenden Möglichkeiten zur Verfügung.

Hint	Syntax im UI-Code	Im BASIC-Code
ExpandWidth	ExpandWidth	—
ExpandHeight	ExpandHeight	—
NoWiderThanChildren	NoWiderThanChildren	—
NoHigherThanChildren	NoHigherThanChildren	—
initialSize	initialSize = **x**, **y** [, **count**]	lesen, schreiben
minimumSize	minimumSize = **x**, **y** [, **count**]	lesen, schreiben
maximumSize	maximumSize = **x**, **y** [, **count**]	lesen, schreiben
fixedSize	fixedSize = **x**, **y** [, **count**]	lesen, schreiben
xSize	—	nur lesen
ySize	—	nur lesen

ExpandWidth, ExpandHeight

Diese Hints bestimmen, dass das Objekt seine Breite (ExpandWidth) bzw. seine Höhe (ExpandHeight) so groß machen soll, dass der gesamte vom Parent zur Verfügung gestellte Platz ausgenutzt wird. Oftmals hat auch das Parent den entsprechenden Hints gesetzt.

Syntax UI- Code: **ExpandWidth**
ExpandHeight

NoWiderThanChildren, NoHigherThanChildren

Diese Hints bestimmen, dass ein Objekt nicht größer werden soll, als nötig ist um seine Children aufzunehmen.
Anmerkung: Primaries ignorieren NoWiderThanChildren.

Syntax UI- Code:	**NoWiderThanChildren** **NoHigherThanChildren**

initialSize, minimumSize, maximumSize

Diese Hints legen die anfängliche (initialSize), minimale (minimumSize) bzw. maximale (maximumSize) eines Objekts fest. Das ist häufig nützlich für größenveränderliche Groups, Dialoge, View-Objekte usw., die die Hints ExpandWidth und / oder ExpandHeight gesetzt haben. Primary-Objekte unterstützen die ~Size-Hints jedoch nicht.

Syntax: siehe fixedSize

fixedSize

Das ist der wohl am häufigsten verwendete Size-Hint. Er gibt dem Objekt eine feste Größe. Achten Sie bei der Auswahl der Größenangabe auf eine zweckmäßige "Grundeinheit", z.B. ST_LINES_OF_TEXT oder ST_AVG_CHAR_WIDTH. Details dazu finden Sie im Abschnitt 3.3.2.

Syntax	UI- Code: **fixedSize** = **width** , **height** [, **count**] width: Breite des Objekts height: Höhe des Objekts count: falls angebracht: "Zähler"
	Lesen: **<numVar>** = **<obj>**.**fixedSize** (**n**) n = 0: Breite n = 1: Höhe n = 2: Count n = 3: Test ob der Hint gesetzt ist (TRUE) oder nicht (FALSE)
	Schreiben: **<obj>**.**fixedSize** = **width** , **height** [, **count**] Spezialfall: width oder height = –1: Löschen des Hints.

Beachten Sie, dass es sich um Hints (Hilfen für den Geometriemanager) handelt. Einige Objekte ignorieren diese Werte. Falls das ein Problem darstellt, ist eine häufig geeignete Lösung, das Objekt in eine Group einzuschließen. Groups unterstützen alle diese Hints.

Beispiel:

```
Group   InfoGroup
  Children = InfoText, MoreInfoButton
  orientChildren = ORIENT_VERTICALLY
  ExpandWidth
  ExpandHeight
  DrawInBox
  END Object
```


```
Memo   InfoText
  Caption$ = "Information eingeben"
  fixedSize = 40 + ST_AVG_CHAR_WIDTH,  5 + ST_LINES_OF_TEXT
  END Object

Button MoreInfoButton
  Caption$ ="Mehr Infos"
  justifyCaption = J_CENTER
  ExpandWidth
  END Object
```

Der Parameter "count" wird bei Objekten verwendet, die eine Anzahl an "Unterobjekten" darstellen. Der typische Fall sind "scrollbare" Listen, wie im Beispiel dargestellt. Die Liste ist 15 Zeichen breit, 10 Textzeilen hoch und stellt 10 Einträge gleichzeitig dar.


```
DynamicList dynlist
  caption$= "DynamicList"
  justifycaption = J_TOP
  < ... >
  fixedSize = 15 + ST_AVG_CHAR_WIDTH, \
              10 + ST_LINES_OF_TEXT, 10
  END Object
```

xSize, ySize

Diese Werte liefern die aktuelle Größe des Objekts in Pixeln. Dabei verwenden sie eine GEOS-internen Struktur, die im PC/GEOS-SDK "visBounds" heißt. Diese Struktur speichert die Position und Größe des Rechtecks, dass das gesamte Objekt umschließt. Das umfasst auch einen eventuell automatisch um das Objekt

gezogenen Rahmen oder die "Seiten" eines Buttons. Deswegen können die von xSize und ySize gelieferten Werte größer sein, als Sie mit fixedSize festgelegt haben. Außerdem sind die Werte nur gültig, solange das Objekt auch dem Schirm ist. Dabei darf es auch im Hintergrund oder von anderen Objekten verdeckt sein. Aber für Objekte, die auf visible = FALSE gesetzt oder nicht im generic Tree sind, sind die Werte möglicherweise ungültig.

Syntax Lesen: **<numVar>** = **<obj>**.**xSize**
<numVar> = **<obj>**.**ySize**

Anmerkung: Die Instancevariablen xSize und ySize sind nicht nur für Generic-Class Objekte, sondern auch für VisualClass Objekte (siehe Kapitel 5 des Objekthandbuchs) definiert.

3.3.5 Positionierung der Objekte

Die Positionierung der generischen Objekte ist eine der elementaren Aufgaben des Geometriemanagers. Sie sollten hier nur eingreifen, wenn es unbedingt sein muss. Folgende Möglichkeiten zur Verfügung:

Hint	Syntax im UI-Code	Im BASIC-Code
placeObject	placeObject = **numWert**	lesen, schreiben
fixedPosition	fixedPosition = **x**, **y**	lesen, schreiben
xPosition	—	nur lesen
yPosition	—	nur lesen

placeObject

Dieser Hint ermöglicht es Ihnen, Objekte an Stellen zu platzieren, auf die Sie sonst keinen Zugriff hätten. Dazu stehen die folgenden Konstanten zur Verfügung. Sie dürfen jeweils nur eine Konstante verwenden, eine Kombination (mit +) ist nicht zulässig.

Konstante	Wert	Bedeutung
MENU_BAR	1	Objekt in der Menüleiste platzieren
REPLY_BAR	2	Objekt in der Replybar platzieren
TITLE_BAR_LEFT	4	Objekt links in der Titelzeile platzieren
TITLE_BAR_RIGHT	8	Objekt rechts in der Titelzeile platzieren
	16 - 512	reserviert für Views. Siehe dort.

MENU_BAR
Platziert das Objekt in der Menüzeile eines Primary-Objekts. Das Objekt muss dazu direktes Child des Primary-Objekts sein.

REPLY_BAR
Platziert das Objekt in einer automatisch erzeugten Replybar eines Dialog-Objekts. Das Objekt muss dazu direktes Child des Dialog-Objekts sein. Existiert keine automatisch erzeugte Replybar, so ist dieser Wert wirkungslos.

TITLE_BAR_LEFT, TITLE_BAR_RIGHT
Platziert das Objekt auf der linken (TITLE_BAR_LEFT) oder der rechten (TITLE_BAR_RIGHT) Seite der Titelzeile eines Primary- oder Display-Objekts. Das Objekt muss dazu direktes Child des Primary- oder Display-Objekts sein.

Syntax UI- Code: **placeObject** = **position**
position: Wert aus der Tabelle oben
Lesen: **<numVar>** = **<obj>**.**placeObject**
Schreiben: **<obj>**.**placeObject** = **position**
Spezialfall: position = 0: Löschen des Hints.

---

fixedPosition

Dieser Hint schreibt die Position eines Objekts relativ zu seinem Parent fest. FixedPosition ist extrem hoch priorisiert. Verwenden Sie diesen Hint für generische Objekte nur im äußersten Notfall, da der Geometriemanager ihn auch dann befolgt, wenn er unsinnige Ergebnisse hervorruft und Teile des automatischen Geometriemanagements außer Kraft gesetzt werden können.

Syntax UI- Code: **fixedPosition** = **xPos** , **yPos**
Lesen: **<numVar>** = **<obj>**.**fixedPosition** (n)
n = 0: xPos
n = 1: yPos
n = 2: Test ob der Hint gesetzt ist (TRUE) oder nicht (FALSE)
Schreiben: **<obj>**.**fixedPosition** = **xPos** , **yPos**
Spezialfall: xPos oder yPos = –1: Löschen des Hints.

xPosition, yPosition

Diese Werte liefern die aktuelle Position des Objekts. Dabei verwenden sie eine GEOS-internen Struktur, die im PC/GEOS-SDK "visBounds" heißt. Damit gelten die gleichen Einschränkungen, die bei **xSize** und **ySize** oben beschrieben wurden. Die mit **xPosition** und **yPosition** gelesenen Werte werden für generische Objekte regelmäßig von den eventuell mit **fixedPosition** gesetzten Werten abweichen, da fixedPosition die Position relativ zum Parent-Objekt setzt, die visBounds sich jedoch üblicherweise auf das zugehörige "Window" (Primary, Dialog..) beziehen.

Syntax Lesen: **<numVar>** = **<obj>**.**xPosition**
<numVar> = **<obj>**.**yPosition**

Anmerkung: Die Instancevariablen xPosition und yPosition sind nicht nur für GenericClass Objekte sondern auch für VisualClass Objekte (siehe Kapitel 5 des Objekthandbuchs) definiert.

3.3.6 Spezielle Attribute

Die hier beschriebenen Hints sind zwar auf GenericClass Ebene definiert, zeigen jedoch nicht bei allen Objekten eine Wirkung. Die Objekte, für die sie am häufigsten verwendet werden sind unten jeweils mit angegeben.

Hint	Syntax im UI-Code	Im BASIC-Code
DrawInBox	DrawInBox	—
MakeToolbox	MakeToolbox	—
MakeReplyBar	MakeReplyBar	—
tColor	tColor = **color**	lesen, schreiben
bgColor	bgColor = **col1**, **col2**	lesen, schreiben
NoSeparatorLine	NoSeparatorLine	—

DrawInBox

Dieser Hint zeichnet einen Rahmen um ein Objekt. Er wird sehr häufig für Group-Objekte und Listen-Objekte verwendet.

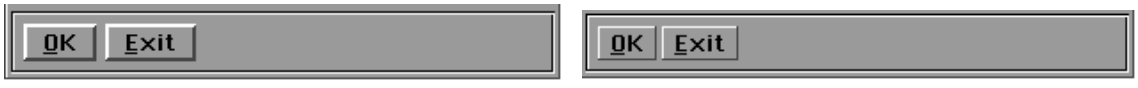
Syntax UI-Code: **DrawInBox**

MakeToolbox

Dieser Hint bewirkt, dass das ein Group-Objekt seine Children als "Werkzeugleiste" (engl. tool box = Werkzeugkasten) darstellt. Viele Objekte stellen sich als "Tool" anders dar, wobei die Funktionalität aber nicht geändert wird.

Syntax UI-Code: **MakeToolbox**

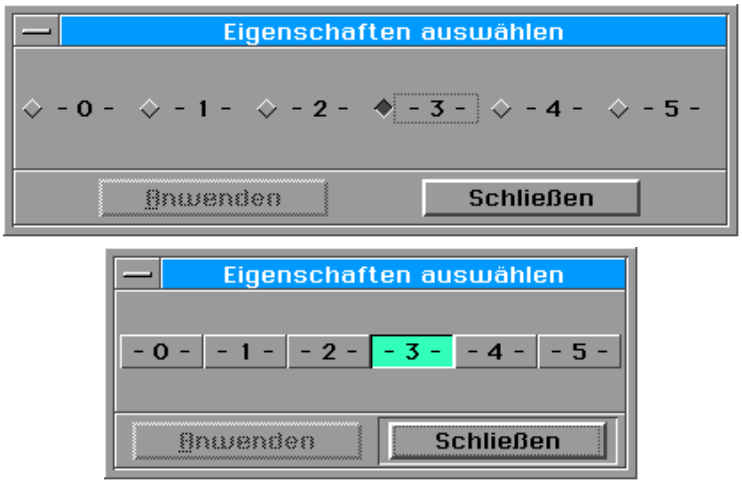
Beispiele:
Eine Group mit **DrawInBox** gesetzt. Rechts ist außerdem **MakeToolbox** gesetzt.



Der UI-Code dazu:

```
Group AReplyBar
  Children = OKButton, ExitButton
  orientChildren = ORIENT_HORIZONTALLY
  DrawInBox
  ExpandWidth
  MakeToolbox
  END Object
```

Ein Listen-Objekt (RadioButtonGroup), links ohne, rechts mit MakeToolBox:



MakeReplyBar

Eine ReplyBar ist die Kontroll-Leiste mit den Buttons OK, Abbrechen usw., die sich in vielen Dialogboxen findet. Die Children werden automatisch auf eine spezielle Weise angeordnet.

Syntax UI-Code: **MakeReplyBar**

```
Group AReplyBar
  Children = OKButton, ExitButton
  MakeReplyBar
  END Object
```

tColor, bgColor

Einige Objekte, insbesondere Buttons und Listeneinträge von scrollbaren Listen, unterstützen farbigen Caption-Text (tColor) und farbige Hintergründe (bgColor). Es gibt dabei zwei Hintergrundfarben: eine für den "unselektierten" und eine für den "selektierten" Zustand.

Syntax UI-Code: **tColor** = **color**
color: Text-Farbe. Es muss eine Indexfarbe sein.
RGB-Farben werden nicht unterstützt.
bgColor = **col1**, **col2**
col1: Unselektierte Farbe. (ebenfalls nur Indexfarbe)
col2: Selektierte Farbe. (ebenfalls nur Indexfarbe)
Lesen: **<numVar>** = **<obj>**.**tColor**
Liefert –1, wenn Hint nicht gesetzt ist
<numVar> = **<obj>**.**bgColor** (n)
n = 0: unselektierte Farbe (col1)

n = 1: selektierte Farbe (col2)
Liefert –1, wenn Hint nicht gesetzt ist

Schreiben: **<obj>**.**tColor** = **color**
color = –1 löscht den Hint

<obj>.**bgColor** = **col1**, **col2**
col1 oder col2 = –1 löscht den Hint

NoSeparatorLine

Dieser Hint beeinflusst nur Groups, die in ein Menü eingebunden sind. Per Default werden sie durch eine Trennlinie abgegrenzt. Damit bekommt das Menü eine bessere Struktur. Dieser Hint entfernt diese Trennlinie.

Syntax UI-Code: **NoSeparatorLine**

3.3.7 Spezielle Hints für Window-Objekte

Die hier beschriebenen Hints sind zwar auf GenericClass Ebene definiert, zeigen jedoch nur bei Window-Objekten wie Primaries, Displays und Dialogen eine Wirkung. Ob ein bestimmter Hint in einer konkreten Situation die beabsichtigte Wirkung hat müssen Sie ausprobieren.

Für Primaries wird z.B. sehr häufig der Hint **SizeWindowAsDesired** verwendet um die Größe des Programmfensters auf das erforderliche Maß zu beschränken.

3.3.7.1 Aussehen und Verhalten anpassen

Hint	Syntax im UI-Code	Im BASIC-Code
NoSysMenu	NoSysMenu	—
NoTitleBar	NoTitleBar	—
WindowNotMovable	WindowNotMovable	—

NoSysMenu

NoSysMenu entfernt das Systemmenu von einem Window-Objekt. Achtung! Mit dem Systemmenu entfernen Sie eventuell den Zugriff auf wichtige Funktionen des Programms!

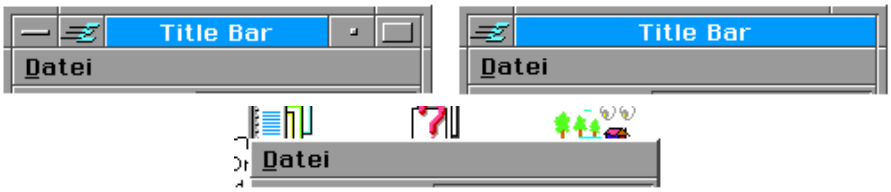
Syntax UI-Code: **NoSysMenu**

NoTitleBar

NoTitleBar entfernt die Titelzeile von einem Window-Objekt. Achtung! Mit der Titelzeile entfernen Sie eventuell den Zugriff auf wichtige Funktionen des Programms!

Syntax UI-Code: **NoTitleBar**

Beispiel



Ein Primary-Objekt: links normal, rechts mit **NoSysMenu** gesetzt und unten mit **NoTitleBar** gesetzt (die Icons im Hintergrund sind jetzt sichtbar).

Der UI-Code für das rechte Bild sieht so aus:

```
Primary APrimary
  SizeWindowAsDesired
  NoTitleBar
  Caption$ = "Title Bar"
    ' wird ignoriert, weil es keine Title Bar mehr gibt :-)
  <.. .. >
  END Object
```

WindowNotMovable

Dieser Hint verhindert, dass ein Fenster, dass normalerweise auf dem Bildschirm verschieblich ist, nicht mehr verschoben werden kann. Sie sollten diesen Hint vorsichtig einsetzten.

Syntax UI-Code:	**WindowNotMovable**

3.3.7.2 Anfängliche Größe festlegen

Die hier beschriebenen Hints legen die Größe eines Fenster-Objekts fest, wenn es erstmalig geöffnet wird. Sie sind nur im UI Code zulässig.

Hint	Syntax im UI-Code
SizeWindowAsDesired	SizeWindowAsDesired
ExtendWindowToBottomRight	ExtendWindowToBottomRight
ExtendWindowNearBottomRight	ExtendWindowNearBottomRight
WindowSizeFromParent	WindowSizeFromParent = **relX**, **relY**
WindowSizeFromScreen	WindowSizeFromScreen = **relX**, **relY**

SizeWindowAsDesired

Dieser Hint verhindert, dass Fenster, oftmals ein Primary-Objekt, am Anfang größer wird, als die Children es benötigen.

Syntax UI-Code:	**SizeWindowAsDesired**

ExtendWindowToBottomRight

Bewirkt, dass sich das Fenster beim erstmaligen Öffnen soweit vergrößert, dass es bis an den rechten und den unteren Rand des Parent Windows reicht.

Syntax UI-Code:	**ExtendWindowToBottomRight**

ExtendWindowNearBottomRight

Wie ExtendWindowToBottomRight, nur dass unten ein kleiner Bereich freigelassen wird.

Syntax UI-Code:	**ExtendWindowNearBottomRight**

WindowSizeFromParent

Dieser Hint bewirkt, dass die Größe eines neu geöffneten Fensters relativ zu seinem Parent-Window festgelegt wird. Beachten Sie, dass das nicht identisch mit dem Parent-Objekt im GenericClass Tree sein muss. Für Dialoge und Primaries ist das Parent Window z.B. der Bildschirm, für Displays ist es die DisplayGroup.

Syntax UI-Code:	**WindowSizeFromParent** = **relX**, **relY**
relX, relY:	relative Größe in Prozent Erlaubte Werte: Null bis 100, jeweils einschließlich

WindowSizeFromScreen

Dieser Hint bewirkt, dass die Größe eines neu geöffneten Fensters relativ zum gesamten Bildschirm festgelegt wird.

Syntax UI-Code:	**WindowSizeFromScreen** = **relX**, **relY**
relX, relY:	relative Größe in Prozent Erlaubte Werte: Null bis 100, jeweils einschließlich

3.3.7.3 Anfängliche Position festlegen

Die hier beschriebenen Hints legen die Position eines Fenster-Objekts fest, wenn es erstmalig geöffnet wird. Sie sind nur im UI-Code zulässig.

Hint	Syntax im UI-Code
PositionWindowAtMouse	PositionWindowAtMouse
WindowPositionFromParent	WindowPositionFromParent = **relX**, **relY**
StaggerWindow	StaggerWindow
CenterWindow	CenterWindow
TileWindow	TileWindow
WindowNoConstraints	WindowNoConstraints

PositionWindowAtMouse

Dieser Hint bewirkt, dass linke obere Ecke eines Fensters, z.B. eines Dialogs oder eines Primary, beim Öffnen an der aktuellen Mausposition steht. Das wir häufig für Kontextmenüs genutzt.

Syntax UI-Code: **PositionWindowAtMouse**

WindowPositionFromParent

Dieser Hint bewirkt, dass die Position eines neu geöffneten Fensters relativ zu seinem Parent-Window festgelegt wird. Beachten Sie, dass das nicht identisch mit dem Parent-Objekt im GenericClass Tree sein muss. Für Dialoge und Primaries ist das Parent Window z.B. der Bildschirm, für Displays ist es die DisplayGroup.

Syntax UI-Code: **WindowPositionFromParent** = **relX**, **relY**
relX, relY: relative Position in Prozent
Erlaubte Werte: Null bis 100, jeweils einschließlich

StaggerWindow, CenterWindow, TileWindow

Diese Hints bewirken, dass mehrere Fenster innerhalb eines Parent-Windows (für Displays z.B. die DisplayGroup), die einen dieser Hints gesetzt haben, auf bestimmte Weise angeordnet werden.

StaggerWindow	überlappend (engl. staggered: gestaffelt)
CenterWindow	zentriert
TileWindow	Platz aufteilen

Syntax UI-Code: **StaggerWindow**
CenterWindow
TileWindow

WindowNoConstraints

Dieser Hint entfernt alle Einschränkungen (engl.: constraints) bezüglich der Fensterpositionierung. Er sollte nur als letzter Versuch benutzt werden, um ein Fenster zu positionieren.

Syntax UI-Code:	**WindowNoConstraints**

3.3.7.4 Window Management

Methode	Aufgabe
BringToTop	Window nach vorne holen
LowerToBottom	Window nach hinten stellen
MoveWin	Window verschieben
ResizeWin	Windows Größe ändern

BringToTop

Diese Methode bringt das betroffene Fenster nach vorne. Das Fenster erhält automatisch den Focus.

Syntax Basic-Code: **<obj>.BringToTop**

LowerToBottom

Diese Methode stellt das betroffene Fenster nach hinten. Es verliert automatisch den Focus, falls es ihn besaß.

Syntax Basic-Code: **<obj>.LowerToBottom**

MoveWin

Diese Methode verschiebt das Window an eine bestimmte Position. Die Position kann in Pixeln angegeben werden (Parameter mode nicht angegeben oder Null) oder als Prozentwert der Größe des übergeordneten Windows (Parameter mode ungleich Null).

Syntax Basic-Code: **<obj>.MoveWin xPos, yPos** [, **mode**]

ResizeWin

Diese Methode ändert die Größe eines Window-Objekts. Die neue Größe kann in Pixeln angegeben werden (Parameter mode nicht angegeben oder Null) oder als Prozentwert der Größe des übergeordneten Windows (Parameter mode ungleich Null).

Syntax Basic-Code: **<obj>.ResizeWin xPos, yPos** [, **mode**]

ResizeWin arbeitet unter Umständen nicht, wenn Sie widersprüchliche Anweisungen geben, z.B. gleichzeitig fixedSize oder SizeWindowAsDesired setzen.

3.3.8 Hintertürchen für Programmierer

Die in den letzten Abschnitten besprochenen Hints, wie z.B. **ExpandWidth** oder **NoTitleBar** sind nur im UI-Code verfügbar:

```
Button MyButton
  Caption$ = "Drück mich"
  ExpandWidth                    ! Maximale Breite einnehmen
  End Object
```

Die "normale" Syntax von R-BASIC erlaubt es nicht, dass sie zur Laufzeit gesetzt oder gelöscht werden. Das ist im Normalfall auch nicht notwendig. Wenn Sie jedoch beispielsweise zur Laufzeit eigene Objekte anlegen (vgl. Kapitel 2.1.5) könnte auch der Bedarf bestehen, solche Hints zu setzen. R-BASIC bietet zur Lösung dieses Problems zwei Befehle an: einen, mit dem ein sonst nur im UI-Code verfügbarer Hint gesetzt werden kann und einen, mit dem er gelöscht wird.

Befehl - Syntax im BASIC-Code	Aufgabe
ObjAddHint **<obj>**, **code** [, **adr**, **size**]	Setzen eines UI Hints
ObjRemoveHint **<obj>**, **code**	Löschen eines UI Hints

Diesen Befehlen wird nicht etwa der Name des Hints übergeben, sondern sein numerischer Code. Was auf den ersten Blick etwas umständlich wirkt hat einen wesentlichen Vorteil: **Sie können auf diese Weise R-BASIC Objekten auch Instance-Variablen bzw. Hints geben, die zwar im PC/GEOS-SDK definiert aber in R-BASIC nicht bekannt sind.** Die folgende Tabelle enthält eine paar der häufiger verwendeten Geometrie-Hints, eine vollständige Liste der Codes der in R-BASIC definierten Hints finden Sie im Anhang, Kapitel E. Weitere Codes können Sie mit dem PC/GEOS-SDK bzw. der PC/GEOS-SDK-Dokumentation erhalten.

Einige häufiger verwendete UI-Hint Codes. Weitere Codes finden Sie im Anhang.

UI - Instance Variable bzw. Hint	numerischer Code
MinimizeChildSpacing	25068
IncludeEndsInChildSpacing	24728
ExpandWidth	24712
ExpandHeight	24708
DrawInBox	24704
MakeToolBox	24976
MakeReplyBar	24744
SizeWindowAsDesired	24936
PrimaryFullScreen	27136
DisplayCurrentSelection	26652

Wenn sie öfter mit diesen beiden Funktionen arbeiten, können Sie sich für die von Ihnen verwendeten Hints natürlich Konstanten definieren. Beachten Sie, dass ich deren Namen von denen der Instance-Variablen unterscheiden müssen, damit der Compiler sie auseinanderhalten kann.

```
CONST   Draw_In_Box = 24704
CONST   C_MakeToolBox = 24976
```

Warnung! ObjAddHint und ObjRemoveHint führen keinerlei Fehlerkontrollen aus, die Parameter werden direkt an die entsprechenden PC/GEOS-SDK-Routinen weitergereicht. Insbesondere wird nicht abgeprüft ob:
- der übergebene Code überhaupt gültig ist
- der übergebene Code für das konkrete Objekt gültig ist
- die Datenwerte, falls es welche gibt, zum Code passen.

Im günstigsten Fall werden die fehlerhaften Werte oder Codes ignoriert, im ungünstigsten Fall kann es zum Systemabsturz kommen.

Hinweis für PC/GEOS-SDK-Programmierer: ObjAddHint und ObjRemoveHint verwenden intern die Messages MSG_META_ADD_VAR_DATA bzw. MSG_META_DELETE_VAR_DATA. Sie können also alles machen, was Sie im PC/GEOS-SDK mit diesen beiden Messages machen können. Das Flag VDF_SAVE_TO_STATE wird jeweils gesetzt. Beide Befehle setzen die Objekte bei Bedarf "not usable" und nehmen dies, wenn nötig, auch wieder zurück.

ObjAddHint

Syntax im BASIC Code: **ObjAddHint <ob>, code [, adr, size]**

<obj>	Referenz auf ein Objekt
code	numerischer Code des Hints
adr	Adresse, falls der Hint Datenwerte benötige
size	Größe dieser Datenwerte

Der Befehl **ObjAddHint** fügt einen Hint oder eine Instance-Variable zu einem Objekt hinzu. Verwenden Sie diesen Befehl, wenn die R-BASIC Syntax ansonsten das Setzen des Hints oder der Instance-Variablen zur Laufzeit nicht zulässt oder Sie einen Hint setzen wollen, der von R-BASIC nicht unterstützt wird.

Sie können den Befehl auf alle Objekte anwenden. Er arbeitet sowohl mit GenericClass- als auch auf VisClass-Objekten, egal ob sie vom Compiler oder zur Laufzeit angelegt wurden.

Beispiel: Hinzufügen der Hints **ExpandWidth** und **ExpandHeight** zu einem Objekt und **MakeToolBox** zu einem anderen.

```
DIM ob as OBJECT

  ObjAddHint MyGroup, 24712
  ObjAddHint MyGroup, 24708

  ob = MyOtherObject
  ObjAddHint ob, 24976
```

Ein komplexeres Beispiel finden Sie unten.

ObjRemoveHint

Syntax im BASIC Code: **ObjRemoveHint <obj>, code**

<obj>	Referenz auf ein Objekt
code	numerischer Code des Hints

Der Befehl **ObjRemoveHint** entfernt einen Hint oder eine Instance-Variable von einem Objekt. Verwenden Sie diesen Befehl, wenn die R-BASIC Syntax ansonsten das Löschen des Hints oder der Instance-Variablen zur Laufzeit nicht zulässt oder Sie einen Hint entfernen wollen, der von R-BASIC nicht unterstützt wird.

Sie können den Befehl auf alle Objekte anwenden. Er arbeitet sowohl mit GenericClass- als auch auf VisClass-Objekten, egal ob sie vom Compiler oder zur Laufzeit angelegt wurden. Es ist auch zulässig ObjRemoveHint für eine Hint zu rufen, der gar nicht gesetzt ist.

Beispiel: Entfernen der Hints **DrawInBox**, **MakeToolBox** und **MakeReplyBar** von diversen Objekten.

```
DIM ob as OBJECT

  ObjRemoveHint MyGroup, 24704
  ObjRemoveHint MyGroup.parent, 24976

  ob = MyButton
  ObjRemoveHint ob.parent, 24744
```

Komplexes Beispiel

Nehmen wir an, Sie wären ein PC/GEOS-SDK-Programmierer. Sie wissen daher, dass die R-BASIC Instance-Variable **bgColor** über den SDK-Hint HINT_GADGET_BACKGROUND_COLORS realisiert ist. Mit Hilfe des SDK haben Sie erfahren, dass

- HINT_GADGET_BACKGROUND_COLORS eine numerische Konstante mit dem Wert (Code) 25072 ist.
- es 4 Byte als Datenwerte benötigt - zwei Farbwerte für den Vordergrund und zwei für den Hintergrund.

R-BASIC erwartet dagegen nur zwei Farbwerte, es setzt für die beiden Vordergrundfarben und für die beiden Hintergrundfarben jeweils den gleichen Wert.

Sie wollen das nun ändern und alle vier Farben verwenden. Dazu müssen Sie die Datenwerte mit dem Befehl POKE oder einem seiner Verwandten in den R-BASIC Speicher schreiben. Welche Adresse Sie dazu verwenden ist egal, nehmen wir an, Sie entscheiden sich für 100.

Die Befehlsfolge sieht dann so aus:

```
POKE 100, RED
POKE 101, YELLOW
POKE 102, CYAN
POKE 103, BLUE

ObjAddHint MyButton, 25072, 100, 4
```

Sie erhalten einen Button, dessen Hintergrundfarbe im ungedrückten Zustand ein Punktraster aus Rot und Gelb ist, im gedrückten Zustand ist es ein Raster aus Blau und Cyan.

Auf diese Weise können Sie R-BASIC Objekten auch Eigenschaften unterschieben, die von R-BASIC selbst nicht unterstützt werden.

3.4 Die "Apply"-Message

Einige R-BASIC Objekte enthalten Informationen oder Daten, die vom Nutzer geändert werden können und dann "zur Anwendung" gebracht werden, indem das Objekt eine Message aussendet (d.h. den zugehörigen ActionHandler aufruft). Diese Message heißt "Apply-Message" bzw. der Handler "ApplyHandler" (engl. to apply: anwenden). Das sind konkret die folgenden Objekt-Klassen. Eine ausführliche Beschreibung der Objekte finden Sie im Kapitel 4 dieses Handbuchs.

- Das **Number**-Objekt stellt eine Zahl dar, die vom Nutzer verändert werden kann. Drückt der Nutzer im Eingabefeld des Text-Objekts die Enter-Taste oder klickt er auf die "Pfeile", so wird die Apply-Message gesendet, d.h. der ApplyHandler wird aufgerufen.
- Die Text-Objekte **Memo** und **InputLine** enthalten einen Text. Drückt der Nutzer z.B. im **InputLine** Objekt die Enter-Taste, so wird die Apply-Message gesendet.
- Die Listen-Objekte **OptionGroup**, **RadioButtonGroup** und **DynamicList**. Hier wird die Apply-Message gesendet wenn der Nutzer einen Listeneintrag anwählt.

Neben einem ApplyHandler besitzen diese Objektklassen auch eine Status-Handler, der im "delayed Mode" (siehe nächstes Kapitel) benötigt wird.

Instancevariable	Syntax im UI-Code	Im BASIC-Code
ApplyHandler	ApplyHandler = <Handler>	nur schreiben
StatusHandler	StatusHandler = <Handler>	nur schreiben

3.4.1 Manuelles Auslösen der Apply-Message

Die ApplyHandler der oben angegebenen Objekte sind bei den entsprechenden Objekten definiert und werden dort ausführlich besprochen, da sie je nach Objekt unterschiedliche Parameter haben.

Es gibt jedoch eine auf GenericClass-Ebene definierte Methode und zwei dazugehörige Instance-Variablen, die ein manuelles auslösen der Apply-Message bei Bedarf, d.h. vom BASIC-Code aus, ermöglicht.

Methoden:

Methode	Aufgabe
Apply	Auslösen der Apply-Message

Syntax BASIC-Code: **<obj>.Apply**

Instance-Variablen

Hint	Syntax im UI-Code	Im BASIC-Code
ApplyEvenIfNotModified	ApplyEvenIfNotModified	—
ApplyEvenIfNotEnabled	ApplyEvenIfNotEnabled	—

Syntax UI-Code:	**ApplyEvenIfNotModified** **ApplyEvenIfNotEnabled**

Apply

Die Methode **Apply** fordert ein Objekt auf, seine Apply-Message zu senden (d.h. seinen ApplyHandler aufzurufen). Objekte, die von Hause aus keinen ApplyHandler haben (wie z.B. **Group**'s, **Dialog**- und **Primary**-Objekte) reichen diese Methode an ihre Children weiter. Das heißt konkret, dass es ausreicht die Apply-Methode eines **Group**-Objekts aufzurufen und alle seine Children bzw. deren Children usw. senden ihre Apply-Message aus, falls sie eine besitzen. Dieses Konzept ist bei genauerer Betrachtung extrem leistungsfähig, da man sich dadurch viel Arbeit ersparen kann.

Wichtig 1: Die oben angegeben Objekte (Number, Text- und Listen-Objekte) senden ihre Apply-Message nur dann aus, wenn sie "modified" (geändert) sind. Ändert der Nutzer das Objekt, klickt er z.B. einen Listeneintrag an, so passiert das automatisch. Vom Basic-Code aus müssen wir aber i.A. selbst dafür sorgen, das Objekt auf "modified" zu setzen. Alle betroffenen Objekte besitzen eine entsprechende Instance-Variable.

Wichtig 2: Nachdem das Objekt seine Apply-Message ausgesendet hat wird der "modified"-Zustand automatisch zurückgesetzt.

Beispiel: Nehmen wir an, wir haben eine Listen-Objekt namens DefaultOptions. Von diesem soll beim Programmstart ein bestimmter Eintrag selektiert werden, der z.B. aus einer Datei gelesen wurde und daher nicht von vorneherein bekannt ist. Dann soll die Liste ihre Apply-Message aussenden um den Rest des Programms über ihren Zustand zu informieren. Das Belegen der passenden Instance-Variable namens "selection" macht die Liste aber nicht "modified". Damit sie ihre Apply-Message aussendet müssen wir das selbst tun. Das Ganze verpacken wir in eine SUB namens InitList:

```
SUB InitList (entry as INTEGER)
  DefaultOptions.selection = entry
  DefaultOptions.modified = TRUE
  DefaultOptions.Apply
  END SUB
```

ApplyEvenIfNotModified, ApplyEvenIfNotEnabled

Wie oben erwähnt muss ein Objekt, dass eine Apply-Message aussenden soll "modified" sein. Die Hints **ApplyEvenIfNotModified** bzw. **ApplyEvenIfNot-Enabled** sorgen dafür, dass ein Objekt seine Apply-Message beim Aufruf der **Apply**-Methode auch dann aussendet, wenn es nicht "modified" oder sogar nicht

"enabled" ist. Diese Hints sind zwar auf GenericClass-Level definiert, zeigen aber nur bei den Objekten, die auch über einen Apply-Handler verfügen, eine Wirkung.

3.4.2 Delayed Mode und Status-Message

Das im Folgenden beschriebenen Eigenschaften und Verhaltensweisen von Objekten sind - richtig eingesetzt - sehr leistungsfähig und können dem Programmierer viel Arbeit ersparen, sie erfordern jedoch ein gutes Überblickswissen über das R-BASIC Objektsystem. Sie sind daher eher etwas für den fortgeschrittenen Programmierer. Meistens ist die gleiche Funktionalität auch anders, dann allerdings mit etwas mehr Programmcode, erreichbar.

Wie oben erwähnt senden die anfangs aufgeführten Objekte (Number, Text- und Listen-Objekte) ihre Apply-Message sofort aus, wenn der Nutzer z.B. eine Listeneintrag auswählt oder auf einen "Pfeil" eines Number-Objekts klickt. Es gibt jedoch auch Situationen, in denen dieses sofortige Reagieren nicht erwünscht ist. In einem komplexen Dialog, in dem z.B. Farbe, Form und Größe eines Objekts eingestellt werden, kann es sinnvoll sein, dass der Nutzer zunächst alle Einstellungen vornimmt und diese Einstellungen dann "auf einmal" angewendet werden sollen. Hinzu kommt, dass gegebenenfalls zwei oder mehrere Objekte voneinander abhängen. In einen "Drucken" Dialog z.B. muss sichergestellt werden, dass die erste zu druckende Seitennummer nicht größer als die letzte zu druckende Seitennummer ist. Die entsprechenden Number-Objekte müssen also miteinander kommunizieren, ohne dass die Änderungen "angewendet" (d.h. die Seiten gedruckt) werden. In R-BASIC wird dieses Verhalten als **Delayed Mode** bezeichnet (engl. to delay: verzögern). Die eigentliche Apply-Message der betroffenen Objekte wird verzögert, nämlich erst auf Anforderung, ausgesendet. Statt ihrer Apply-Message senden die entsprechenden Objekte zunächst eine sogenannte **Status-Message** aus, d.h. es wird der **StatusHandler** aufgerufen. Dieser kann genutzt werden, um andere Objekte zu informieren. Die entsprechende Instance-Variable (StatusHandler) ist genau wie der ApplyHandler bei den jeweiligen Objekten definiert.

Instance-Variable	Syntax im UI-Code	Im BASIC-Code
MakeDelayedApply	MakeDelayedApply	—

Syntax UI-Code: **MakeDelayedApply**

Die auf GenericClass-Ebene definierte Instance-Variable MakeDelayedApply versetzt ein Objekt und seine Children in den Delayed Mode. Sehr häufig ist es deshalb so, dass ein Group-Objekt diese Anweisung im UI-Code erhält, so das alle seine Children, deren Children usw. im Delayed Mode arbeiten. Um die Apply-Message der betroffenen Objekte auszusenden reicht es, die Apply-Methode des Group-Objekts aufzurufen, da diese, wie im vorherigen Abschnitt beschrieben, an alle Children weitergegeben wird.

Beispiel: Eine Gruppe von Objekten innerhalb der Group "BottomGroup" arbeitet im Delayed Mode. Die beiden Number-Objekte kommunizieren über Status-Handler miteinander. Der ApplyButton löst ein "Anwenden" (Aufruf der ApplyHandler) aus.

UI-Code

```
Group BottomGroup
  Children = Memo1, RightGroup
  orientChildren = ORIENT_HORIZONTALLY
  DrawInBox
  MakeDelayedApply    ' wird an die Children
                      ' weitergereicht
  END Object

Group RightGroup
  Children = Number1, Number2, ApplyButton
  orientChildren = ORIENT_VERTICALLY
  justifyChildren = J_RIGHT + J_BOTTOM
  ExpandHeight
  END Object

Number Number1
  Caption$ = "Von "
  ApplyHandler = NumberVonHandler
  StatusHandler = StatusNum1
  END Object

Number Number2
  Caption$ = "Bis "
  ApplyHandler = NumberBisHandler
  StatusHandler = StatusNum2
  END Object

Button ApplyButton
  Caption$ = " Anwenden"
  ActionHandler = DoApply
  END Object


Memo Memo1
  Caption$ = "Note:"
  justifyCaption = J_TOP
  text$ ="Hier Text eingeben ..."
  fixedSize = 30 + ST_AVG_CHAR_WIDTH, 5 + ST_LINES_OF_TEXT
  ApplyHandler = txtAction
  END Object
```

Basic-Code

```
NUMBERACTION NumberVonHandler
  Print "Von:";value
  END ACTION

NUMBERACTION NumberBisHandler
  Print "Bis:";value
  END ACTION

NUMBERACTION StatusNum1
  IF value > Number2.value THEN
    Number2.value = value
    Number2.modified = TRUE       ' wurde verändert!
    END IF
  END ACTION

NUMBERACTION StatusNum2
  IF value < Number1.value THEN
    Number1.value = value
    Number1.modified = TRUE       ' wurde verändert!
    END IF
  END ACTION

BUTTONACTION DoApply
  BottomGroup.Apply
  END Action
```

Wenn Sie dieses Beispiel testen werden Sie feststellen, dass

- Die ApplyHandler nur gerufen werden, wenn das entsprechende Objekt vorher geändert (modifiziert) wurde.
- Ein zweites Anklicken des ApplyButtons nichts bewirkt, es sei denn, Sie haben eins der Objekte zwischendurch wieder geändert.

Das ist aus Performance-Gründen so und liegt daran, dass, wie letzten Abschnitt beschrieben, ApplyHandler nur gerufen werden, wenn das Objekt "modified" ist. Wenn Sie möchten, dass die ApplyHandler auf jeden Fall gerufen werden, egal ob das Objekt modified ist oder nicht, können Sie die Hints **ApplyEvenIfNotModified** bzw. **ApplyEvenIfNotEnabled** aus dem letzten Kapitel verwenden.

Beispiel UI-Code

```
Number Number2
  Caption$ = "Bis "
  ApplyHandler = NumberBisHandler
  StatusHandler = StatusNum2
  ApplyEvenIfNotModified
  END Object
```

Bei Bedarf ist es möglich, die StatusMessage manuell auszulösen. Dazu wird die Methode **SendStatus** verwendet. Diese Methode ist für die oben genannten Objektklassen definiert. Das sind: Number, Memo, InputLine, OptionGroup, RadioButtonGroup und DynamicList.

Methode	Aufgabe
SendStatus	Auslösen der Status-Message

Syntax BASIC-Code: **<obj>.SendStatus**

Hinweis für Dialog-Objekte: Natürlich können Sie ganze Dialogboxen mit dem Hint MakeDelayedApply in den Delayed Mode versetzen. Häufig ist es aber besser stattdessen die Dialog-Instance-Variable dialogType auf den Wert DT_DELAYED_APPLY zu setzen. Dadurch erzeugt der Dialog automatisch einen Apply-Button, und nimmt Ihnen auch sonst viel Arbeit ab. Eine ausführliche Beschreibung zum Dialog-Objekt finden Sie im Kapitel 4.6, der Delayed Mode für Dialoge ist im Kapitel 4.6.6.5 beschrieben.

Schlussbemerkung: Der Delayed Mode ist angebracht und sehr effektiv, wenn die betroffenen Objekte ihre Apply-Message einzeln und unabhängig voneinander senden sollen. Für den Fall, dass Sie erst die Informationen von allen beteiligten Objekten sammeln müssen, bevor Sie fortfahren können, ist es eventuell sinnvoller den Objekten gar keinen ApplyHandler zu geben und die Informationen direkt von den Objekten abzufragen, wie in folgendem Codebeispiel gezeigt:

```
BUTTONACTION DoApply
DIM  von, bis, info$
  von = Number1.value
  bis = Number2.value
  info$ = Memo1.text$
  <.. Auswertung ..>
  END Action
```

(Leerseite)