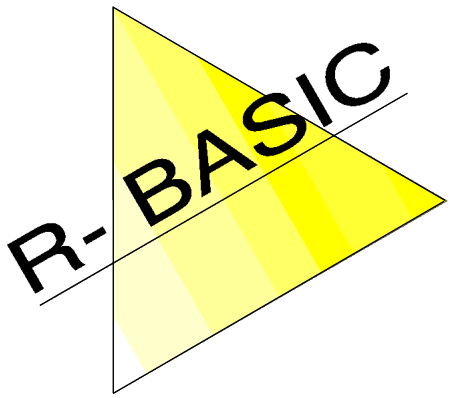
R-BASIC

Einfach unter PC/GEOS programmieren



Objekt-Handbuch

Volume 5
Listen-Objekte, View und Content

Version 1.0

(Leerseite)

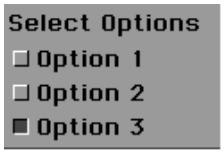
Inhaltsverzeichnis

(Leerseite)

4.8 Listen-Objekte

4.8.1 Überblick

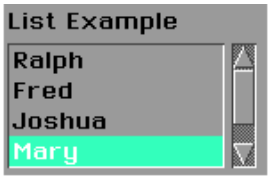
In R-BASIC gibt es 3 Typen von Listen-Objekten:



- Die **OptionGroup** verwaltet eine Liste von Option-Objekten, das jedes für sich und abhängig voneinander den Zustand "ein" oder "aus" haben kann.

- Die **RadioButtonGroup** verwaltet eine Liste von RadioButton-Objekten.

  Diese können einzeln oder in Gruppen ausgewählt (selektiert) werden. Die RadioButtonGroup ist ein sehr vielseitiges Objekt das insbesondere dann zum Einsatz kommt, wenn die Anzahl der Listeneinträge von vorneherein bekannt und unveränderlich ist.



- Die **DynamicList** stammt von der RadioButtonGroup ab und ist daher genauso vielseitig wie diese. Sie wird eingesetzt, wenn die Anzahl der Listeneinträge nicht von vorneherein bekannt ist und / oder sich während des Programmablaufs verändert.

Alle Listen können einen ActionHandler aufrufen, wenn vom Nutzer ein Eintrag selektiert bzw. geändert wird. Über die Instance-Variable **look** kann das Aussehen der Listen weitgehend verändert werden. So können alle Listen-Objekte - nicht nur die DynamicList - als scrollbare Listen auftreten. Die Größe eines als scrollbare Liste erscheinenden List-Objekts wird häufig über den Geometrie-Hint **fixedSize** eingestellt. Beispiele dazu finden Sie bei der Beschreibung der Instance-Variablen **look** der OptionGroup.

Die folgenden Ausführungen gehen zunächst grundsätzlich davon aus, dass die Listen-Objekte im normalen Modus (nicht im sogenannten "Delayed Mode") arbeiten. Das ist der Normalfall, wenn man nicht spezielle Hints setzt, um in den Delayed Mode zu kommen. Dieser "Delayed Mode" ist ausführlich im Kapitel 3.4.2 (Delayed Mode und Status-Message) dieses Handbuchs beschrieben.

Action-Handler-Typen:

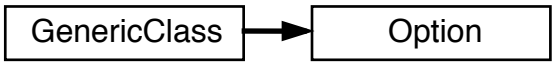
Handler-Typ	Parameter
ListAction	(sender as object, selection as word, modified as word, numSelections as word)

Alle ActionHandler der List-Objekte müssen als **ListAction** deklariert sein. Die Bedeutung der Parameter "selection", "modified" und "numSelections" variiert je nach Listen-Objekt und ActionHandler. Gelegentlich sind einige der Parameter auch bedeutungslos für den speziellen Fall.

4.8.2 Option und OptionGroup

Option

Abstammung



Spezielle Instance-Variablen

Instance-Variable	Syntax im UI-Code	Im BASIC-Code
identifier	identifier = **numWert**	lesen, schreiben

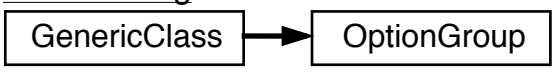
identifier

Die Instance-Variable **identifier** identifiziert das einzelne Option-Objekt. Sie ist vom Typ WORD. Der Wert muss eine 2er-Potenz sein (nur genau 1 Bit gesetzt, d.h. einer der Werte 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 81912, 16384, 32768). Option-Objekte müssen Children einer OptionGroup sein. Innerhalb einer OptionGroup darf jeder Identifier-Wert nur genau einmal vorkommen.

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

OptionGroup

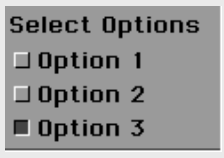
Abstammung



Eine OptionGroup managed eine Liste von Option-Objekten (bis zu 16). OptionGroup-Objekte können nur Option-Objekte als Children haben.

Beispiel UI-Code:

```
OptionGroup ListOfOptions
  Caption$= "Select Options"
  justifyCaption = J_TOP
  Children = bool1, bool2, bool3
  OrientChildren = ORIENT_VERTICALLY
  ApplyHandler = BoolApply
  selection = 4
  END Object

Option bool1: caption$="Option 1":identifier = 1: End Object
Option bool2: caption$="Option 2":identifier = 2: End Object
```



```
Option bool3
  Caption$="Option 3" : identifier = 4:
  End Object
```

Zur Demonstration wurde in an einigen Stellen im Code oben die Syntax mit mehreren, durch Doppelpunkt getrennten Anweisungen verwendet, die auch im UI-Code zulässig ist.

Spezielle Instance-Variablen

Instance-Variable	Syntax im UI-Code	Im BASIC-Code
selection	identifier = **numWert**	lesen, schreiben
isSelected	—	nur lesen
look	look = **numWert**	lesen, schreiben
modified	modified = **numWert**	lesen, schreiben
isModified	—	nur lesen
ApplyHandler	ApplyHandler = **<Handler>**	nur schreiben
StatusHandler	StatusHandler = **<Handler>**	nur schreiben

Methoden:

Methode	Aufgabe
MakeVisible n	Einen bestimmten Eintrag sichtbar machen
SendStatus	Status-Handler aufrufen

selection

Options können den Zustand "ein" oder "aus" haben. Die Instance-Variable **selection** der OptionGroup enthält die Summe der identifier derjenigen Options, die auf "ein" sind. Genauer gesagt ist es die logische Oder-Verknüpfung der **identifier**. Wenn man sich an die Regel hält, dass Option-Identifier nur 2er-Potenzen sein dürfen (was man unbedingt sollte), sind beide Aussagen gleichwertig.

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

isSelected

IsSelected prüft, ob ein bestimmtes Option-Objekt, gegeben durch seinen Identifier, selektiert ist oder nicht.

Syntax Lesen: **<numVar>** = **<obj>** . **isSelected** (**n**)
<obj> OptionGroup Objekt
n: Identifier eines Option-Objekts

look

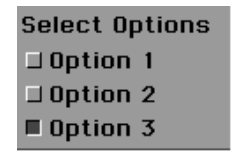
Instance-Variable **look** bestimmt das Aussehen der OptionGroup und ihrer Option-Objekte (engl. look : Aussehen, äußere Erscheinung). Funktionell sind alle Looks identisch.

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

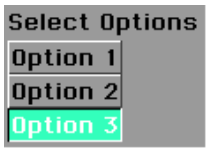
Für alle Listen-Objekte stehen folgende Looks zur Verfügung:

Konstante	Wert	Aussehen
LOOK_NORMAL	0	Klassisches Aussehen
LOOK_SCROLLABLE	1	Scrollbare Liste
LOOK_MINIMIZE_SIZE	2	Minimaler Platzverbrauch
LOOK_TOOLBOX	4	ToolBox-Style

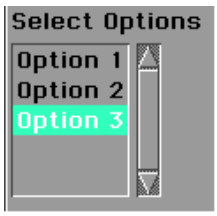
Die Look-Werte können kombiniert werden (mit +), was insbesondere bei LOOK_MINIMIZE_SIZE + LOOK_TOOLBOX gelegentlich Sinn macht. Ungültige bzw. widersprüchliche Kombinationen können jedoch zu seltsamen Effekten führen.



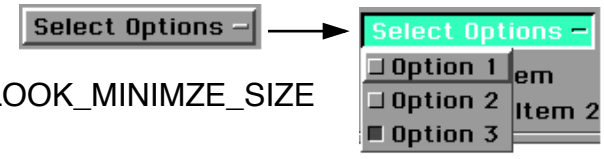
LOOK_NORMAL



LOOK_TOOLBOX



LOOK_SCROLLABLE

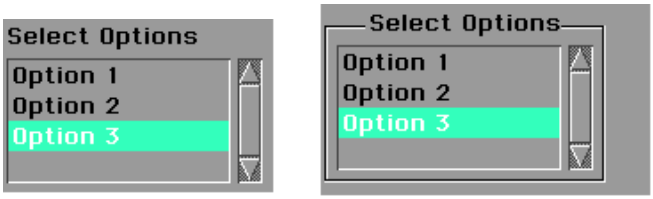


LOOK_MINIMZE_SIZE

Das Bild zeigt die Liste mit dem UI-Code von oben, jedoch jeweils mit verschiedenen Werten für **look** gesetzt.

Insbesondere bei scrollbaren Listen besteht häufig der Bedarf die Größe und die Anzahl der gleichzeitig sichtbaren Listeneinträge festzulegen. Dazu eignet sich der Geometrie-Hint **fixedSize**. Die in den folgenden Bildern dargestellten Listen haben die Instance-Variable **look** auf LOOK_SCROLLABLE und folgenden fixedSize-Hint gesetzt:

```
fixedSize = 15 + ST_AVG_CHAR_WIDTH, 4 + ST_LINES_OF_TEXT, 4
```



Die Listen-Objekte im Bild sind 15 Zeichen breit und das Listen-Fenster ist 4 Zeilen hoch (4 + ST_LINES_OF_TEXT). Der letzte Parameter (4) bestimmt, dass 4 Einträge im Fenster gleichzeitig dargestellt werden sollen. Sinnvollerweise ist er identisch mit der Höhe, gemessen in Textzeilen. Die Liste rechts im Bild hat - zur Demonstration - zusätzlich den Hint **DrawInBox** gesetzt.

modified

Die Instance-Variable **modified** der OptionGroup enthält die logische OR-Verknüpfung der Identifier derjenigen Options, die seit der letzten Apply-Aktion modifiziert wurden.

Beachten Sie, dass ein Verändern des Objekts vom BASIC-Code aus (z.B. Belegen der Instance-Variable **selection**), das Objekt nicht als "modified" markiert, d.h. der Wert der Instance-Variablen **modified** wird nicht verändert. Sie können dies bei Bedarf selbst machen, indem Sie die Anweisung "<obj>.modified = **numWert**" verwenden, wobei "numWert" ein einzelner Identifier oder die OR-Verknüpfung mehrerer Identifier der Option Objekte aus der OptionGroup sein soll.

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

Wenn Sie die Instance Variable **modified** lesen, werden Sie feststellen, dass sie Null enthält, es sei denn, Sie haben sie explizit auf einen anderen Wert gesetzt. Ändert der Nutzer nämlich des Zustand eines Option-Objekts, so passiert intern folgendes:

- Die Instance-Variable **modified** wird mit dem Identifier des betroffenen Option-Objekts belegt.
- Es wird geprüft ob ein ApplyHandler vorhanden ist und dieser wird ggf. aufgerufen. Der Wert von **modified** wird dem Handler übergeben.
- Die Instance-Variable **modified** wird zurückgesetzt (mit Null belegt).

Hinweis: Im sogenannten Delayed Mode (siehe entsprechendes Kapitel weiter unten) werden die letzten beiden Schritte nicht ausgeführt, so dass die Instance-Variable **modified** eine eigene Bedeutung erhält.

isModified

IsModified liest den "modified"-Zustand eines bestimmten Option-Objekts, gegeben durch seinen Identifier, aus.

Syntax	Lesen:	**<numVar>** = **<obj>** . **isModified** (**n**)
	<obj>	OptionGroup Objekt
	n:	Identifier eines Option-Objekts

ApplyHandler

Der **ApplyHandler** der OptionGroup wird aufgerufen, wenn eines der Option-Objekte in der Group geändert wird. Er muss als **ListAction** deklariert sein.

Syntax	UI- Code:	**ApplyHandler** = **<Handler>**
	Schreiben:	**<obj>**.**ApplyHandler** = **<Handler>**

Beispiel, passend zum UI-Code oben:

```
ListAction BoolApply
  Print selection; modified
  IF modified AND 1 THEN Print "Option 1 geklickt"
  IF modified AND 4 THEN Print "Option 3 geklickt"
  END Action
```

Der Parameter **selection** enthält die selektierten Options (OR-Verknüpfung, d.h. die Summe der Identifier).
Der Parameter **modified** enthält den Identifier, der geändert wurde und so den Apply-Handler auslöste.
Die Abfrage erfolgt mit dem logischen Operator AND, siehe Beispiel

Achtung! Der Parameter numSelections ist hier bedeutungslos.

Hinweis: Es ist möglich den ApplyHandler der OptionGroup manuell (vom BASIC-Code aus) zu aktivieren. Dazu wird die von der GenericClass geerbte Methode **Apply** verwendet. Da ApplyHandler nur ausgelöst werden, wenn das Objekt "modified" ist, muss es vorher als "modified" markiert werden. Alternativ könnte man dem Objekt auch den Hint *ApplyEvenIfNotModifie*d geben.

Beispiel:

```
  ListOfOptions.modified = TRUE
  ListOfOptions.Apply
```

Eine ausführliche Beschreibung dazu finden Sie im Kapitel 3.4 (Die "Apply"-Message) dieses Handbuchs.

MakeVisible

MakeVisible sorgt dafür, dass ein bestimmtes Option-Objekt für den Nutzer sichtbar wird. Diese Methode ist für scrollbare Listen (look = LOOK_SCROLLABLE) sinnvoll.

Syntax im Basic-Code: **<obj>.MakeVisible n**
<obj>: OptionGroup-Objekt
n: Identifier des gewünschten Option-Objekts

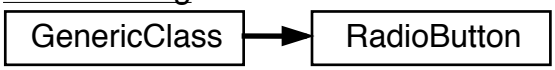
StatusHandler, SendStatus

OptionGroups unterstützen den sogenannten Delayed Mode. Er ist weiter unten, im Kapitel 4.8.5 beschrieben.

4.8.3 RadioButton und RadioButtonGroup

RadioButton

Abstammung



Spezielle Instance-Variablen

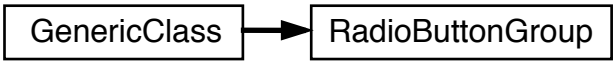
Instance-Variable	Syntax im UI-Code	Im BASIC-Code
identifier	identifier = **numWert**	lesen, schreiben

identifier

RadioButtons haben einen **identifier**, der sie identifiziert. Es ist vom Typ WORD und muss innerhalb einer RadioButtonGroup eindeutig sein. 65535 (alle Bits im WORD gesetzt) ist nicht zulässig, auch wenn es keine sofortige Fehlermeldung gibt. Dieser Wert wird verwendet wenn kein RadioButton-Objekt selektiert ist / werden soll. Für ihn gibt es die Konstante NONE_SELECTED.
RadioButton-Objekte müssen Children einer RadioButtonGroup sein.

RadioButtonGroup

Abstammung



Eine RadioButtonGroup managed eine Liste von RadioButton-Objekten (theoretisch bis über 65000). RadioButtonGroup-Objekte können nur RadioButton-Objekte als Children haben.

Beispiel:

```
RadioButtonGroup Itemgroup
  Caption$= "Select an Item"
  justifyCaption = J_TOP
  Children = item1, item2, item3
  OrientChildren = ORIENT_VERTICALLY
  ApplyHandler = ItemApply
  selection = 2
  End Object

RadioButton item1: Caption$="Item 1":identifier = 1: End Object
RadioButton item2: Caption$="Item 2":identifier = 2: End Object
RadioButton item3
  Caption$="Item 3"
  identifier = 3
  End Object
```

Die UI-Anweisungen für die Objekte item1 und item2 wurden zur Demonstration in einer Zeile untergebracht. Dazu wird - wie im BASIC-Code auch - ein Doppelpunkt zur Trennung verwendet.

Spezielle Instance-Variablen

Instance-Variable	Syntax im UI-Code	Im BASIC-Code
behavior	behavior = **numWert**	lesen, schreiben
look	look = **numWert**	lesen, schreiben
selection	selection = **numWert**	lesen, schreiben
isSelected	—	nur lesen
numSelections	—	nur lesen
DisplayCurrentSelection	DisplayCurrentSelection	—
modified	modified = **numWert**	lesen, schreiben
ModifiedOnRedundantSelection	ModifiedOnRedundantSelection	—
ApplyHandler	ApplyHandler = **<Handler>**	nur schreiben
DoublePressHandler	DoublePressHandler = **<Handler>**	nur schreiben
StatusHandler	StatusHandler = **<Handler>**	nur schreiben

Methoden:

Methode	Aufgabe
MakeVisible n	Einen bestimmten Eintrag sichtbar machen
SelectItem n [,FALSE]	Selektiert-Status eines Eintrags ändern
SendStatus	Status-Handler aufrufen

behavior

Die Instance-Variable **behavior** bestimmt, wie sich die Group bezüglich Selektionsmöglichkeiten der Einträge verhält.

Konstante	Wert	Bedeutung
LB_EXCLUSIVE	0	Das ist der Default-Wert. Nur genau ein Element kann selektiert sein
LB_EXCLUSIVE_NONE	1	Ein oder kein Element kann selektiert sein. Ist kein Element selektiert, wird als "selection" 65535 (NONE_SELECTED) geliefert.
LB_EXTENDED_SELECTION	2	Wie LB_EXCLUSIVE_NONE, aber der Nutzer kann die Selektion durch Ziehen mit der Maus oder durch Shift-Klick oder Strg-Klick "erweitern". Es können also mehrere Elemente selektiert sein.
LB_NON_EXCLUSIVE	3	Jedes Element kann unabhängig von den anderen einzeln selektiert werden.

look

Instance-Variable **look** bestimmt das Aussehen der RadioButtonGroup und ihrer RadioButton-Objekte (engl. look : Aussehen, äußere Erscheinung). Funktionell sind alle Looks identisch.

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

Für alle Listen-Objekte stehen die bei der OptionGroup beschriebenen Looks (LOOK_NORMAL, LOOK_SCROLLABLE, LOOK_MINIMIZE_SIZE und LOOK_TOOLBOX) zur Verfügung. Dort finden Sie auch Bilder und weitergehende Informationen dazu.

selection

Die Instance-Variable **selection** enthält den Identifier des aktuell selektierten RadioButton-Objekts. Ist kein Objekt selektiert, enthält sie den Wert 65535 (NONE_SELECTED). Falls mehrere Objekte selektiert sind, enthält **selection** den Identifier eines der selektierten Objekte.

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

Konstante	Wert	Bedeutung
NONE_SELECTED	65535	Spezialwert für die Instance-Variable "selection" wenn kein Eintrag selektiert ist oder kein Eintrag selektiert werden soll. **Behavior** sollte den Wert LB_EXTENDED_SELECTION oder LB_EXCLUSIVE_NONE haben.

isSelected

IsSelected prüft, ob ein bestimmtes RadioButton-Objekt bzw. Listeneintrag, gegeben durch seinen Identifier, selektiert ist oder nicht.
Siehe auch: SelectItem

Syntax	Lesen:	**<numVar>** = **<obj>** . **isSelected** (**n**)
		<obj> RadioButtonGroup Objekt
		n: Identifier eines RadioButton-Objekts bzw. Listeneintrags

numSelections

Die Instance-Variable **numSelections** enthält die Anzahl der aktuell selektierten Listeneinträge.

Syntax Lesen: **<numVar>** = **<obj>** . **numSelections**

DisplayCurrentSelection

Der Hint **DisplayCurrentSelection** beeinflusst RadioButtonGroups mit **look** = LOOK_MINIMIZE_SIZE und bewirkt, dass im minimierten Zustand statt des Caption-Textes der RadioButtonGroup der Text des aktuell selektierten RadioButtons angezeigt wird.

Syntax UI- Code: **DisplayCurrentSelection**

```
RadioButtonGroup Itemgroup
  Caption$= "Select an Item"
  Children = item1, item2, item3
  look = LOOK_MINIMIZE_SIZE
  DisplayCurrentSelection
  fixedSize = 15 + ST_AVG_CHAR_WIDTH, 1 + ST_LINES_OF_TEXT
  ApplyHandler = ItemApply
  selection = 2
  End Object
```

Das Bild zeigt die RadioButtonGroup entsprechend dem obigen Code, links ohne und rechts mit dem Hint **DisplayCurrentSelection**.

modified

Die Instance-Variable **modified** der RadioButtonGroup enthält die Information, ob die Selektion der RadioButtonGroup seit der letzten Apply-Aktion geändert wurde.

Beachten Sie, dass ein Verändern des Objekts vom BASIC-Code aus (z.B. Belegen der Instance-Variable **selection**), das Objekt nicht als "modified" markiert, d.h. der Wert der Instance-Variablen **modified** wird nicht verändert. Sie können dies bei Bedarf selbst machen, indem Sie die Anweisung "<obj>.modified = TRUE" verwenden.

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

Wenn Sie die Instance Variable **modified** lesen, werden Sie feststellen, dass sie Null enthält, es sei denn, Sie haben sie explizit auf einen anderen Wert gesetzt. Ändert der Nutzer nämlich Auswahl innerhalb der Liste, so passiert intern folgendes:

- Die Instance-Variable **modified** wird mit TRUE belegt.
- Es wird geprüft ob ein ApplyHandler vorhanden ist und dieser wird ggf. aufgerufen. Der Wert von **modified** wird dem Handler übergeben.
- Die Instance-Variable **modified** wird zurückgesetzt (mit FALSE belegt).

Hinweis: Im sogenannten Delayed Mode (siehe entsprechendes Kapitel weiter unten) werden die letzten beiden Schritte nicht ausgeführt, so dass die Instance-Variable **modified** eine eigene Bedeutung erhält.

ModifiedOnRedundantSelection

Der Hint **ModifiedOnRedundantSelection** bewirkt, das eine RadioButtonGroup sich selbst als modified markiert, wenn der Nutzer den bereits selektierten Eintrag erneut auswählt. Im Wesentlichen bedeutet dass, dass die RadioButtonGroup ihre Apply Message aussendet wenn der Nutzer einen bereits selektierten Eintrag erneut selektiert.

Syntax UI-Code: **ModifiedOnRedundantSelection**

ApplyHandler

Der **ApplyHandler** der RadioButtonGroup wird aufgerufen, wenn der User ein Element selektiert / die Selektion ändert. Er muss als **ListAction** deklariert sein.

Beispiel, passend zum UI-Code oben:

```
ListAction   ItemApply
  Print selection; numSelections
  END Action
```

Der Parameter **selection** enthält den Identifier, der geändert wurde und so den Apply-Handler auslöste.
Der Parameter **numSelections** enthält die Anzahl der der selektierten Elemente.
Der Parameter **modified** ist immer 65535, d.h. alle Bits des WORD sind gesetzt. Das ist das Analogon zur Integer-Konstanten TRUE (= –1).

Syntax	UI- Code:	**ApplyHandler** = **<Handler>**
	Schreiben:	**<obj>**.**ApplyHandler** = **<Handler>**

Hinweis: Es ist möglich den ApplyHandler der RadioButtonGroup manuell (vom BASIC-Code aus) zu aktivieren. Dazu wird die von der GenericClass geerbte Methode Apply verwendet. Da ApplyHandler nur ausgelöst werden, wenn das Objekt "modified" ist, muss es vorher als "modified" markiert werden. Alternativ könnte man dem Objekt auch den Hint ApplyEvenIfNotModified geben.

Beispiel:

```
Itemgroup.modified = TRUE
Itemgroup.Apply
```

Eine ausführliche Beschreibung dazu finden Sie im Kapitel 3.4 (Die "Apply"-Message) dieses Handbuchs.

DoublePressHandler

Der **DoublePressHandler** wird aufgerufen, wenn der User ein Element mit der Maus doppelklickt. War der Eintrag bis dahin noch nicht selektiert wird vorher der **ApplyHandler** aufgerufen. Die Parameter entsprechen denen des ApplyHandlers.

Syntax	UI- Code:	**ApplyHandler** = **<Handler>**
	Schreiben:	**<obj>**.**ApplyHandler** = **<Handler>**

Hinweis: GEOS unterstützt einen DoublePressHandler nur, wenn die Instance-Variable behavior auf LB_EXCLUSIVE oder LB_EXTENDED_SELECTION gesetzt ist. R-BASIC kann nichts dafür - sorry.

MakeVisible

MakeVisible sorgt dafür, dass ein bestimmtes RadioButton-Objekt bzw. ein bestimmter Listeneintrag einer DynamicList für den Nutzer sichtbar wird. Diese Methode ist für scrollbare Listen (look = LOOK_SCROLLABLE, z.B. DynamicList Objekte) sinnvoll.

Syntax im Basic-Code:		**<obj>**.**MakeVisible n**
	<obj>:	RadioButtonGroup-Objekt
	n:	Identifier des gewünschten RadioButton-Objekts bzw. Listeneintrags

SelectItem

Mit SelectItem können Sie den Status "Selektiert" oder "nicht selektiert" für ein bestimmtes RadioButton-Objekt bzw. einen bestimmten Listeneintrag einer DynamicList ändern. Diese Methode ist für Listen, die eine Mehrfachauswahl erlauben, sinnvoll (behavior = LB_EXCLUSIVE_NONE, LB_EXTENDED _SELECTION oder LB_NON_EXCLUSIVE). Für Standard-Listen (behavior = LB_EXCLUSIVE) sollten Sie die Instancevariable selection benutzen.

Syntax im Basic-Code: **<obj>.SelectItem n** [, state]

<obj>:	RadioButtonGroup-Objekt
n:	Identifier des gewünschten RadioButton-Objekts bzw. Listeneintrags
state:	TRUE (Default): Eintrag selektieren FALSE: Eintrag auf "nicht selektiert" setzen

StatusHandler, SendStatus

RadioButtonGroups unterstützen den sogenannten Delayed Mode. Er ist weiter unten, im Kapitel 4.8.5 beschrieben.

4.8.4 DynamicList

Abstammung

Eine DynamicList ist eine erweiterte RadioButtonGroup. Daher erbt sie alle Instance-Variablen und Fähigkeiten dieser Klasse (behavior, look, selection, modified, ApplyHandler, DoublePressHandler, SelectItem, Arbeit im Delayed Mode usw.). Zusätzlich hat sie folgende Besonderheiten:

- Die Instance-Variable **look** steht per Default auf LOOK_SCROLLABLE. Sie können das natürlich im UI-Code ändern.
- Eine DynamicList hat im UI-Code **keine Children**. Sie erzeugt und verwaltet ihren Children (Listeneinträge) selbst.
- Sie **müssen** der Instance-Variable **numItems** auf eine Wert ungleich Null setzen und einen **QueryHandler** für eine DynamicList schreiben, sonst werden keine Listeneinträge angezeigt.

Spezielle Instance-Variablen

Instance-Variable	Syntax im UI-Code	Im BASIC-Code
numItems	numItems = **numWert**	lesen, schreiben
QueryHandler	QueryHandler = **<Handler>**	nur schreiben

Methoden:

Methode	Aufgabe
ItemText$	Listeneintrag einen Text zuweisen
ItemGString	Listeneintrag eine Grafik zuweisen
InsertItems	Eine bestimmte Anzahl von Einträgen hinzufügen
RemoveItems	Eine bestimmte Anzahl von Einträgen entfernen

So arbeitet eine DynamicList

Nehmen wir an, wie haben eine DynamicList mit 5 Einträgen, die hier als Items bezeichnet werden, so wie im Code-Beispiel dargestellt.

```
DynamicList DynList
  Caption$ = "List Example"
  justifyCaption = J_TOP
  numItems = 5
  fixedSize = 15 + ST_AVG_CHAR_WIDTH, \
                  3 + ST_LINES_OF_TEXT, 3
  selection = 3
  ApplyHandler = MyApplyHandler
  QueryHandler = MyListQuery
  END Object
```

Die Instance-Variable **numItems** bestimmt, wie viele Items eine DynamicList hat. Die Items sind selbst Objekte, sie werden aber im UI-Code nicht aufgeführt, sondern die DynamicList erzeugt sie bei Bedarf selbst. Die von der RadioButtonGroup-Klasse geerbte Instance-Variable **selection** legt fest, welcher Eintrag am Anfang selektiert ist. Ebenfalls von der RadioButtonGroup Klasse geerbt ist die Instance-Variable ApplyHandler. **FixedSize** hingegen ist von der GenericClass geerbt und legt im Beispiel fest, dass die Liste 15 Zeichen breit und 3 Zeilen hoch ist, wobei 3 Items gleichzeitig angezeigt werden sollen.

Will eine DynamicList eines ihrer Items darstellen, so benötigt sie eine Information, welchen Text das Item darzustellen hat. Dazu ruft sie den **QueryHandler** auf (engl. to query: anfordern). Diesem wird die Nummer des Items, das dargestellt werden soll, übergeben. Die Zählung beginnt dabei immer mit Null. Der Handler muss den anzuzeigenden Text (bzw. die Grafik) ermitteln und ihn an die DynamicList übergeben, wie im Beispielcode dargestellt.

```
LISTACTION MyListQuery
DIM name$   AS String

  ON selection SWITCH
  CASE 0: name$ = "Ralph"  : END CASE
  CASE 1: name$ = "Fred"   : END CASE
  CASE 2: name$ = "Joshua" : END CASE
  CASE 3: name$ = "Mary"   : END CASE
  CASE 4: name$ = "Antoinette": END CASE
  DEFAULT:  name$ = "no name"              'Nur zur Sicherheit!
  END SWITCH

  sender.ItemText$(selection) = name$

  END Action
```

numItems

Die Instance-Variable **numItems** bestimmt, wie viele Listeneinträge die Dynamic-List hat. Sie kann im sowohl UI-Code als auch im BASIC-Code gesetzt werden.

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

Es ist explizit zulässig, **numItems** im UI-Code nicht oder mit Null zu belegen, was die gleiche Wirkung hat. Die Liste bleibt dann zunächst leer und sendet keine Query-Message aus. Es ist daher eine gute Idee, **numItems** am Programmende auf Null zu setzen.

Tipp: Der Ändern Sie den Wert von **numItems** im BASIC-Code, so stellt sich die Liste neu dar. Wenn Sie also die Anzahl der Einträge oder deren Inhalt in der Liste zur Laufzeit ändern wollen, so müssen Sie nur **numItems** einen Wert zuweisen. Die Liste fordert dann über ihren Queryhandler alle Einträge neu an.
Alternativ können Sie die Methoden InsertItems bzw. RemoveItems verwenden.

ItemText$

Die Methode **ItemText$**(index) kann nur im BASIC-Code aufgerufen werden. Das passiert üblicherweise im QueryHandler der DynamicList. Über **ItemText$**(index) wird der Liste mitgeteilt, welchen Text das entsprechende Item darzustellen hat. "Index" bestimmt, welchem Item der Text zugeordnet wird.

Syntax Schreiben:	**<obj>.ItemText$(index)** = **"Text"**

ItemGString

Die Methode **ItemGString**(index) kann nur im BASIC-Code aufgerufen werden. Das passiert üblicherweise im QueryHandler der DynamicList. Über **ItemGString**(index) wird der Liste mitgeteilt eine Grafik oder einen formatierten Text für das entsprechende Item darzustellen. "Index" bestimmt, welchem Item die Grafik zugeordnet wird.

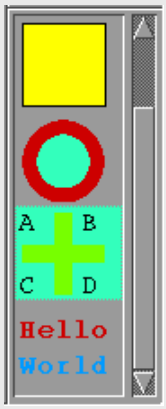
Syntax Schreiben:	**<obj>.ItemGString** (**index**) = **<gsHan>** **<gsHan>**: Handle eines GStrings

Wenn Sie Grafiken in DynamicList-Objekten darstellen wollen sollten Sie Breite und Höhe der Listen mit Hilfe des Hints fixedSize festlegen. Die ersten beiden Parameter dieses Hints bestimmen Breite und Höhe der **gesamten Liste**, der dritte Parameter bestimmt die Anzahl der gleichzeitig dargestellten Items. Folglich ist die Höhe eines einzelnen Listeneintrags der Quotient aus Höhe der Liste und Anzahl der Einträge.

```
DynamicList  MyList
  QueryHandler = QueryGraphic
  selection = 0

  ' Die Liste soll 4 Grafiken mit
  ' je 50x50 Pixel gleichzeitig darstellen
  ' Weil sie selbst etwas Platz braucht machen
  ' wir sie etwas größer
  ' fixedSize = sizeX, sizeY, numElements
  fixedSize = 60, 205, 4

  ' Insgesamt sollen es 5 Elemente sein.
  numItems = 5

End Object
```

Der QueryHandler erzeugt verschieden grafische Symbole, die als Listeneintrag dargestellt werden.

```
LISTACTION QueryGraphic
dim gsHan as HANDLE

  ' Wir starten die Aufzeichnung eines GString
  ' Alle Grafik- und Textausgaben gehen
  ' jetzt in den GString

  gsHan = StartRecordGS()
  on selection SWITCH
  case 0:
    FillEllipse 2, 2, 48, 48, blue
    end case
  case 1:
    FillRect  2, 2, 48, 48, yellow
    Rectangle 2, 2, 48, 48, black
    end case
  case 2:
    FillEllipse 2, 2, 48, 48, red
    FillEllipse 10, 10, 40, 40, cyan
    end case
  case 3:
    FillRect 2, 20, 48, 30, light_green
    FillRect 20, 2, 30, 48, light_green
    PRINT atxy 0,0;"A"
    PRINT atxy 35,0;"B"
    PRINT atxy 0,35;"C"
    PRINT atxy 35,35;"D"
    end case
  case 4:
    printfont.style = ts_bold
    PRINT atxy 0,0;ink red;"Hello"
    PRINT atxy 0, 20;ink blue;"World"
    end case
  end switch

  ' EndRecordGS beendet den Aufzeichnusmodus.
  EndRecordGS gsHan

  ' Wir übergeben die Grafik an den Listeneintrag
  sender.ItemGString(selection) = gsHan

  ' Ganz wichtig:
  ' Das Handle wieder freigeben, sonst frisst
  ' das Programm Systemhandles!
  FreeGS gsHan

  END Action
```



Tipp: Möchten Sie die grafischen Einträge nebeneinander darstellen, so verwenden Sie orientChildren = ORIENT_HORIZONTALLY. Die Liste muss dann

natürlich 4 Einträge breit und nur einen Eintrag hoch sein. Den kompletten Code dazu finden Sie bei den Beispielen zu den Listenobjekten.

```
DynamicList  MyList
  QueryHandler = QueryGraphic
  selection = 0
  orientChildren = ORIENT_HORIZONTALLY
  fixedSize = 205, 60, 4
  numItems = 5
End Object
```

Achtung! DynamicList Objekte verwalten ihre Einträge als Children selbst. Auch die Captions der Listeneinträge werden in dem Objektblock gespeichert, in dem die Liste ist. Im Normalfall befinden sich auch noch andere Objekt in diesem Block, so dass er anfangs bereits ca. 4 kByte groß ist. Damit ist ein gewisser Platzbedarf für die Listeneinträge bereits berücksichtigt. Da man mit **ItemGString** aber auch relativ große Grafiken zuweisen kann (z.B. wenn der GString eine Bitmap enthält), kann der Objektblock zur Laufzeit trotzdem sehr groß werden und der Speichermanager bekommt ein Problem ("Hauptspeicher voll"). Um dieses Problem zu vermeiden sollte man Listen, die viele oder große grafische Elemente darstellen sollen, in einem eigenen Objektblock ablegen. Dazu verwendet man die UI-Anweisung ForceNewObjBlock (siehe Kapitel 2.1.4). Diese weist den Compiler an unverzüglich mit einem neuen Objektblock zu beginnen, auch wenn im aktuell aufgebauten Block noch Platz zu sein scheint.
Weitere Informationen zu GStrings und auch zum Speicherbedarf vom GStrings finden Sie im R-BASIC Programmierhandbuch, Kapitel 2.8.5 (Arbeit mit Graphic Strings).

```
ForceNewObjBlock

DynamicList MyBigList
  ....
  END Object

ForceNewObjBlock
```

QueryHandler

Der **QueryHandler** wird automatisch aufgerufen, wenn die DynamicList eines seiner Items darstellen will. QueryHandler müssen als **ListAction** deklariert sein. Der Parameter **selection** enthält die Nummer des Items, für das ein Text angefordert wird. Die anderen Parameter sind bedeutungslos.
Die korrekte Reaktion des QueryHandlers ist, wie in den Code-Beispielen oben, die Methoden **ItemText$**(selection) oder **ItemGString**(index) der DynamicList aufzurufen.

Syntax	UI- Code:	**QueryHandler** = **<Handler>**
	Schreiben:	**<obj>**.**QueryHandler** = **<Handler>**

Das Zusammenspiel zwischen DynamicList und QueryHandler funktioniert automatisch, so dass Sie sich nicht weiter darum kümmern müssen. Sie müssen nur sicherstellen, dass der QueryHandler zu jedem selection-Wert, der von der Liste kommen kann (Null ... numItems–1), einen passenden Text oder eine Grafik bereitstellt. Dabei ist es, wie im Beispiel-Code zu ItemText$ oben, sinnvoll auch "unerwartete" Fälle zu berücksichtigen, falls sie später etwas ändern oder ein Programmierfehler auftritt.

InsertItems

InsertItems fügt eine bestimmte Anzahl von Listeneinträgen ab einer wählbaren Position in die Liste ein. Die aktuell selektierten Einträge bleiben selektiert.

Syntax im BASIC-Code: **<obj>.InsertItems pos, anz**
pos: Position, aber der eingefügt werden soll
Null: Einfügen am Anfang
pos > aktuelle Anzahl: Anhängen
anz: Anzahl der neuen Einträge

RemoveItems

RemoveItems löscht eine bestimmte Anzahl von Listeneinträgen ab einer wählbaren Position. Die aktuell selektierten Einträge bleiben selektiert, falls sie nicht gelöscht wurden.

Syntax im BASIC-Code: **<obj>.InsertItems pos, anz**
pos: Position, aber der gelöscht werden soll
anz: Anzahl der zu löschenden Einträge

4.8.5 Listen-Objekte im Delayed Mode

Alle Listen-Objekte können im "Delayed Mode" (engl.: verzögerter Modus) arbeiten. Dazu muss man dem Objekt selbst bzw. einem seiner Parents im UI-Code den Hint **MakeDelayedApply** geben oder man bindet das Objekt als Child in einem Dialog ein, dessen **dialogType** Instance Variable auf DT_DELAYED_APPLY gesetzt ist. Dieser "Delayed Mode" ist ausführlich im Kapitel 3.4.2 (Delayed Mode und Status-Message) dieses Handbuchs beschrieben, eine Beschreibung des Dialog-Objekts im Delayed Mode finden Sie im Kapitel 4.6.6.5.

Instance Variable	Syntax im UI-Code	Im BASIC-Code
StatusHandler	StatusHandler = **<Handler>**	nur schreiben

Syntax UI- Code: **StatusHandler** = **<Handler>**
Schreiben: **<obj>.StatusHandler** = **<Handler>**

Der **StatusHandler** wird im Delayed Mode statt des ApplyHandlers gerufen, wenn der Nutzer die Auswahl innerhalb der Liste ändert. Der ApplyHandler hingegen wird erst auf Anforderung gerufen (siehe Kapitel 3.4.2).

Die Instance-Variable **modified** kann einen Wert ungleich Null enthalten, nämlich dann, wenn das Objekt vom User modifiziert wurde, der ApplyHandler aber noch nicht gerufen wurde. Der Aufruf des ApplyHandlers setzt auch im Delayed Mode den modified-Status zurück. Falls kein ApplyHandler gesetzt ist wird der modified-Status wird immer dann zurückgesetzt, wenn der ApplyHandler gerufen werden müsste.

Bei einer OptionGroup gilt außerdem:
- Ändert der Nutzer den Zustand eines Option-Objekts der OptionGroup, so wird die Instance-Variable **modified** mit dessen identifier logisch OR verknüpft.
- Der Parameter "modified" enthält folglich sowohl beim StatusHandler als auch beim ApplyHandler die logische OR-Verknüpfung der Identifier der seit dem letzten Aufruf des ApplyHandler veränderten Option-Objekte.

Methode	Aufgabe
SendStatus	Status-Handler aufrufen

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

Die Methode **SendStatus** fordert das Objekt auf, seinen StatusHandler aufzurufen (d.h. seine Status-Message zu senden).

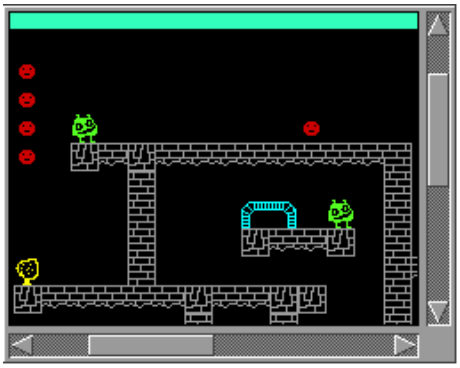
(Leerseite)

4.9 View und Content

4.9.1 Überblick

Die View Objektklasse stellt ein "Fenster" bereit, in dem beliebige grafische Daten - einschließlich Texte - dargestellt werden können. Die Inhalte dieses Fensters, d.h. die grafischen Daten, werden vom "Content"-Objekt (content: engl. Inhalt) bereitgestellt.

Dieses Objekt muss nur die Daten darstellen können. Alles andere, wie Scrolling, Zoom oder Clipping (engl. to clip: beschneiden) macht das View. Das View fordert bei Bedarf das Content-Objekt auf, sich selbst darzustellen, aber welcher Teil der Darstellung auf dem Bildschirm erscheint (Clipping), ob er vergrößert oder verkleinert ist usw., darum kümmert sich das View.



Es gibt drei Objektklassen, die als Content für ein View dienen können: Die Klasse "VisContent", die beliebige Daten darstellen kann, die Klasse "BitmapContent", die eine editierbare Bitmap bereitstellt und die Klasse "GenContent", die andere GenericClass Objekte beinhaltet. Eine Übersicht über diese Klassen finden Sie weiter unten. Die GenContent Klasse ist dort ausführlich beschrieben. Bei VisContent und BitmapContent handelt es sich um VisualClass Objekte. Sie sind deshalb im Kapitel 5 des Objekthandbuchs beschrieben.

Häufig ist es so, dass der Nutzer im "Ansicht"-Menü festlegen kann, ob das Dokument in Originalgröße, vergrößert oder verkleinert dargestellt werden soll. Für diesen Zweck gibt es das "ViewControl" Objekt, das zum einen die erforderliche UI bereitstellt und zum anderen automatisch im Hintergrund mit dem View-Objekt zusammenarbeitet. Die Aufgabe des Programmierers ist es lediglich die Objekte zu deklarieren und mit wenigen Anweisungen zu konfigurieren.

Die Kombination der View/Content ist sehr universell und für praktisch alle Anwendungsgebiete geeignet. Das Ausnutzen dieser Möglichkeiten erfordert daher eine gewisse Einarbeitung in die Eigenschaften der beiden Objektklassen und ihrem Zusammenspiel, wobei man um gelegentliches Experimentieren nicht herumkommt. Die Verwendung der Klasse BitmapContent stellt bereits einen Spezialfall dar, der in der Grundkonfiguration sehr einfach zu handhaben ist.
Es gibt in R-BASIC weitere, spezialisierte und damit noch einfacher zu handhabende Möglichkeiten, Grafiken darzustellen. Einen Überblick über diese Möglichkeiten, die ohne die Verwendung eines View-Objekts auskommen, und Verweise zu den entsprechenden Abschnitten in diesem Handbuch finden Sie im Kapitel 2.2.

Mausunterstützung

Das Viewobjekt leitet die Mausereignisse direkt an sein Content-Objekt weiter. Bei Bedarf können Sie Maushandler für das Content-Objekt bzw. seine Children schreiben.

Tastaturhandling

Im Normalfall behandelt das View-Objekt die Tastaturereignisse selbständig, indem es sie an sein Content-Objekt weiterreicht. Sie können sich aber in das Tastaturhandling eines View-Objekts einklinken, indem Sie einen Tastaturhandler schreiben. Zum Beispiel könnten Sie die Tastenkombination Strg-'+' benutzen, um den Zoomfaktor des Views zu vergrößern. Eine ausführliche Beschreibung, wie man einen Tastaturhandler schreibt und was es dabei zu beachten gilt, finden Sie im Handbuch "Spezielle Themen", Kapitel 14.
Es ist sehr selten, dass man einen Tastaturhandler für ein Viewobjekt benötigt. In den meisten Fällen wird die Tastatur vom Content bzw. seinen Children behandelt.

Wichtig: Das View-Objekt gibt die Tastaturereignisse zuerst an sein Content weiter, bevor es den BASIC Tastaturhandler aufruft. Sollten beide Objekte (View und Content) einen Tastaturhandler haben wird daher zuerst der Tastaturhandler des Content-Objekts aufgerufen und erst danach der Tastaturhandler des View-Objekts. Im Kapitel 14.4 des Handbuchs "Spezielle Themen" ist am Beispiel eines Textobjekts beschrieben, wie man vorgehen muss, um den BASIC-Tastaturhandler aufzurufen, bevor das Objekt das Tastaturereignis an sein Content weitergibt.

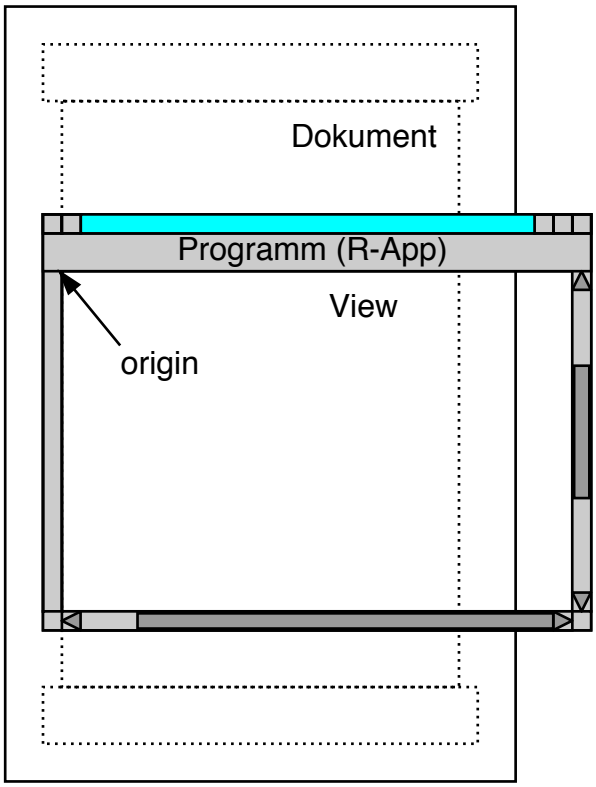
Focus und Target

Das View-Objekt ist ein Knoten in der Focus- und Target-Hierarchie. Es ist möglich zu überwachen, ob ein View-Objekt den Focus oder das Target hat, indem man einen Focus- bzw. Target-Handler schreibt. Die notwendigen Details zur Arbeit mit Focus und Target finden Sie im Kapitel 12 (Focus und Target) des Handbuchs "Spezielle Themen". Das Arbeiten mit Focus und Target ist etwas für erfahrene Programmierer und im Zusammenhang mit einer View/Content Kombination nur in wenigen Fällen notwendig.

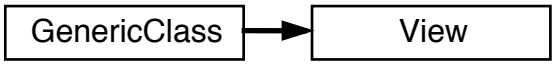
4.9.2 Das View

Das View-Objekt ist die Schnittstelle zwischen ihrer Programm-UI und den darzustellenden Inhalten. Es arbeitet in vielen Situationen automatisch mit seinem Content-Objekt zusammen und sorgt bei Bedarf für das Scrolling, Clipping und den Zoom.
Häufig entsprechen die darzustellenden Daten (Text, Grafik ...) einem Dokument. In GeoDraw sind dies Grafiken, in GeoWrite Texte und in R-BASIC der Quellcode eines Programms. Der Einfachheit halber sprechen wir im Folgenden auch dann von "Dokument" wenn das Programm keine Dokument-Dateien hat. In diesem Sinne ist in einem Spiel das Spielfeld das darzustellende "Dokument".

Im Kern ist es so, dass das View einen bestimmten Ausschnitt aus dem Dokument darstellt. Welcher Bereich das ist und ob er vergrößert, verkleinert oder in Originalgröße dargestellt wird, das bestimmt entweder der Nutzer über das Ansicht-Menü, (das ein ViewControl-Objekt enthält) bzw. über die Rollbalken des View-Objekts oder der Programmierer legt dies über Programmbefehle fest.
Die Position, die in der linken oberen Ecke des View dargestellt wird heißt "origin" (engl.: Ursprung).



Abstammung:



Spezielle Instance-Variablen:

Variable	Syntax im UI-Code	Im BASIC-Code
Content	Content = **<obj>**	lesen, schreiben
contentSize	contentSize = **sizeX**, **sizeY**	lesen, schreiben
hControl	hControl = **numWert**	lesen, schreiben
vControl	vControl = **numWert**	lesen, schreiben
HideScrollersWhenNotScrollable	HideScrollersWhenNotScrollable	—
viewColor	viewColor = **numWert**	lesen, schreiben
viewAttrs	viewAttrs = **bitsToSet**,**bitsToClear**	lesen, schreiben
scale	scale = **xWert** , **yWert**	lesen, schreiben
scaleToFitOptions	scaleToFitOptions = **numWert**	lesen, schreiben
origin	origin = **xWert** , **yWert**	lesen, schreiben
viewIncrement	viewIncrement = **xWert** , **yWert**	lesen, schreiben
suspendUpdate	—	lesen, schreiben
DoNotWinScroll	DoNotWinScroll	—
ImmediateDragUpdates	ImmediateDragUpdates	—
DelayedDragUpdates	DelayedDragUpdates	—
hLink	hLink = **<obj>**	lesen, schreiben
vLink	vLink = **<obj>**	lesen, schreiben
inputOptions	inputOptions = **numWert**	lesen, schreiben
focusable	focusable = **FALSE** I **TRUE**	lesen, schreiben
holdsLargeText	holdsLargeText = **TRUE**	lesen, schreiben

Methoden:

Methode	Aufgabe
Redraw	View und Content neu zeichnen
GetVisibleRect	Aktuell dargestellten Bereich holen
ScrollToOffset	View um einen bestimmten Betrag scrollen
ScrollCmd	View Scroll-Kommando ausführen
InitiateDragScroll	Drag-Scrolling in Nicht-Standard-Situationen aktivieren
SetDragBounds	Begrenzung für Drag-Scrolling setzen
SetPointerImage	Mauszeiger ändern (aus DATA-Zeilen lesen)
ClearPointerImage	Mauszeiger wieder auf Standard zurücksetzen

4.9.2.1 Das Content eines View

Jedes View benötigt genau ein Content-Objekt, dass die Daten des Dokuments darstellt. Wie bereits oben erwähnt verwenden wir den Begriff Dokument auch dann, wenn die darzustellenden Daten zu keiner Dokument-Datei im engeren Sine gehören. Die Instancevariable **Content** enthält das aktuelle Content-Objekt. Häufig ist das gleichzeitig der Screen zur Ausgabe von Grafik und Text. Das muss jedoch nicht sein. Die Instancevariable **contentSize** enthält die Größe des Content-Objekts, also die Größe des vom View darzustellenden Dokuments. Mit der Methode **Redraw** können Sie ein Neuzeichnen des Views und seines Content auslösen.

Content

Die Instance-Variable **Content** (engl.: Inhalt) enthält das aktuelle Content-Objekt des View's. Es kann im BASIC-Code gelesen und geschrieben werden. Wird dem View ein neues Content-Objekt zugeordnet stellt sich das View automatisch neu dar, so dass der neue Content sichtbar wird. Dabei kommuniziert das View mit dem neuen Content und kann, je nachdem, welche Werte für hControl, vControl und viewAttrs gesetzt sind, gegebenenfalls seine Größe neu bestimmen.

Ein eventuell vorher mit dem View verbundenes Content-Objekt wird dabei automatisch abgekoppelt. Die Zuweisung eines "leeren" Content-Objekts mit der Funktion **NullObj()** ist zulässig.

Syntax UI-Code: **Content** = **<obj>**
<obj> muss namentlich aufgeführt werden.
Variablen sind im UI-Code nicht zulässig.
Lesen: **<objVar>** = **<obj>**.**Content**
Schreiben: **<obj>**.**Content** = **<obj2>**

Beachten Sie, dass die Zuweisung eines neuen Content-Objekts den aktuelle "Screen" nicht beeinflusst (siehe auch Kapitel 2.3.1). Wenn das alte Content-Objekt der "Screen" war, bleibt er es auch. Alle Grafik- und Text-Ausgaben gehen weiterhin an dieses Objekt. Möglicherweise müssen Sie also zusätzlich das neue Content-Objekt auch der Screen-Variablen zuweisen.

Beispiel UI-Code:

```
View  MyView
  hControl = HVC_SCROLLABLE
  vControl = HVC_SCROLLABLE
  Content = MyBitmapContent
END Object
```

Beispiele BASIC-Code:

```
DIM ob as OBJECT

ob = MyView.Content                ' Altes Content merken
MyView.Content = MyOtherContent    ' Neues Content-Objekt zuweisen
....
MyView.Content = ob                ' Altes Content wieder zuweisen
```

contentSize

Die Instance-Variable **contentSize** speichert die x- und y-Ausdehnung der vom Content-Objekt darzustellenden Dokuments. Gemeinsam mit seiner eigenen Größe und einem eventuell eingestellten Zoom-Faktor kann das View dann z.B. entscheiden, ob es Scrollbalken verwenden muss und wie groß deren "innerer Balken" zu sein hat.

In vielen Fällen verwaltet das View die **contentSize** automatisch, indem es mit dem Content-Objekt kommuniziert. Bei Bedarf kann **contentSize** aber sowohl im UI-Code als auch im BASIC-Code geschrieben werden.

Syntax UI-Code: **contentSize** = **xSize**, **ySize**
Lesen: **<numVar>** = **<obj>**.**contentSize** (0) ! x-Size
<numVar> = **<obj>**.**contentSize** (1) ! y-Size
Schreiben: **<obj>**.**contentSize** = **xSize**, **ySize**

Beispiel UI-Code:

```
View  MyView
  < .. andere Instance-Variablen hier .. >
  contentSize = 100, 200
  END Objekt
```

Beispiele BASIC-Code:

```
DIM breite  AS  Real

' Ausgabe in der Dokument-Größe in der Form: Breite x Höhe"
breite = MyView.contentSize(0)
Print "Alte Größe = "; breite ; " x "; MyView.contentSize(1);
  "Pixel"

MyView.contentSize = 64, 48   ' Neue Größe zuweisen
```

Redraw

Die Methode Redraw löst ein Neuzeichnen des kompletten View-Objekts und seines Content aus.

Syntax im Basic-Code:	**<viewObj>.Redraw**

Hinweis: Die Redraw-Methode anderer Objekte akzeptiert einen Parameter. Für View-Objekte ist der Parameter syntaktisch möglich, wird jedoch ignoriert.

4.9.2.2 View Geometrie

Da ein View ein generisches Objekt ist können alle Geometrie-Hints aus dem Kapitel 3.3, insbesondere aus dem Kapitel 3.3.4 (Objektgröße) mit einem View verwendet werden. Die wichtigsten sind der Einfachheit halber hier noch einmal aufgeführt. Beachten Sie, dass die ~Size-Hints mit den Werten für hControl bzw. vControl in Konflikt geraten können.

GenericClass Hint	UI-Code Syntax	Im BASIC Code
ExpandWidth	ExpandWidth	—
ExpandHeight	ExpandHeight	—
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

Zusätzlich verfügt das View über eigene Geometriefähigkeiten, die mit der Größe des darzustellenden Dokuments und der Bereitstellung von Rollbalken (Scroller) zusammenhängen.
Die Instancevariablen **hControl** und **vControl** legen fest, wie sich das View in horizontaler (hControl) oder vertikaler (vControl) Richtung darstellt. Mit dem Hint **HideScrollersWhenNotScrollable** kann man bewirken, dass Rollbalken nur dann dargestellt werden, wenn sie wirklich gebraucht werden. Schließlich kann man mit **viewColor** die Hintergrundfarbe des View einstellen.

hControl, vControl

Die Instancevariablen **hControl** und **vControl** legen fest, wie sich das View in horizontaler (hControl) oder vertikaler (vControl) Richtung darstellt. Zulässige Werte sind Kombinationen der HVC_-Konstanten (HVC: horizontal-vertical-control). Die Werte sind so gewählt, dass jede Konstante genau ein Bit gesetzt hat

(Bitflags). Mehrere Konstanten können mit + oder OR verknüpft werden, die Abfrage, ob ein bestimmter Wert gesetzt ist kann mit der logischen AND Funktion erfolgen.

Syntax UI-Code: **hControl** = **numWert**
vControl = **numWert**
numWert ist eine Kombination der HVC_-Konstanten
Lesen: **<numVar>** = **<obj>**.**hControl**
<numVar> = **<obj>**.**vControl**
Schreiben: **<obj>**.**hControl** = **numWert**
<obj>.**vControl** = **numWert**
numWert ist eine Kombination der HVC_-Konstanten

Folgende Konstanten stehen zur Verfügung. Hier nicht aufgeführte Werte (z.B. 2) sollten auch nicht verwendet werden, da ihre Wirkung unbestimmt ist.
Auch die Kombination widersprüchlicher Werte (z.B. HVC_SCROLLABLE + HVC_NO_SCROLLBAR) kann seltsame Folgen haben.

Konstante	Wert dezimal	hex	gesetztes Bit
HVC_SCROLLABLE	128	&H80	7
HVC_TAIL_ORIENTED	32	&H20	5
HVC_NO_SCROLLBAR	16	&H10	4
HVC_NO_LARGER_THAN_CONTENT	8	&H08	3
HVC_NO_SMALLER_THAN_CONTENT	4	&H04	2
HVC_KEEP_ASPECT_RATIO	1	&H01	0

HVC_SCROLLABLE
Das View soll in diese Dimension scrollbar sein. Die Scrollleisten werden auch gezeigt, wenn es eigentlich nicht erforderlich ist.

HVC_TAIL_ORIENTED
Bestimmt, dass das View das untere/rechte Ende des Content-Bereichs weiterhin darstellen soll, wenn dieser Bereich bereits dargestellt wird und sich die Größe des Content-Objekts ändert.

HVC_NO_SCROLLBAR
Das View soll keine Scrollleisten in der entsprechenden Dimension anzeigen, auch wenn es scrollbar ist.

HVC_NO_LARGER_THAN_CONTENT
Das View soll sich in der gegeben Dimension nicht größer als das Content machen, wobei der Wert, der in der **contentSize** Instance-Variable steht, maßgebend ist. Per Default gibt es keine Restriktionen bezüglich der View-Größe.

HVC_NO_SMALLER_THAN_CONTENT

Das View soll sich in der gegeben Dimension nicht kleiner als das Content machen, wobei der Wert, der in der **contentSize** Instance-Variable steht, maßgebend ist. Per Default gibt es keine Restriktionen bezüglich der View-Größe.

HVC_KEEP_ASPECT_RATIO

Das View soll das Seitenverhältnis in der Darstellung beibehalten, indem es seine Größe in der gegebenen Dimension basierend auf der Größe der anderen Dimension berechnet.

Beispiel UI-Code:

```
View  MyView
  hControl = HVC_NO_LARGER_THAN_CONTENT +
             HVC_NO_SMALLER_THAN_CONTENT
  vControl = HVC_SCROLLABLE
  Content = MyBitmapContent
END Object
```

Beispiel: Abfrage mit AND, ob das View scrollbar ist

```
IF   MyView.vControl  AND  HVC_SCROLLABLE  THEN ...
```

HideScrollersWhenNotScrollable

Bewirkt, dass der Rollbalken verschwindet, wenn das View in die zugehörige Richtung nicht scrollen kann, weil bereits das gesamte Dokument angezeigt wird.

Syntax	UI-Code:	**HideScrollersWhenNotScrollable**
	Lesen:	—
	Schreiben:	—

viewColor

Die Instancevariable viewColor bestimmt die Hintergrundfarbe für das View. Per Default ist sie weiß.

Syntax	UI-Code:	**viewColor** = **color**
	Lesen:	**<numVar>** = **<obj>**.**viewColor**
	Schreiben:	**<obj>**.**viewColor** = **color**
	color:	numerischer Farbwert. Index- oder RGB-Farbe

4.9.2.3 Die View Attribute

Die Instancevariable viewAttrs enthält wichtige Konfigurationsdaten für das View aus verschiedenen Bereichen.

viewAttrs

ViewAttrs sind Bitflags, d.h. jedes Bit steht für eine bestimmte Eigenschaft, die einzeln zu- oder abgeschaltet werden kann. Bits, die in der untenstehenden Tabelle nicht aufgeführt sind, sind immer Null.
Per Default ist kein Bit aus der Tabelle gesetzt.

Syntax	UI-Code:	**viewAttrs** = **bitsToSet** , **bitsToClear**
	Lesen:	**<numVar>** = **<obj>**.**viewAttrs** (0) Die BASIC-Syntax erfordert beim Lesen von viewAttrs einen Parameter. Der Wert wird hier ignoriert.
	Schreiben:	**<obj>**.**viewAttrs** = **bitsToSet** , **bitsToClear**
	bitsToSet:	zu setzende Attribute, Bitflags, siehe Tabelle
	bitsToClear:	zu setzende Attribute, Bitflags, siehe Tabelle

Verfügbare Attribute für viewAttrs

Konstante	Wert	Wert hex.
VA_CONTROLLED	32768	&H8000
VA_GENERIC_CONTENTS	16384	&H4000
VA_DRAG_SCROLLING	4096	&H1000
VA_NO_WIN_FRAME	2048	&H800
VA_SAME_COLOR_AS_PARENT_WIN	1024	&H400
VA_VIEW_FOLLOWS_CONTENT_GEOMETRY	512	&H200
VA_SCALE_TO_FIT	8	&H08
VA_ADJUST_FOR_ASPECT_RATIO	4	&H04

Bedeutung / Wirkung der einzelnen Attribute:

- VA_CONTROLLED
  Das View arbeitet mit einem ViewControl zusammen
- VA_GENERIC_CONTENTS
  Wird automatisch gesetzt, wenn das Content-Objekt ein GenContent ist. Dieses Bit kann nur gelesen werden.
- VA_DRAG_SCROLLING
  Das View soll Drag-Scrolling unterstützen.
- VA_NO_WIN_FRAME
  Keinen Rahmen um das View zeichnen.
- VA_SAME_COLOR_AS_PARENT_WIN
  Die Hintergrundfarbe des View soll sich der Farbe seines Parent-Objekts anpassen.

- VA_VIEW_FOLLOWS_CONTENT_GEOMETRY
  Das View soll sich der Größe seines Content-Objekts anpassen.
- VA_SCALE_TO_FIT
  Das View soll im ScaleToFit-Modus arbeiten. Dieses Bit kann sowohl manuell als auch von einem ViewControl (zumeist im Ansicht-Menü) aus gesetzt werden.
- VA_ADJUST_FOR_ASPECT_RATIO
  Das View soll das Seitenverhältnis automatisch korrigieren. Das wird z.B. verwendet, wenn die Pixel eines Bildschirms nicht quadratisch sind.

4.9.2.4 Scaling und Scrolling

In den meisten Fällen stellt der Nutzer den gewünschten Skalierungsfaktor über das Ansicht-Menü ein, dass ein ViewControl-Objekt enthält. Die gewünschte Position im Dokument wählt der Nutzer über die Rollbalken des View. Es ist jedoch auch möglich die Skalierung und Position per Programmcode einzustellen.
Mit der Instancevariablen **scale** kann man den Skalierungsfaktor ändern. **ScaleToFitOptions** beeinflusst das Verhalten des View, wenn es sich im Modus "Größe anpassen" befindet. Manuell kann man diesen Modus einstellen, wenn man das Bit VA_SCALE_TO_FIT in der Instancevariablen viewAttrs setzt.
Die Instancevariable **origin** (engl. für Ursprung) enthält die Dokumentkoordinaten, die links oben im View dargestellt werden sollen. Mit der Methode **GetVisibleRect** erhält man die Koordinaten des gesamten, vom View dargestellten Bereichs. Je nach Skalierungsfaktor können das mehr oder weniger Pixel sein, als das View selbst groß ist. Die Methoden **ScrollToOffset** und **ScrollCmd** erlauben es dem Programmierer, das View in eine bestimmte Richtung um einen bestimmten Betrag zu scrollen. Die Instancevariable **viewIncrement** enthält die Werte um die das View bei Scrolloperationen scrollen soll. Mit **SuspendUpdate** kann man verhindern, dass aufeinander folgende Zoom- und Scrolloperationen ein wiederholtes Neuzeichnen des View bewirken. Schließlich kann man mit **DoNotWinScroll** verhindern, dass das View überhaupt auf Scroll-Kommandos reagiert.

scale

Die Instancevariable scale enthält den aktuellen Skalierungsfaktor (Zoomfaktor) des View-Objekts, getrennt für x- und y-Richtung.
Hinweis: Views, die mit einem ViewControl zusammenarbeiten (in viewAttrs ist das Bit VA_CONTROLLED gesetzt) unterstützen keine unterschiedlichen Skalierungsfaktoren in x- und in y-Richtung.

Syntax	UI-Code:	**scale** = **xScale**, **yScale**
	Lesen:	**<numVar>** = **<obj>**.**scale**(0) ' xScale lesen
		<numVar> = **<obj>**.**scale**(1) ' yScale lesen
	Schreiben:	**<obj>**.**scale** = **xScale**, **yScale**
	xScale:	Skalierungsfaktor in x-Richtung
	yScale:	Skalierungsfaktor in y-Richtung

scaleToFitOptions

Wenn der Nutzer im Ansicht-Menü den Eintrag "Größe anpassen" auswählt stellt das View den Zoomfaktor so ein, dass das gesamte Dokument sichtbar wird. Mit der Instancevariablen scaleToFitOptions können Sie dieses Verhalten modifizieren. In der Tabelle unten finden Sie die Möglichkeiten. ScaleToFitOptions sind Bitflags, d.h. Sie können mehrere Werte kombinieren.

Syntax	UI-Code:	**scaleToFitOptions** = **options** [, **xSize**, **ySize**]
	Lesen:	**<numVar>** = **<obj>**.**scaleToFitOptions** (0) Die BASIC-Syntax erfordert beim Lesen von scaleToFitOptions einen Parameter. Der Wert wird hier ignoriert.
	Schreiben:	**<obj>**.**scaleToFitOptions** = **options** [, **xSize**, **ySize**]
	options:	numerischer Wert, Bitflags, siehe Tabelle
	xSize, ySize:	Parameter für SFO_PAGE_SIZE

Mögliche Werte für scaleToFitOptions

Konstante	Wert	Wirkung
SFO_PAGE_SIZE	1	Der Skalierungsfaktor wird anhand der Parameter xSize und ySize berechnet statt nach der wahren Größe des Dokuments.
SFO_BASED_ON_X	2	Der Skalierungsfaktor wird anhand der Breite des Dokuments berechnet. In y-Richtung wird möglicherweise nur ein Teil des Dokuments sichtbar sein.
SFO_BOTH_DIMENSIONS	4	Der Skalierungsfaktor wird in x- und in y-Richtung getrennt voneinader berechnet. Dadurch ändert sich das Seitenverhältnis in Abhängigkeit von der Größe des Views.

Tipp: Um bereits am Programmstart ein "angepasstes" Dokument zu erhalten setzen Sie im UI-Code das Bit VA_SCALE_TO_FIT in der Instancevariablen viewAttrs.

```
View DemoView
  viewAttrs = VA_SCALE_TO_FIT, 0
  scaleToFitOptions = SFO_BOTH_DIMENSIONS
    ' Bewirkt, dass sich der Skalierungsfaktor in beide
    ' Richtungen automatisch der Größe des View anpasst.
  ...
END Object
```

Alternativ können Sie das Bit auch zur Laufzeit setzen. Das Aufrufen der Redraw-Methode ist nur nötig, wenn das Bit VA_CONTROLLED nicht gesetzt ist. Sie zwingt das View, sich neu darzustellen.

```
DemoView.viewAttrs = VA_SCALE_TO_FIT, 0
'Falls nötig: DemoView.Redraw
```

origin

Origin enthält die Dokument-Koordinaten, die in der linken oberen Ecke des View dargestellt werden. Wenn Sie im BASIC Code den Wert verändern scrollt das View automatisch dorthin.
Origin kann nicht im UI-Code verwendet werden.

Syntax	Lesen:	**<numVar>** = **<obj>.origin** (0)	xOrigin lesen
		<numVar> = **<obj>.origin** (1)	yOrigin lesen
	Schreiben:	**<obj>.origin** = **xOrigin** , **yOrigin**	
	xOrigin, yOrigin: Position, in Dokumentkoordinaten		

GetVisibleRect

Die Methode GetVisibleRect liefert die Koordinaten des Bereichs des Dokuments, der im View zu sehen ist. Sie liefert eine Struktur vom Typ RectDWord, die folgendermaßen definiert ist:

```
                                    Struct RectDWord
  x0, y0, x1, y1 as LongInt
  End Struct
```

X0 und y0 bezeichnen dabei die linke obere Ecke des sichtbaren Bereichs. Das entspricht der Instancevariablen origin. X1 und y1 bezeichnen die rechte untere Ecke des sichtbaren Bereichs.

Syntax	im Basic-Code:	**<rect>** = **<viewObj>.GetVisibleRect**
	rect:	Variable vom Typ RectDWord

ScrollToOffset

Die Methode ScrollToOffset scrollt das View um einen bestimmten Betrag in x- und/oder in y-Richtung.
Um direkt zu einem bestimmten Bereich des Dokuments zu scrollen setzen Sie bitte die Instancevariable origin oder verwenden Sie die Methode ScrollCmd.

Syntax im Basic-Code:	**<viewObj>.ScrollToOffset** (**xOffs**, **yOffs**)
xOffs, yOffs:	Verschiebung, in Dokumentkoordinaten Null und negative Werte sind zulässig.

ScrollCmd

Die Methode ScrollCmd scrollt das View in eine bestimmte Richtung bzw. an eine bestimmte Position.
Alternativ können Sie die Instancevariable origin setzen oder die Methode ScrollToOffset verwenden.

Syntax im Basic-Code:	**<viewObj>.ScrollCmd** (**cmd** [, **param**]
cmd:	Auszuführende Operation, siehe Tabelle
param:	Zusätzlicher Parameter für Kommandos SC_SET_Y_ORIGIN und SC_SET_X_ORIGIN

Erlaubte Kommandos für ScrollCmd. Eine "Seite" entspricht dabei immer der aktuellen Höhe bzw. Breite des im View sichtbaren Bereichs.

Konstante	Wert	Wirkung - Kommando scrollt das View ...
SC_TOP	1	nach ganz oben
SC_PAGE_UP	2	eine Seite nach oben
SC_UP	3	um den durch viewIncrement gegebenen Wert nach oben
SC_SET_Y_ORIGIN	4	an die durch "param" gegebene y-Position
SC_DOWN	5	um den durch viewIncrement gegebenen Wert nach unten
SC_PAGE_DOWN	6	eine Seite nach unten
SC_BOTTOM	7	ganz nach unten
SC_LEFT_EDGE	8	ganz nach links
SC_PAGE_LEFT	9	eine Seite nach links
SC_LEFT	10	um den durch viewIncrement gegebenen Wert nach links
SC_SET_X_ORIGIN	11	an die durch "param" gegebene y-Position
SC_RIGHT	12	um den durch viewIncrement gegebenen Wert nach rechts
SC_PAGE_RIGHT	13	eine Seite nach rechts
SC_RIGHT_EDGE	14	ganz nach rechts.

viewIncrement

Die Instancevariable viewIncrement enthält - getrennt für x- und y-Richtung - den Wert um den das View scrollen soll, wenn der Nutzer z.B. die Pfeile an den Rollbalken anklickt. ViewIncrement wird ebenso für einige Funktionen der Methode ScrollCmd (siehe oben) und für das Drag-Scrolling verwendet.

Syntax	UI-Code:	**viewIncrement** = **xInc** , **yInc**	
	Lesen:	**<numVar>** = **<obj>**.**viewIncrement** (0)	xInc lesen
		<numVar> = **<obj>**.**viewIncrement** (1)	yInc lesen
	Schreiben:	**<obj>**.**viewIncrement** = **xInc** , **yInc**	
	xInc, yInc:	neue Increment-Werte, in Dokumentkoordinaten	
		Defaultwerte: xInc = 20, yInc = 15	

SuspendUpdate

SuspendUpdate = TRUE bewirkt, dass Scrolling und Scaling Operationen zunächst nicht angezeigt werden. Das ist sinnvoll, wenn mehrere dieser Operationen nacheinander durchgeführt werden müssen und ein ständiger Neuaufbau des Bildschirms vermieden werden soll.
Ein abschließendes SuspendUpdate = FALSE stellt das View und das Dokument dann in seinem neuen Zustand dar.

Syntax	im Basic-Code:	**<viewObj>**.**suspendUpdate** = TRUE
		<viewObj>.**suspendUpdate** = FALSE

DoNotWinScroll

Verhindert, dass das View scrollt. Die Rollbalken werden trotzdem upgedatet. Sehr selten verwendet.

Syntax	UI-Code:	**DoNotWinScroll**
	Lesen:	—
	Schreiben:	—

4.9.2.5 Drag Scrolling

Stellen Sie sich vor, Sie klicken mit der linken Maustaste in das View und "ziehen" die Maus mit gedrückter Taste aus dem View. In vielen Fällen ist es erwünscht, dass das View dann zum dem Bereich scrollt, über dem sich die Maus jetzt befinden würde. Dieses Verhalten nennt man "Drag-Scrolling". Um es zu aktivieren müssen Sie nur das Bit VA_DRAG_SCROLLING in der Instance-variablen viewAttrs setzen.

```
View DemoView
  defaultTarget
  viewAttrs = VA_CONTROLLED + VA_DRAG_SCROLLING, 0
  ...
End Object
```

Per Default aktiviert das View den Drag-Scroll Modus nur beim Drücken der linken Maustaste. Mit der Methode **InitiateDragScroll** können Sie diesen Modus auch für andere Fälle, z.B. beim Drücken der rechten Maustaste, aktivieren. Schließlich können Sie mit **SetDragBounds** den Bereich, in dem das Dragging stattfinden soll, einschränken.
Die Hints **ImmediateDragUpdates** und **ImmediateDragUpdates** kontrollieren, wie oft das View während des Dragging das Dokument neu zeichnet.
Mit der Instancevariablen viewIncrement können Sie kontrollieren um welchen Betrag das Dokument bei jedem Neuzeichnen gescrollt wird.

InitiateDragScroll

Per Default wird das Drag-Scrolling vom View beim Drücken der linken Maustaste aktiviert. InitiateDragScroll aktiviert das Drag-Scrolling in anderen Situationen, z.B. beim Drücken der rechten Maustaste.

Syntax im Basic-Code:	**<viewObj>.InitiateDragScroll**

Beachten Sie, dass das View selbst keinen Maushandler haben kann. Bei Bedarf müssen Sie einen Maushandler für das Content-Objekt des Views schreiben.

SetDragBounds

Die Methode SetDragBounds schränkt den Bereich für das Drag-Scrolling ein. Das View scrollt dann während des Draggings nicht über den angegeben Bereich hinaus.

Syntax im Basic-Code:	**<viewObj>.SetDragBounds x0, y0, x1, y1**
x0, y0, x1, y1:	Koordinaten des Rechtecks, in dem das View scrollen soll.

Achtung! Das View merkt sich die DragBounds. Wenn Sie einschränkende DragBounds gesetzt haben müssen Sie sie auch wieder zurücknehmen.

Beispiel: Sowohl die linke als auch die rechte Maustaste sollen Dragging unterstützen. Beim Drücken der linken Maustaste sollen DragBounds gesetzt werden. Eine vollständige Version des Codes finden Sie bei den Beispielen unter "Objekte\View-Content\Dragging Demo"

```
View DemoView
  defaultTarget
  viewAttrs = VA_CONTROLLED + VA_DRAG_SCROLLING
  Content = DemoBitmap
  ...
END Object

BitmapContent DemoBitmap
    bitmapFormat = 960, 720, 8
    DefaultScreen
    OnMouseButton = MouseButtonHandler
END Object
```

```
MOUSEACTION MouseButtonHandler
  Case ME_LEFT_DONW
    ' Das DragScroling startet automatisch. Nur DragBounds setzen
    DemoView.SetDragBounds 50, 100, 480, 320
    End Case
  Case ME_LEFT_UP
    ' Nicht vergessen: DragBounds auf Maximum!
    DemoView.SetDragBounds 0, 0, 960, 720
    End Case
  Case ME_RIGHT_DOWN
    ' Das DragScroling manuell starten
    DemoView.InitiateDragScroll
    End Case
  End Switch
END ACTION   ' MouseButtonHandler
```

ImmediateDragUpdates, DelayedDragUpdates

Diese beiden Hints beeinflussen die Häufigkeit, mit der das Dokument während des Drag-Scrolling neu gezeichnet wird. Die Hints können nur im UI-Code gesetzt werden.

Syntax UI-Code:	**ImmediateDragUpdates** **DelayedDragUpdates**

4.9.2.6 Ändern des Mauszeigers

Häufig ist es erwünscht, dass sich die Form des Mauszeigers ändert, wenn sich die Maus über dem View befindet. Für ein Ballerspiel ist z.B. ein Fadenkreuz angemessen. Mit der Methode **SetPointerImage** können Sie einen neuen Mauszeiger festlegen. Die Methode **ClearPointerImage** stellt den vorherigen Mauszeiger wieder ein.

So definiert man einen Mauszeiger

Mauszeiger sind immer 16x16 Pixel groß. Jedes Pixel kann 4 Werte annehmen. Die folgende Tabelle zeigt die Zusammenhänge.

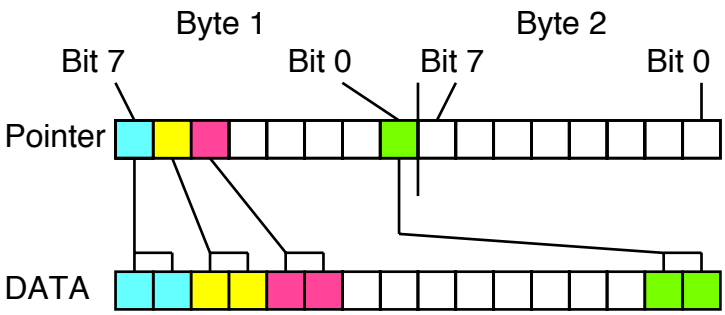
Zeiger Pixel Wert	Ergebnis auf dem Bildschirm
0	Bildschirm Pixel ungeändert (Zeiger ist transparent)
1	Bildschirm Pixel wird Schwarz
2	Bildschirm Pixel wird XOR mit Zeiger-Pixel verknüpft
3	Bildschirm Pixel wird Weiß

Mauspointer werden unter R-BASIC in DATA-Zeilen definiert. Dabei sind die ersten beiden Werte der "Hotspot", also das Pixel, das die eigentliche Position des Mauszeigers darstellt. Darauf folgen 32 Word-DATA-Werte, die den Mauszeiger beschreiben.

Der einfachste und empfehlenswerte Weg zum Erzeugen der DATA-Zeilen ist, das Programm "**Mouse Pointer Creator**", dass auf der R-BASIC Webseite verfügbar ist.

Hintergrundinformation: So sind die DATA-Werte aufgebaut

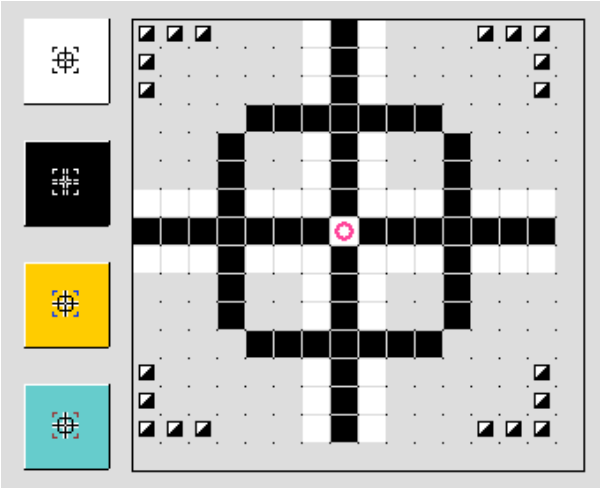
Um die DATA-Werte zu erhalten werden jeweils 8 Pixel zu einem Word-Wert entsprechend dem folgenden Schema zusammengefasst. Die Farben dienen nur dem besseren Verständnis der Zuordnung.



Damit ergibt sich folgende Berechnungsvorschrift

DATA-Wert = 16384 * Bit7 + 4096 * Bit6 + 1024 * Bit5
+ 256 * Bit4 + 64 * Bit3 + 16 * Bit2
+ 4 * Bit1 + 1 * Bit0

Beispiel: Wir wollen einen Mauszeiger haben, der so aussieht:



Grau hinterlegte Pixel sind transparent, das schwarz/weiße Grafiksymbol zeigt an, dass dieses Pixel mit dem Untergrund XOR verknüpft werden soll. Der rote Kreis ist der Hotspot. Das Bild ist ein Screenshot des "Mouse Image Creator" Programms. Auf der linken Seite ist außerdem zu sehen, wie der Mauszeiger vor verschiedenen Hintergrundfarben aussehen wird.

Die Pixel in der ersten Zeile haben entsprechend der Tabelle oben folgende Werte:

2, 2, 2, 0, 0, 0, 3, 1, 3, 0, 0, 0, 0, 2, 2, 2

Daraus ergeben sich die ersten beiden DATA-Werte zu 43021 und 49320.
Der komplette Mauspointer wird so definiert:

```
DATA  7,  7       ' Hotspot
DATA  43021, 49320,  32781, 49160,  32781, 49160,  85, 21504
DATA  269, 49408,  269, 49408,  65021, 65020,  21847, 21844
DATA  65021, 65020,  269, 49408,  269, 49408,  85, 21504
DATA  32781, 49160,  32781, 49160,  43021, 49320,  0, 0
```

Diesen Mauspointer und ein Beispiel zur Anwendung verschiedener Mauspointer finden Sie in der Beispieldatei "Mauszeiger und Freihandlinie" im Ordner "Beispiel\Objekte\View und Content".

Erfahrene Programmierer werden hier sicher die Darstellung mit Hexadezimal-Zahlen bevorzugen.

SetPointerImage

SetPointerImage liest einen Mauspointer aus Data-Zeilen und weist ihn dem View zu.

Syntax im Basic-Code:
<viewObj>.SetPointerImage

Tipp: Kennzeichnen Sie den Mauspointer durch eine LABEL Anweisung und verwenden Sie die Anweisung RESTORE <LabelName> um einen bestimmten Mauspointer anzuwählen.

```
Label   Pointer1
DATA    hotX, hotY
DATA    imageWert1, ImageWert2, ...
        ' insgesamt 32 Image Werte
```

ClearPointerImage

ClearPointerImage setzt einen Mauszeiger auf den Standard-Mauszeiger zurück.

Syntax im Basic-Code: **<viewObj>.ClearPointerImage**

4.9.2.7 Verlinkte Views

Gelegentlich ist es wünschenswert, dass Scroll-Operationen für mehrere Views gleichzeitig ausgeführt werden. Wenn Sie beispielsweise zwei (oder mehr) Views nebeneinander haben und das eine View hochscrollen kann es sinnvoll sein, dass die anderen Views automatisch mitscrollen. Man sagt dann, die Views sind miteinander "verlinkt". R-BASIC unterstützt die Verlinkung sowohl in horizontaler Richtung (Views teilen horizontale Scroll- und Scale-Ereignisse) als auch in vertikaler Richtung (Views teilen vertikale Scroll- und Scale-Ereignisse). Die geometrische Anordnung der Views auf dem Bildschirm spielt dabei keine Rolle.
Zu den von den Views geteilten Ereignissen zählen auch das Setzen der Instancevariablen origin und scale sowie die Scroll Kommandos (Methoden ScrollCmd und ScrollToOffset).

Die Verlinkung von Views muss immer zirkular erfolgen. Nehmen wir an, Sie haben 3 Views. Dann zeigt View1 auf View2, View2 auf View3 und View3 wieder zurück zu View1.

hLink

Die Instancevariable hLink enthält das nächste in horizontaler Richtung verlinkte View. Das letzte horizontal verlinkte View muss wieder zurück zum ersten View zeigen.
Horizontal verlinkte Views senden horizontale Scroll- und Scale-Ereignisse automatisch an die anderen Views weiter.

Syntax	UI-Code:	**hLink** = **<obj>**
	Lesen:	**<objVar>** = **<obj>**.**hLink**
	Schreiben:	**<viewObj>**.**hLink** = **<obj2>**

vLink

Die Instancevariable vLink enthält das nächste in vertikaler Richtung verlinkte View. Das letzte vertikal verlinkte View muss wieder zurück zum ersten View zeigen.
Vertikal verlinkte Views senden vertikale Scroll- und Scale-Ereignisse automatisch an die anderen Views weiter.

Syntax	UI-Code:	**vLink** = **<obj>**
	Lesen:	**<objVar>** = **<obj>**.**vLink**
	Schreiben:	**<viewObj>**.**vLink** = **<obj2>**

Beispiel. Die beiden Views sind sowohl horizontal als auch vertikal verlinkt und zeigen so immer den gleichen Bereich ihres Dokuments.

```
View DemoView1
  ...
  hLink = DemoView2
  vLink = DemoView2
END Object

View DemoView2
  ...
  hLink = DemoView1
  vLink = DemoView1
END Object
```

Achtung! Wenn Sie die View-Verlinkung zur Laufzeit ändern, müssen Sie selbst darauf achten, dass keine widersprüchlichen Verlinkungen entstehen. Falls Sie ein View zur Laufzeit vernichten wollen, müssen Sie sicherstellen, dass es nicht mehr mit anderen Views verlinkt ist. Verwenden Sie dazu die Funktion NullObj().

```
  obj.hLink = NullObj()
  obj.vLink = NullObj()
```

4.9.2.8 Sonstige Konfigurationsoptionen

inputOptions

Per Default sendet das View sowohl Maus- als auch Tastaturereignisse direkt an sein Content. Mit der Instancevariablen inputOptions können Sie einige Optimierungen vornehmen. InputOptions sind BitFlags, d.h. Sie können verschiedene Werte kombinieren.

Syntax	UI-Code:	**inputOptions** = **numVal**
	Lesen:	**<numVar>** = **<obj>**.**inputOptions**
	Schreiben:	**<obj>**.**inputOptions** = **numVal**
	numVal:	numerischer Wert: Siehe Tabelle

Folgende inputOptions stehen zur Verfügung:

Konstant	Wert	Funktion
VIO_DONT_SEND_MOUSE_EVENTS	1	Keine Maus-Move Ereignisse senden. Button-Ereignisse werden trotzdem gesendet.
VIO_DONT_SEND_KBD_RELEASES	2	Tastaturereignis "Taste losgelassen" nicht senden

focusable

Die Instancevariable focusable bestimmt, ob das View den Focus bekommen kann (focusable = TRUE) oder nicht (focusable = FALSE). Der Defaultwert ist TRUE.

Syntax	UI-Code:	**focusable** = **numVal**
	Lesen:	**<numVar>** = **<obj>**.**focusable**
	Schreiben:	**<obj>**.**focusable** = **numVal**
	numVal:	numerischer Wert: TRUE oder FALSE

targetable

Views erben die Instancevariable targetable von der GenericClass. Tragetable bestimmt, ob das View zum Target werden kann (targetable = TRUE, Defaultwert) oder nicht (targetable = FALSE).
Um mit einem ViewControl zusammenzuarbeiten muss das View targetable sein. Außerdem muss das Bit VA_CONTROLLED in der Instancevariablen viewAttrs gesetzt sein.

holdsLargeText

Die Instancevariable holdsLargeText muss für das View und das zugehörige VisContent auf TRUE gesetzt werden, damit die View/Content Kombination mit einem LargeText Objekt zusammenarbeiten kann. Details dazu finden Sie im Kapitel 4.10.9 (VisText und LargeText) des Objekthandbuchs.

Syntax	UI-Code:	**holdsLargeText** = TRUE
	Schreiben:	**<obj>.holdsLargeText =** TRUE I FALSE

Children eines View-Objekts

Im Normalfall besitzt ein View keine Children. Möglich ist das jedoch, da ein View von der GenericClass abstammt. Um die Children innerhalb des View zu platzieren, können Sie **den Children** die - ebenfalls von der GenericClass geerbten Hint **placeObject** geben. Dieser bestimmt, in welchem Bereich des View das Child platziert werden soll. Dabei stehen die folgenden Werte zur Verfügung:

Wert	Objekt wird platziert ...
16	X-Scroller-Bereich
32	Y-Scroller Bereich
64	Linke Seite des View
128	Obere Seite des View
256	Rechte Seite des View
512	Untere Seite des View

Beispiel: Platziere eine Button unter dem View

```
View    MyView
    Children = MyButton
    <... >
    END

Button  MyButton
    Caption$ = "Neu zeichnen"
    placeObject = 512
    < .. >
    END Object
```

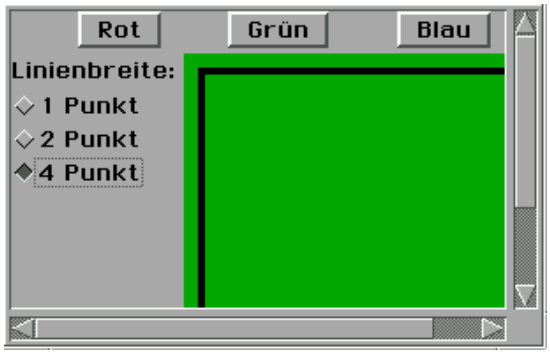
4.9.3 VisContent

Objekte der Klasse VisContent dienen primär dazu, Grafiken in einem skalierbaren und scrollbaren View auszugeben. Sie können außerdem auf Tastatur- und Mauseingaben reagieren. Das VisContent-Objekt muss nur die Grafik bereitstellen, das View-Objekt kümmert sich um den darzustellenden Bereich, Scrolling und Zoom. VisContent Objekte können Children der Klasse VisObj haben, die ihrerseits Grafik ausgeben und auf Tastatur und Maus reagieren können.
Eine ausführliche Beschreibung der VisContent Klasse finden Sie im Objekthandbuch, Kapitel 5.4.

4.9.4 BitmapContent

Objekte der Klasse BitmapContent verwalten eine editierbare Bitmap. Bitmaps sind digitalisierte Bilder. Sie bestehen aus einer rechteckigen Anordnung von einzelnen Bildpunkten (Picture Element: Pixel). Jedem Pixel kann eine eigene Farbe zugeordnet werden. In die Bitmaps der Klasse BitmapContent kann Text oder Grafik geschrieben werden. Das BitmapContent-Objekt legt die zugehörige Bitmap automatisch selbst an, so dass sie sofort benutzt werden kann.
Die BitmapContent Klasse ist von der VisContent Klasse abgeleitet. Sie erbt daher die meisten ihrer Fähigkeiten und Eigenschaften. Von besonderer Bedeutung ist dabei die Fähigkeit, auf Tastatur- und Mauseingaben zu reagieren.
Eine ausführliche Beschreibung der BitmapContent Klasse finden Sie im Objekthandbuch, Kapitel 5.2.

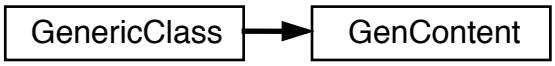
4.9.5 GenContent



Objekte der Klasse GenContent dienen dazu, GenericClass Objekte (z.B. Buttons) in einem scrollbaren View darzustellen. Die Applikation "Voreinstellungen" macht von dieser Möglichkeit Gebrauch. GenContent-Objekte können jede Art von GenericClass Objekten als Children haben.

GenContent-Objekte haben keine eigenen Maus- oder Tastaturhandler, sondern leiten diese Ereignisse wie eine Group direkt an ihre Children weiter. Wenn Sie ein GenContent-Objekt zur Laufzeit einem View zuweisen wird es automatisch visible gesetzt. Umgekehrt wird es automatisch not visible gesetzt, wenn Sie es von einem View abkoppeln.

Abstammung:



Spezielle Instance-Variablen:

Variable	Syntax im UI-Code	Im BASIC-Code
keepFocusVisible	keepFocusVisible	—
contentAttrs	contentAttrs = **attrsToSet**, **AttrsToClear**	lesen, schreiben

Tipp: Wenn die Anordnung der Children im GenContent nicht so ist, wie Sie wünschen, sollten Sie eine Group als Child des GenContent anlegen und ihre Objekte innerhalb dieser Group anordnen.

KeepFocusVisible

Der Hint keepFocusVisible sorgt dafür, dass das Objekt, das den Focus hat, nicht aus dem sichtbaren Bereich herausgescrollt wird.

Syntax	UI-Code:	**KeepFocusVisible**
	Lesen:	—
	Schreiben:	—

contentAttrs

Die Instancevariable contentAttrs enthält diverse Konfigurationsbits.

Syntax	UI-Code:	**contentAttrs** = **attrsToSet** , **attrsToClear**
	Lesen:	**<numVar>** = **<obj>**.**contentAttrs** (0) Die BASIC-Syntax erfordert beim Lesen von contentAttrs einen Parameter. Der Wert wird hier ignoriert.
	Schreiben:	**<obj>**.**contentAttrs** = **attrsToSet** , **attrsToClear**
	attrsToSet:	zu setzende Attribute, Bitflags, siehe Tabelle
	attrsToClear:	zu setzende Attribute, Bitflags, siehe Tabelle

Konstante	Wert (hex)
CA_SAME_WIDTH_AS_VIEW	128 (&h80)
CA_SAME_HEIGHT_AS_VIEW	64 (&h40)
CA_VIEW_DOC_BOUNDS_SET_MANUALLY	4 (&h04)

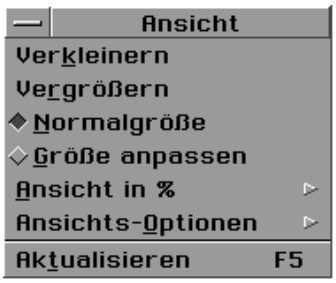
Bits, die in der Tabelle nicht aufgeführt sind sollten Sie nicht setzen. Das kann zu unerwarteten Ergebnissen führen.

Bedeutung der einzelnen Bits:

- CA_SAME_WIDTH_AS_VIEW
- CA_SAME_HEIGHT_AS_VIEW
  Das Contentobjekt passt seine Breite bzw. Höhe an die Größe des View-Objekts an. Diese Konfigurationsbits ersetzen die Hints ExpandWidth und ExpandHeight, die bei GenContent Objekten nicht funktionieren, da sie kein Parent-Objekt haben.

- CA_VIEW_DOC_BOUNDS_SET_MANUALLY
  Dieses Bit sollte für GenContent Objekte nicht hilfreich sein. Bei Bedarf finden Sie eine Beschreibung beim VisContent Objekt.

4.9.6 ViewControl

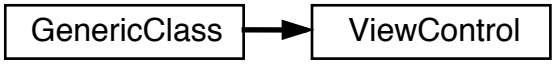


Objekte der Klasse ViewControl stellen die UI bereit, um den Zoomfaktor und einige andere Eigenschaften des aktuell aktiven View-Objekts einzustellen. Die Zusammenarbeit zwischen View und ViewControl erfolgt dabei automatisch ohne weiteres Zutun des Programmierers.

Das einzige, was Sie tun müssen, damit ein View mit dem ViewControl zusammenarbeitet ist, das Bit VA_CONTROLLED in der Instancevariablen viewAttrs des View-Objekts zu setzen. Außerdem müssen Sie für genau ein View den Hint "defaultTarget" setzen.

Üblicher Weise hat ein Programm genau ein ViewControl-Objekt, das ein Child des "Ansicht" Menüs ist. Ein Codebeispiel finden Sie am Ende des Kapitels.

Abstammung:



Spezielle Instance-Variablen:

Variable	Syntax im UI-Code	Im BASIC-Code
vcFeatures	vcFeatures = **numVal**	lesen, schreiben
vcAttrs	vcAttrs = **attrToSet**, **attrToClear**	lesen, schreiben
vcMinZoom	vcMinZoom = **numVal**	lesen, schreiben
vcMaxZoom	vcMaxZoom = **numVal**	lesen, schreiben
vcScale	—	nur lesen
targetView	—	nur lesen

Methoden:

Methode	Aufgabe
ScaleView100	"Normalgröße" aktivieren
ScaleViewToFit	"Größe anpassen" aktivieren
ZoomInView	"Vergrößern" aktivieren
ZoomOutView	"Verkleinern" aktivieren
RedrawView	"Aktualisieren" aktivieren
ScrollLeftView	View nach links scrollen
ScrollRightView	View nach rechts scrollen
ScrollUpView	View nach oben scrollen
ScrollDownView	View nach unten scrollen

vcFeatures

Die Instancevariable vcFeatures stellt ein, welche UI das ViewControl-Objekt bereitstellen soll. Jedes Bit des Wertes steht für eine Eigenschaft, die einzeln zu- oder abgeschaltet werden kann. Die Arbeit mit Bitflags ist im Programmierhandbuch, Vol. 2, Kapitel 2.3.5.4 beschrieben.

Syntax	UI-Code:	**vcFeatures** = **numVal**
	Lesen:	**<numVar>** = **<obj>**.**vcFeatures**
	Schreiben:	**<obj>**.**vcFeatures** = **numVal**
	numVal:	numerischer Wert, Bitflags, siehe Tabelle

Die folgenden Features stehen zur Verfügung. Per Default sind alle Features aktiviert, mit Ausnahme von "Aspektverhältnis umrechnen" und "Alle Fenster vergrößern / verkleinern".

Konstante	Wert	Zugehörige UI
VCF_MAIN_100	&H4000	Normalgröße
VCF_MAIN_SCALE_TO_FIT	&H2000	Größe anpassen (Hauptmenü)
VCF_ZOOM_IN	&H1000	Vergrößern-Button
VCF_ZOOM_OUT	&H0800	Verkleinern-Button
VCF_REDUCE	&H0400	Verkleinert auf 25%, 50%, 75% im Untermenü
VCF_100	&H0200	Normal (100%) im Untermenü
VCF_ENLARGE	&H0100	Vergrößert auf 125%, 150%, 175%, 200% im Untermenü
VCF_BIG_ENLARGE	&H0080	Vergrößert auf 300%, 400% im Untermenü
VCF_SCALE_TO_FIT	&H0040	Größe anpassen im Untermenü
VCF_ADJUST_ASPECT_RATIO	&H0020	Aspektverhältnis umrechnen im Untermenü
VCF_APPLY_TO_ALL	&H0010	Alle Fenster vergrößern / verkleinern im Untermenü
VCF_SHOW_HORIZONTAL	&H0008	Horizontale Bildlaufleiste ein-/ausblenden im Untermenü
VCF_SHOW_VERTICAL	&H0004	Vertikale Bildlaufleiste ein-/ausblenden im Untermenü
VCF_CUSTOM_SCALE	&H0002	Ansicht eingeben (in%) ...
VCF_REDRAW	&H0001	Neu Zeichnen
VC_DEFAULT_FEATURES	&HFFCF	Default: Alles außer "Aspektverhältnis" und "Alle Fenster vergrößern / verkleinern"

vcMinZoom

Diese Instancevariable enthält den kleinstmöglichen Scale-Faktor in Prozent. Der Defaultwert ist 25.

Syntax	UI-Code:	**vcMinZoom** = **numVal**
	Lesen:	**<numVar>** = **<obj>**.**vcMinZoom**
	Schreiben:	**<obj>**.**vcMinZoom** = **numVal**
	numVal:	numerischer Wert, in Prozent

vcMinZoom

Diese Instancevariable enthält den größtmöglichen Scale-Faktor in Prozent. Der Defaultwert ist 400. Wenn Sie den Wert auf unter 400 setzen sollten Sie auch das Bit VCF_BIG_ENLARGE in der Instancevariable vcFeatures zurücksetzen.

Syntax	UI-Code:	**vcMaxZoom** = **numVal**
	Lesen:	**<numVar>** = **<obj>**.**vcMaxZoom**
	Schreiben:	**<obj>**.**Max** = **numVal**
	numVal:	numerischer Wert, in Prozent

vcAttrs

Die Instancevariable vcAttrs enthält die Information, ob die Einstellungen aus dem Untermenü "Ansichts-Optionen" an die betroffenen Views gesendet werden sollen oder nicht. Ein Zugriff im BASIC-Code ist im Allgemeinen nicht erforderlich. Im UI-Code können Sie vcAttrs zur Konfiguration der Default-Werte für das Untermenü "Ansichts-Optionen" verwenden (siehe Beispiel unten).

Syntax	UI-Code:	**vcAttrs** = **attrsToSet** , **attrsToClear**
	Lesen:	**<numVar>** = **<obj>**.**vcAttrs** (0) Da vcAttrs zwei Parameter hat erfordet die BASIC-Syntax, dass der zu lesende Wert angegeben wird. Der übergebene Wert wird hier jedoch ignoriert.
	Schreiben:	**<obj>**.**vcAttrs** = **attrsToSet** , **attrsToClear**
attrsToSet , attrsToClear:		numerischer Wert, Bitflags, siehe Tabelle

Dieses Feld wird vom ViewControl Objekt automatisch verwaltet. Klickt der Nutzer z.B. auf "Horizontale Bildlaufleiste ein-/ausblenden" im Ansicht-Menü so wird das entsprechende Bit in vcAttrs automatsch angepasst.
Jedes Bit des Wertes steht für eine Eigenschaft, die einzeln zu- oder abgeschaltet werden kann. Die Arbeit mit Bitflags ist im Programmierhandbuch, Vol. 2, Kapitel 2.3.5.4 beschrieben.

Die folgenden Konstanten stehen zur Verfügung:

Konstante	Wert	Zugehörige Funktion
VCA_ADJUST_ASPECT_RATIO	&h8000	Aspektverhältnis umrechnen
VCA_APPLY_TO_ALL	&H4000	Alle Fenster vergrößern / verkleinern
VCA_SHOW_HORIZONTAL	&H2000	Horizontale Bildlaufleiste ein-/ ausblenden
VCA_SHOW_VERTICAL	&H1000	Vertikale Bildlaufleiste ein-/ ausblenden
VC_DEFAULT_ATTRS	&H3000	Default: Beide Bildlaufleisten ein-/ ausblenden

ScaleView100
ScaleViewToFit
ZoomInView
ZoomOutView
RedrawView
ScrollLeftView
ScrollRightView
ScrollUpView
ScrollDownView

Diese Methoden bewirken, dass das ViewControl die in der Tabelle aufgelisteten Kommandos an das kontrollierte View sendet. Sie können damit z.B. Toolbuttons realisieren, die die wichtigsten Operationen des View-Menüs auslösen können. Ein Beispiel finden Sie im Ordner Beispiel\Objekte\View-Content in der Datei "ViewControl Tool Demo". Um die grafischen Captions dieses Beispiels nutzen zu können müssen Sie mindestens die Version 2 des Pakets "More Tool Images" von der R-BASIC Webseite installiert haben.

Syntax am Beispiel ScaleViewToFit:
<ViewControlObj>.ScaleViewToFit

Methode	Aufgabe
ScaleView100	"Normalgröße" aktivieren
ScaleViewToFit	"Größe anpassen" aktivieren
ZoomInView	"Vergrößern" aktivieren
ZoomOutView	"Verkleinern" aktivieren
RedrawView	"Aktualisieren" aktivieren
ScrollLeftView	View nach links scrollen
ScrollRightView	View nach rechts scrollen
ScrollUpView	View nach oben scrollen
ScrollDownView	View nach unten scrollen

Hinweis: Wenn das ViewControl mehrere Views kontrolliert (in vcAttrs is VCF_APPLY_TO_ALL gesetzt und die Option ist aktiv) arbeitet diese Methoden

häufig nur mit einem der Views (mit dem, das aktuell das Target ist). Sie können dann die Views direkt ansprechen um die gewünschte Operation auszuführen.

targetView

Die Instancevariable targetView enthält das aktive View, d.h. das View, das aktuell vom ViewControl kontrolliert wird, oder ein Null-Objekt, falls gerade kein View aktiv ist. Der Wert kann nur gelesen werden.

Syntax Lesen: **<objVar>** = **<obj>**.**targetView**

Beispiele

Im einfachsten Fall müssen Sie das ViewControl-Objekt nicht konfigurieren.

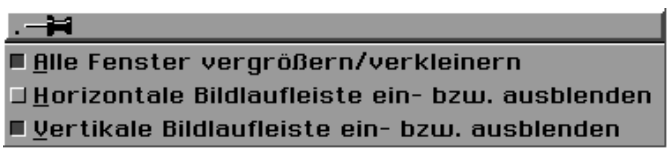
```
Menu DemoViewMenu
  Caption$ = "Ansicht"
  Children = DemoViewControl
End Object

ViewControl DemoViewControl
End Object
```

Um ein View mit dem ViewControl Objekt zusammen arbeiten zu lassen müssen Sie zwei Einstellungen vornehmen:

```
View DemoView
  DefaultTarget
  viewAttrs = VA_CONTROLLED, 0
  ...
END Object
```

Nehmen wir jetzt an, wie haben eine Applikation mit 2 Views. Eins davon sollte den Hint DefaultTarget gesetzt haben, beide das Bit VA_CONTROLLED.
Wir wollen folgendes:

- Der Zoom-Faktor soll auf 200% begrenzt werden.
- Im ViewControl soll die Option "Alle Fenster vergrößern / verkleinern" vorhanden sein.
- Per Default soll diese Option aktiv sein.
- Per Default soll der horizontale Rollbalken verborgen sein.



Das zugehörige ViewControl sieht dann so aus:

```
ViewControl DemoViewControl
  vcFeatures = VC_DEFAULT_FEATURES + VCF_APPLY_TO_ALL \
               – VCF_BIG_ENLARGE
  vcMaxZoom = 200
  ' Default-Werte für Ansichts-Optionen setzen
  vcAttrs = VCA_APPLY_TO_ALL, VCA_SHOW_HORIZONTAL
End Object
```

(Leerseite)

(Leerseite)