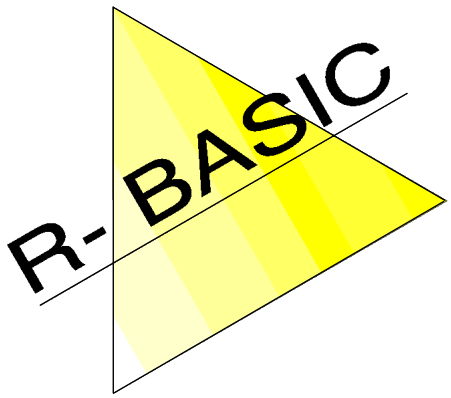
R-BASIC

Einfach unter PC/GEOS programmieren



Objekt-Handbuch

Volume 10
Visual Objekt Klassen

Version 1.0

(Leerseite)

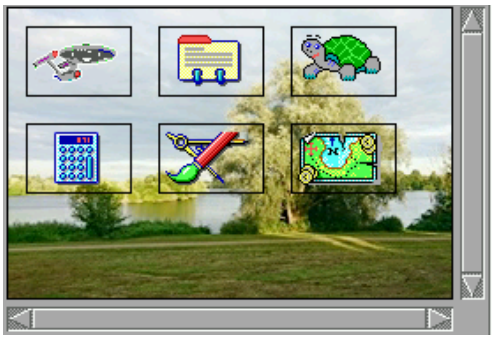
Inhaltsverzeichnis

Volume 9

Volume 10

(Leerseite)

5.4 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.

Dazu kann das VisContent-Objekt selbst Grafik ausgeben, oder es verwendet Children (der Klasse VisObj), die ihrerseits Grafik ausgeben und auf Tastatur und Maus reagieren können. Es ist die Entscheidung des Programmierers, welcher Weg benutzt wird. Es ist auch möglich, dass das VisContent, wie im Bild gezeigt, eine Hintergrundgrafik ausgibt, auf die dann die VisObj-Objekte gezeichnet werden.

Abstammung:

Da VisContent Objekte von der VisualClass abstammen, kommen Sie nicht in den generic Tree des Programms. Stattdessen werden sie über die Instance-Variable "Content" eines Views mit dem View verbunden. Das View muss aber in den generic Tree des Programms eingebunden werden.

Spezielle Instancevariablen:

Variable	Syntax im UI-Code	Im BASIC-Code
contentAttrs	contentAttrs = **toSet** , **toClear**	lesen, schreiben
holdsLargeText	holdsLargeText = **TRUE**	lesen, schreiben

Methoden:

Methode	Aufgabe
CreateVisObject	Neues VisObj-Objekt anlegen
DestroyVisObject	Mit CreateVisObject VisObj-Objekt vernichten

5.4.1 Grundlegende Fähigkeiten

VisContent-Objekte erben viele Fähigkeiten von der VisualClass. Dazu gehören die Arbeit mit der Maus, der Tastatur, dem Clipboard sowie die Arbeit mit der Focus- und der Target-Hierarchie. Im Kapitel 5.1 dieses Handbuchs (Die Visual-

Class) finden Sie dazu eine Kurzbeschreibung sowie Verweise auf die zugehörigen Kapitel im Handbuch.

Weitere Fähigkeiten erben VisContent-Objekte von der VisGroup Class. Das sind insbesondere die Darstellung auf dem Bildschirm (Ausgabe von Text und Grafik), die Anordnung der eigenen Children sowie die Bestimmung der eigenen Größe. Die zugehörigen Instancevariablen, Methode und Actionhandler sind ausführlich im Kapitel 5.3 (VisGroup) des Objekthandbuchs beschrieben.

Besondere Hinweise für VisContent-Objekte

- Das VisContent-Objekt gibt (wie alle R-BASIC Objekte) die Tastaturereignisse zuerst an seine Children weiter, bevor es den BASIC Tastaturhandler aufruft. Sollten beide Objekte (Content und Children) einen Tastaturhandler haben wird daher zuerst der Tastaturhandler der Children-Objekte aufgerufen und erst danach der Tastaturhandler des Content-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 seine Children weitergibt.
- Auch Mausereignisse werden zuerst an die Children weitergegeben, bevor der eigene BASIC Handler aufgerufen wird. Sollten beide Objekte (Content und Children) einen Maushandler haben wird daher zuerst der Maushandler der Children-Objekte aufgerufen und erst danach der Maushandler des Content-Objekts.
- Um mit der Zwischenablage arbeiten zu können muss das VisContent-Objekt im gepufferten Modus arbeiten (Instancevariable **buffered** = TRUE).
- Sie sollten die visPosition-Werte für VisContent-Objekte nicht ändern. Das kann zu unerwarteten Ergebnissen, insbesondere einer Verschiebung der Position aller beteiligten Objekte, führen.

contentAttrs

Die Instancevariable contentAttrs enthält drei Konfigurationsbits. Bits, die in der Tabelle unten nicht aufgeführt sind sollten Sie nicht setzen. Das kann zu unerwarteten Ergebnissen führen, da sie intern verwendet werden.

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)

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, wenn das möglich ist. Häufig stellt man das View in die entsprechende Richtung dann so ein, dass es nicht scrollbar ist.

  Hinweis:
  Im Allgemeinen müssen Sie zusätzlich die Bits VSF_EXPAND_WIDTH bzw. VSF_EXPAND_HEIGHT in der Instancevariablen visSizeFlags setzen, um die gewünschte Wirkung zu erreichen. Bitte beachten Sie die Konfigurationen in den Beispieldateien.

- CA_VIEW_DOC_BOUNDS_SET_MANUALLY
  Dieses Bit wird selten benutzt und verhindert, dass das VisContent dem View automatisch seine Größe mitteilt. Sie müssen dann die Instancevariable contentSize des View verwenden, um dem View-Objekt die Content-Größe mitzuteilen. Das View benötigt diese Information, um seine Rollbalken anzupassen.

holdsLargeText

Die Instancevariable holdsLargeText muss für das VisContent und das zugehörige View auf TRUE gesetzt werden, damit die View/VisContent 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

5.4.2 View-Content Konfiguration

Auf welche Weise ein VisContent mit seinem View interagiert, ob es seine Größe dem View anpasst oder umgekehrt, ob das gesamte Content oder nur ein Teil davon zu sehen ist, hängt sowohl von den Einstellungen des VisContent als auch ganz entscheidend von der Einstellung des View-Objekts ab.
Eine ausführliche Beschreibung alle Instancevariablen eines View finden Sie im Kapitel 4.9 des Objekthandbuchs. Die folgenden Abschnitte beschreiben typische Konfigurationen und decken einen großen Teil der Möglichkeiten ab.

5.4.2.1 Wenn das Content eine feste Größe hat

Typische Situationen, in der ein VisContent-Objekt eine feste Größe hat, sind z.B. eine Seite in einen Schreibprogramm oder ein Gameboard.
Um einem VisContent eine feste Größe zu geben, gibt es zwei Möglichkeiten. Entweder, Sie haben eine Objekt-Tree, dessen Objekte eine feste Position und Größe haben, oder Sie geben dem Content explizit eine feste Größe, indem sie folgende Instancevariablen im VisContent setzen:

```
  visSizeOptions = VSO_FIXED_SIZE
  visSize = 400, 330              ' Als Beispiel
```

Im letzten Fall kann das VisContent ebenfalls Children haben, muss es aber nicht.

Jeweils ein Beispiel für beide Varianten finden Sie in den Beispieldateien "VisContent Fixed Size" und "VisContent Fixed Children", im Beispiel-Ordner "Visual Class".

Wenn das Content eine feste Größe hat wird das View sich entweder der Größe des Content anpassen, oder in eine oder beide Richtungen scrollbar sein. Im Folgenden werden ein paar typische Konfigurationen dargestellt.

Um das Content in einem in beide Richtungen scrollbaren View darzustellen, muss das View folgendermaßen konfiguriert werden. Ändert der Nutzer jetzt die Größe des View, z.B. indem er die Größe des zugehörigen Hauptfensters ändert, so wird mehr oder weniger vom Content-Objekt zu sehen sein und die Rollbalken passen sich an.

```
View DemoView
  Content = DemoContent
  hControl = HVC_SCROLLABLE + HVC_NO_LARGER_THAN_CONTENT
  vControl = HVC_SCROLLABLE + HVC_NO_LARGER_THAN_CONTENT
  initialSize = 200, 200      ' zum Beispiel
End OBJECT
```

Um nur in eine Richtung scrollable zu sein, passen wir die Größe des Views in die andere Richtung an die des Content an.

```
View DemoView
  Content = DemoVisContent
  hControl = HVC_SCROLLABLE + HVC_NO_LARGER_THAN_CONTENT
  vControl = HVC_NO_LARGER_THAN_CONTENT +\
                          HVC_NO_SMALLER_THAN_CONTENT
End OBJECT
```

Eine weitere Möglichkeit ist, dass das View in beide Richtungen stets exakt so groß ist, wie das Content. Um das zu erreichen gibt es zwei Möglichkeiten:

```
View DemoView
  Content = DemoContent
  hControl = HVC NO LARGER THAN CONTENT + \
                              HVC NO SMALLER THAN CONTENT
  vControl = HVC NO LARGER THAN CONTENT + \
                              HVC NO SMALLER THAN CONTENT
End OBJECT
```

oder

```
View DemoView
  Content = DemoContent
  viewAttrs = VA_VIEW_FOLLOWS_CONTENT_GEOMETRY , 0
End OBJECT
```

Eine besondere Situation ist, dass der Nutzer die Größe des View-Objekts verändern kann, aber stets das gesamte Content-Objekt sichtbar sein soll. Das ist z.B. für ein Gameboard interessant. Wenn der Nutzer die Größe des Spielprogramms verändert, wird stets das gesamte Spielfeld sichtbar bleiben.
Dazu muss das View seinen Zoomfaktor automatisch so ändern, dass das gesamte Content innerhalb des verfügbaren Platzes dargestellt wird. Das erreicht man, indem das Bit VA_SCALE_TO_FIT in der Instancevariablen **viewAttrs** gesetzt wird.
Damit das View seine Anfangsgröße kennt und anfangs einen Skalierungsfaktor von 1 einstellt, geben wir sowohl einen Wert für initialSize als auch (den gleichen Wert) für contentSize vor.

Im Folgenden werden mehrere Varianten vorgestellt, die sich in ihrem Verhalten jeweils etwas unterscheiden. Den kompletten Code finden Sie in der Beispieldatei "VisContent ScaleToFit" im Ordner "Visual Class".

Variante 1:
Die Höhe des Views soll sich der vom Nutzer eingestellten Breite anpassen.

```
View DemoView
  Content = DemoContent
  viewAttrs = VA_SCALE_TO_FIT, 0
  initialSize = 490,210           ' Das passt in unserem Beispiel
  contentSize = 490,210

  hControl = HVC NO LARGER THAN CONTENT
  vControl = HVC KEEP ASPECT RATIO

End OBJECT
```

Variante 2:
Analog zur Variante 1 soll sich die Breite des Views der vom Nutzer eingestellten Höhe anpassen.

```
View DemoView
  Content = DemoContent
  viewAttrs = VA_SCALE_TO_FIT, 0
  initialSize = 490,210           ' Das passt in unserem Beispiel
  contentSize = 490,210

  hControl = HVC_KEEP_ASPECT_RATIO
  vControl = HVC NO LARGER THAN CONTENT

End OBJECT
```

Variante 3:
Der Skalierungsfaktor orientiert sich sowohl an der Breite als auch an der Höhe des View. ' Dabei kann das View höher als das Content werden.

```
View DemoView
  Content = DemoContent
  viewAttrs = VA SCALE TO FIT, 0
  initialSize = 490,210           ' Das passt in unserem Beispiel
  contentSize = 490,210

  hControl = HVC_NO_LARGER_THAN_CONTENT
  vControl = HVC_NO_LARGER_THAN_CONTENT

End OBJECT
```

Variante 4:
SFO_BOTH_DIMENSIONS in der Instancevariablen scaleToFitOptions bewirkt, dass der Zoomfaktor in beide Richtungen unabhängig voneinander berechnet wird.

```
View DemoView
  Content = DemoContent
  viewAttrs = VA_SCALE_TO_FIT, 0
  initialSize = 490,210           ' Das passt in unserem Beispiel
  contentSize = 490,210

  scaleToFitOptions = SFO_BOTH_DIMENSIONS
  hControl = HVC_NO_LARGER_THAN_CONTENT
  vControl = HVC_NO_LARGER_THAN_CONTENT

End OBJECT
```

5.4.2.2 Wenn das Content eine variable Größe hat

Typische Situationen, in den ein VisContent-Objekt eine veränderliche Größe hat sind z.B. ein Dateimanager oder der Grafikviewer Gonzo. Hier kann sich die Anzahl der dargestellten Objekte (Children des VisContent) ändern und wenn Sie die Größe des Hauptfensters verändern, ordnen sich die Children neu an.

Im Allgemeinen ist das View in diesem Fall in eine Richtung scrollbar, während sich das VisContent in der anderen Richtung an die Größe des Views anpasst. Das folgende Codefragment zeigt eine typische Konfiguration für diesen Fall. Das View passt sich der Größe seines Parent-Objekts an und ist in vertikaler Richtung scrollbar. Das Content passt sich in der Breite der Größe des View an, seine Höhe berechnet es aus der Geometrie seiner Children. Diese oder eine ähnliche Konfiguration wird in mehreren Beispielen im Ordner "Visual Class" verwendet.

```
View DemoView
  Content = DemoContent
  vControl = HVC_SCROLLABLE
  initialSize = 400, 250
  ExpandWidth
  ExpandHeight
  ...
End OBJECT

VisContent DemoContent
  allowChildrenToWrap = TRUE
  ' Das Content-Objekt soll immer so breit sein, wie das
  ' zugehörige View.
  ' Dazu müssen die folgenden Instancevariablen gesetzt sein.
  contentAttrs = CA_SAME_WIDTH_AS_VIEW , 0
  visSizeFlags = VSF_EXPAND_WIDTH
  ...
End OBJECT
```

5.4.3 Anlegen und Vernichten von Objekten

Methoden:

Methode	Aufgabe
CreateVisObject	Neues VisObj-Objekt anlegen
DestroyVisObject	Mit CreateVisObject VisObj-Objekt vernichten

Im Zusammenhang mit VisObj-Objekten ist es sehr häufig, dass Sie VisObj-Objekte zur Laufzeit anlegen und wieder vernichten müssen. Ein Beispiel ist der GeoMananger, der für jede Datei im Verzeichnis ein eigenes VisObj-Objekt verwendet. Auch der Grafikviewer Gonzo legt für jede gefundene Grafikdatei ein VisObj-Objekt an und vernichtet es wieder, wenn das Verzeichnis gewechselt wird.
Natürlich können Sie die im Kapitel 2.1.5 des Objekthandbuchs beschriebenen Routinen CreateObject und DestroyObject auch für VisObj-Objekte benutzen. Sie müssen sich dann aber auch um das Anlegen und die Verwaltung der zugehörigen Objektblöcke kümmern. Insbesondere, wenn Sie viele Objekte anlegen wollen bzw. deren Anzahl vorher nicht kennen, kann das sehr aufwändig sein.

Aus diesem Grund bietet das VisContent Objekt zwei Methoden, mit denen Sie VisObj-Objekte zur Laufzeit ganz einfach anlegen und wieder vernichten können. Das VisContent managed die erforderlichen Objektblöcke dabei selbständig. **CreateVisObject** legt ein neues VisObj-Objekt an und bindet es bei Bedarf als Child des VisContent in den visual Tree ein. **DestroyVisObject** vernichtet ein mit CreateVisObject angelegtes VisObj-Objekt wieder.

CreateVisObject

Die Methode CreateVisObject legt ein neues VisObj-Objekt an und bindet es bei Bedarf als Child des VisContent in den visual Tree ein. Falls erforderlich wird ein neuer Objektblock angelegt.

Syntax: **<objVar>** = **<obj>**.**CreateVisObject** **xSize**, **ySize** [,**addAsChild**]
xSize: Anfängliche Breites des neu angelegten Objektes
ySize: Anfängliche Höhe des neu angelegten Objekts
addAsChild: Objekt als Child des VisContent in den visual Tree einbinden (TRUE) oder nicht (FALSE). Der Defaultwert ist TRUE, d.h. das Objekt wird als letztes Child des VisContent eingebunden.

Um Objekte, die mit der Methode CreateVisObject angelegt wurden, zu vernichten, müssen Sie die Methode DestroyVisObject verwenden. Verwenden Sie niemals die Routine DestroyObject dafür. Allerdings ist es zulässig ein Programm zu beenden, ohne die angelegten VisObj-Objekte zu vernichten.

Nachdem das Objekt angelegt wurde, muss es noch weiter initialisiert werden. Im Allgemeinen benötigt es einen OnDraw-Handler sowie häufig einen OnMouseButton-Handler.
Nachdem alle gewünschten VisObj-Objekte angelegt wurden, müssen Sie den visual Tree aktualisieren, indem Sie die Methode MarkInvalid für das VisObj-Objekt oder das VisContent aufrufen. Erst dann erscheinen die neu angelegten Objekte auf dem Schirm.

Beispiel. Den vollständigen Code finden Sie in der Beispieldatei "Create VisObj Demo".

```
BUTTONACTION DemoCreateChild
DIM obj AS OBJECT

  obj = DemoContent.CreateVisObject 160, 100

  ' Initialisieren: Drawhandler, Maushandler usw.
  obj.OnDraw =   VisObjDraw
  <mehr ...ausgelassen>

  ' Das neue Objekt auf dem Schirm sichtbar machen
  obj.MarkInvalid

End ACTION   ' DemoCreateChild
```

Im Modus **customManageChildren** ist es häufig sinnvoll, das neue VisObj nicht als letztes, sondern als erstes Child des VisContent einzubinden, damit es über allen schon vorhandenen Objekten gezeichnet wird. Dazu ist es sinnvoll (aber nicht erforderlich), die Methode CreateVisObject mit dem Parameter addAsChild = FALSE aufzurufen. Abschließend weisen wir das VisContent als Parent zu, wobei als Child-Position der Wert Null verwendet wird. Außerdem müssen wir dem Objekt eine visPosition zuweisen. Im Beispiel berechnen wir sie aus der Anzahl der Children, die das DemoContent schon hat. Um das Objekt sichtbar zu machen, müssen wir abschließend die Methode MarkInvalid aufrufen.

Beispiel. Den kompletten Code finden Sie in Beispieldatei "Create Custom Managed VisObj".

```
DIM obj AS OBJECT
DIM n

  obj = DemoContent.CreateVisObject 80, 60, FALSE
  obj.Parent = DemoContent, 0

  n = DemoContent.numChildren - 1
  obj.visPosition = 20*n + 10, 20*n+ 10

  ' Initialisieren: DrawHandler, Maushandler
  < .. siehe Beispiel-Datei ..>

  obj.MarkInvalid

End ACTION   ' DemoCreateChild
```

Dynamisch mit CreateVisObject angelegten Objekte lassen sich ohne Einschränkungen genauso verwenden, wie statisch im UI-Code deklarierte Objekte. Es ist also möglich, aus den neu angelegten VisObj-Objekten einen komplexen Tree aufzubauen. Dazu müssen Sie einfach die Instancevariable parent des VisObj-Objekts entsprechend setzen. Außerdem ist es zulässig, dynamisch mit CreateVisObject angelegte Objekte mit statisch im UI-Code deklarierten Objekten in einem Tree zu vermischen.

DestroyVisObject

Die Methode DestroyVisObject vernichtet ein mit der Methode CreateVisObject angelegtes Objekt. Enthält der zugehörige Objektblock danach keine Objekte mehr, so wird er automatisch freigegeben.

Syntax: **<obj>.DestroyVisObject <objVar>**
<objVar>: Referenz auf ein mit CreateVisObject angelegtes VisObj-Objekt

---

Sie können DestroyVisObject nur mit Objekten verwenden, die von CreateVisObject angelegt wurden. Andernfalls kommt es zu einem Laufzeitfehler.
Sie können ein VisObj-Objekt vernichten, wenn es noch mit seinem Parent verlinkt ist. Allerdings darf es keine Children haben, sonst kommt es zu einem Laufzeitfehler.
Wenn Sie ein Objekt mit DestroyVisObject vernichten, müssen Sie im Modus "customManageChildren" die Methode MarkInvalid aufrufen, damit das Objekt vom Bildschirm verschwindet. Im normalen Modus ist dies nicht nötig, der visual Tree stellt sich automatisch neu dar.

Beispiel: Das letzte Child eines VisContent-Objekts soll vernichtet werden.

```
SUB DestroyLastChild
DIM ob as OBJECT
DIM count

  count = DemoContent.numChildren
  IF count = 0 THEN RETURN            ' Keine Children mehr

  ' Die Zählung der Children beginnt bei Null
  ' -> Das letzte Child hat die Nummer count-1
  ob = DemoContent.children(count-1)
  DemoContent.DestroyVisObject ob

End SUB
```

Beispiele für die Verwendung von DestroyVisObject finden Sie in den Beispieldateien "Create VisObj Demo" und "Create Custom Managed VisObj".

5.5 VisObj

5.5.1 Überblick

Objekte der Klasse VisObj dienen primär dazu, Grafiken auszugeben.
Sie können außerdem auf Tastatur- und Mauseingaben reagieren. Die VisObj-Objekte (bzw. das VisContent-Objekt) müssen nur die Grafik bereitstellen, das zugehörige View-Objekt kümmert sich um den darzustellenden Bereich, Scrolling und Zoom.

VisObj-Objekte können direkte Children eines VisContent-Objekts sein und sie können Children der Klasse VisObj haben, die ihrerseits Grafik ausgeben und auf Tastatur und Maus reagieren können. Es ist eine Frage des Programm-Konzepts, ob man die grafische Ausgabe über ein BitmapContent, ein VisContent ohne Children oder eine Reihe von VisObj-Objekten realisiert. VisObj-Objekte bieten sich immer an, wenn einzelne Objekte mit der Maus angeklickt oder über den Bildschirm bewegt werden sollen.

Abstammung:

VisObj-Objekte erben viele Fähigkeiten von der VisualClass. Dazu gehören die Arbeit mit der Maus, der Tastatur, dem Clipboard sowie die Arbeit mit der Focus- und der Target-Hierarchie. Im Kapitel 5.1 dieses Handbuchs (Die VisualClass) finden Sie dazu eine Kurzbeschreibung sowie Verweise auf die zugehörigen Kapitel im Handbuch.

Weitere Fähigkeiten erben VisObj-Objekte von der VisGroup Class. Das sind insbesondere die Darstellung auf dem Bildschirm (Ausgabe von Text und Grafik), die Anordnung der eigenen Children sowie die Bestimmung der eigenen Größe. Die zugehörigen Instancevariablen, Methode und Actionhandler sind ausführlich im Kapitel 5.3 (VisGroup) des Objekthandbuchs beschrieben.

Besondere Hinweise für VisObj-Objekte

- Um mit der Zwischenablage arbeiten zu können muss das VisObj-Objekt im gepufferten Modus arbeite (Instancevariable **buffered** = TRUE).
  Verwenden Sie ClpPaste nicht, wenn das Objekt aktuell der Screen ist. Es kommt dann zu einer Verschiebung der eingefügten Grafik.

Spezielle Instancevariablen:

Variable	Syntax im UI-Code	Im BASIC-Code
visDataValue	visDataValue = **numWert**	lesen, schreiben
drawable	drawable = TRUE | FALSE	lesen, schreiben
detectable	detectable = TRUE | FALSE	lesen, schreiben
managed	managed = TRUE | FALSE	lesen, schreiben
isDragging	—	nur lesen
dragGString	—	lesen, schreiben
dragPosition	—	nur Lesen
dragOffset	—	nur Lesen
grabFocusOnMouseEvents	grabFocusOnMouseEvents = **numWert**	lesen, schreiben
grabTargetOnMouseEvents	grabTargetOnMouseEvents = **numWert**	lesen, schreiben

Methoden:

Methode	Aufgabe
DragStart	Drag-Modus initialisieren
DragMoveTo	Umriss im Drag-Modus bewegen
DragEnd	Drag-Modus beenden, Objekt neu positionieren
DragAbort	Drag-Modus abbrechen
DrawHandles	Anfasser zeichnen
TestHandles	Prüfen, ob Position innerhalb eines Anfassers liegt
DrawFrame	Rahmen um das Objekt zeichnen
DrawInverse	Bereich des Objekts invertiert zeichnen

5.5.2 Grundlegende Fähigkeiten

Spezielle Instancevariablen:

Variable	Syntax im UI-Code	Im BASIC-Code
visDataValue	visDataValue = **numWert**	lesen, schreiben
drawable	drawable = TRUE | FALSE	lesen, schreiben
detectable	detectable = TRUE | FALSE	lesen, schreiben
managed	managed = TRUE | FALSE	lesen, schreiben

visDataValue

Die Instancevariable visDataValue enthält einen numerischen Wert (LongInt-Bereich), mit dem Sie bei Bedarf verschiedene VisObj-Objekte auseinanderhalten können, ohne auf die relativ umständliche Verwendung der Instancevariablen privData zurückzugreifen. Außerdem können Sie visDataValue bereits im UI-Code zuweisen.

Syntax UI-Code: **visDataValue** = **value**
value: numerischer Wert, LongInt
Lesen: **<numVar>** = **<obj>**.**visDataValue**
Schreiben: **<obj>**.**visDataValue** = **value**

Natürlich steht Ihnen die Instancevariable privData auch für VisObj-Objekte zur Verfügung, wenn Sie für jedes individuelle VisObj-Objekt mehr als einen einzelnen Zahlenwert abspeichern wollen. Eine Beschreibung der Instancevariablen privData finden Sie im Handbuch Spezielle Themen, Kapitel 19.

Beispiel: Sie haben mehrere VisObj-Objekte, die den gleichen OnDraw-Handler haben sollen, sich aber in der Farbe unterscheiden müssen.

```
VisObj  Obj1
  OnDraw = VDraw
  visDataValue = RED
  ...
END OBJECT

VisObj  Obj1
  OnDraw = VDraw
  visDataValue = BLUE
  ...
END OBJECT
```

```
DRAWCATION VDraw
  Color sender.visDataValue, 0
  ...
END ACTION
```

drawable

Die Instancevariable drawable bestimmt, ob sich das Objekt und seine Children auf dem Schirm darstellen können (TRUE), oder nicht (FALSE). Der Defaultwert ist TRUE. Setzen Sie den Wert zur Laufzeit auf FALSE, so werden das Objekt und seine Children vom Schirm genommen und der Hintergrund im Bereich des Objekts wird neu gezeichnet.
Beachten Sie, dass das Objekt auch weiterhin bei der Berechnung der Geometrie berücksichtigt wird.

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

Hinweis für einen seltenen Fall: In Bereichen, in denen die Children über den Rand des Objekts hinausragen sollten, wird der Hintergrund nicht neu gezeichnet, wenn Sie drawable zur Laufzeit auf FALSE setzen. Rufen Sie in diesem Fall die Methode MarkInvalid für das zugehörige VisContent auf.

detectable

Die Instancevariable detectable bestimmt, ob das Objekt Mausereignisse erhalten kann (TRUE), oder nicht (FALSE). Der Defaultwert ist TRUE.

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

managed

Die Instancevariable managed bestimmt, ob das Objekt mit dem Geometrie-manager zusammenarbeiten kann (TRUE) oder nicht (FALSE). Der Defaultwert ist TRUE. Setzen Sie den Wert auf FALSE, so wird der Geometriemanager das Objekt ignorieren. Sie müssen dann einen Wert für visPosition setzen.
Sie können den Wert auch im Modus customManageChildren auf TRUE lassen.

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

Hinweis für einen seltenen Fall: Sollten Sie managed zur Laufzeit auf FLASE setzen, so müssen Sie anschließend (!) einen Wert für visPosition zuweisen und die Methode MarkInvalid aufrufen. Je nach Situation wird dabei möglicherweise der Hintergrund nicht korrekt neu gezeichnet. Rufen Sie in diesem Fall die Methode MarkInvalid für das zugehörige VisContent auf.
Das gilt entsprechend auch, wenn Sie managed zur Laufzeit auf TRUE setzen.

5.5.3 Maus- und Tastatur-Input

5.5.3.1 Arbeit mit der Maus

Sie können innerhalb eines Maushandlers Grafik auf den Schirm ausgeben, z.B. um einen Umriss zu zeichnen, während Sie das Objekt über den Schirm bewegen. Dazu müssen Sie das Objekt temporär zum Screen machen. Eine ausführliche Beschreibung, wie man mit der Maus arbeitet, finden Sie im Handbuch "Spezielle Themen", Kapitel 17.

Es ist dabei von essentieller Bedeutung, dass wir der Zuweisung der globalen Screen-Variablen große Aufmerksamkeit widmen. Sie sollten am Beginn jeder Zeichenaktion den aktuell aktiven Screen in einer Variablen sichern und ihn am Ende wieder zurücksetzen. Vergessen Sie das Zurücksetzen des Screens, kann GEOS crashen - entweder gleich oder beim Beenden des Programms. Nur BitmapContent-Objekte dürfen beim Beenden des Programms der Screen sein.

Ähnliches gilt für das Grabben der Maus. GEOS wird crashen, wenn beim Beenden des Programms noch ein Objekt die Maus gegrabbt hat.

Das folgende Codefragment stammt aus dem Beispiel "Create VisObj Demo" und zeigt, wie man korrekt auf einen Mausklick reagiert. Die Sub DrawFigure sichert den Screen in einer (lokalen) Variablen und zeichnet eine inverse Ellipse.

```
MOUSEACTION VisObjButtonPressed

  ON event SWITCH
  CASE ME LEFT DOWN
    ' Linke Maustaste gedrückt: Inversen Kreis zeichnen
    DrawFigure(sender)
    ' Maus grabben, damit das Loslassen der
    ' Maustaste erkannt wird, selbst wenn der Mauszeiger
    ' das Objekt verlassen hat
    sender.GrabMouse
    End CASE
  CASE ME_LEFT_UP
    ' Inversen Kreis löschen und Maus releasen
    DrawFigure(sender)
    sender.ReleaseMouse
    End CASE
  End SWITCH

End ACTION   ' VisObjButtonPressed
```

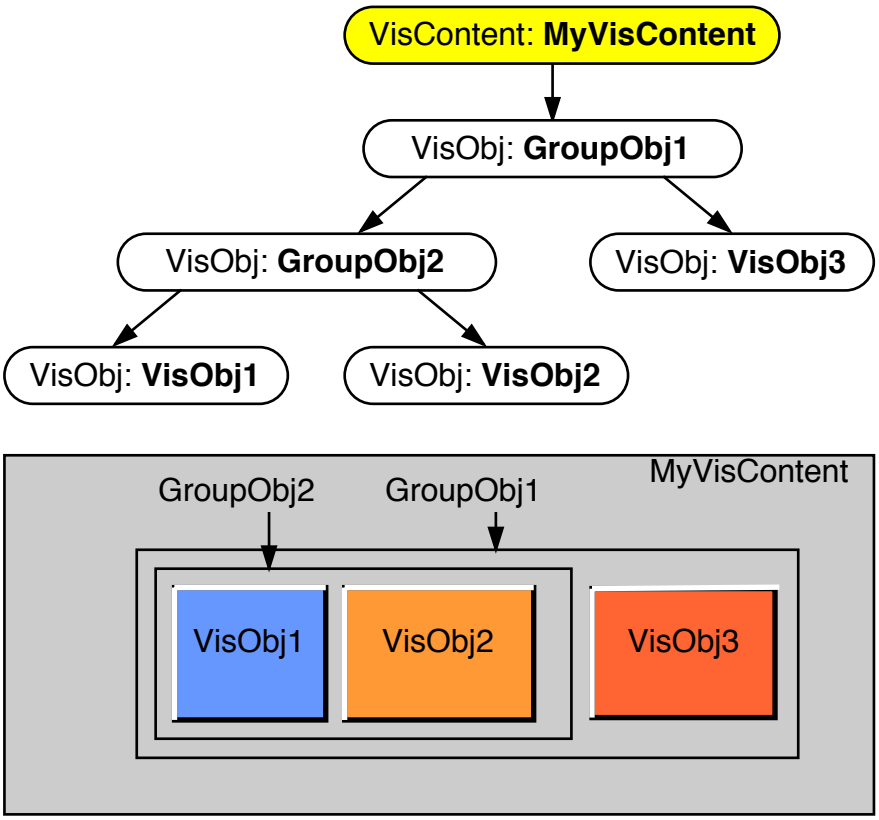
```
SUB DrawFigure (obj AS OBJECT)
DIM scr AS OBJECT

  ' Zunächst müssen wir den aktuellen Screen sichern
  scr = Screen
  Screen = obj
```

```
  ' Jetzt können wir etwas zeichnen
  graphic.mixmode = MM_INVERT
  FillEllipse 20, 20, MaxX - 20, MaxY - 20

  ' Ganz wichtig: den originalen Screen wieder herstellen
  ' auch wenn es, wie hier, ein Null-Objekt ist.
  Screen = scr

End SUB 'DrawFigure
```

5.5.3.2 Der Weg der Nutzer-Eingaben

In der täglichen Programmierarbeit ist es nicht nötig, genau zu wissen, wie die Maus- bzw. Tastatur-Ereignisse von Objekt zu Objekt weitergeleitet werden. Reagiert Ihr Programm jedoch nicht so, wie Sie es erwartet haben, ist es hilfreich, die hier besprochenen Informationen zu kennen.

Nehmen wir an, Sie haben einen visual Tree, wie er im nächsten Bild dargestellt ist. Jedes der dargestellten Objekte habe sowohl einen Handler für Mausklicks (OnMouseButton) als auch einen Handler für Tastatureingaben (OnKeyPressed). Für den Weg, den die Eingabe-Ereignisse gehen, ist es allerdings egal, ob die Objekte einen passenden Handler haben, oder nicht.



Die VisObjekte VisObj1 und VisObj2 werden innerhalb der Grenzen ihrer Parents (VisGroupObj1 und VisGroup2) dargestellt, liegen also "über" ihren Parents.

Klickt der Nutzer jetzt mit der linken Maustaste auf das Objekt VisObj1, so wird zunächst erkannt, dass das Ereignis innerhalb von MyVisContent stattfand. Danach geht es jeweils an dasjenige Child, in dessen Grenzen es stattfand. Also zunächst an GroupObj1, von dort an GroupObj2 (und nicht an VisObj3!) und schließlich an VisObj1. Folglich werden die folgenden Maushandler (falls vorhanden) in der angegebenen Reihenfolge ausgeführt:

1. MyVisContent
2. GroupObj1
3. GroupObj2
4. VisObj1

Das Objekt VisObj1 hat anschließend den Focus und das Target - vereinfacht gesagt, es wird zum "aktiven" Objekt. Das bedeutet, es erhält Tastaturereignisse direkt vom VisContent.
Betätigt der Nutzer anschließend eine Taste auf der Tastatur, so werden die folgenden Tastaturhandler in der angegebenen Reihenfolge ausgeführt:

1. VisObj1
2. MyVisContent

Diese Reihenfolge resultiert daraus, dass das VisContent - wie alle Objekte - Tastaturereignisse zuerst an seine Children weiterleitet, bevor es sie selbst behandelt.
Der Tastatur-Handler der dazwischen liegenden Objekte GroupObj1 und GroupObj2 werden nicht aktiviert, da VisObj1 die Ereignisse direkt vom VisContent erhält.

Solange nur eines der Objekte einen Tastatur- bzw. Maushandler hat, sind die Verhältnisse einfach. Es wird einfach der entsprechende Handler ausgeführt. In den meisten Fällen wird man also eine Baumstruktur wählen, in der nur die am Ende der Zweige befindlichen Objekte einen Maus- bzw. (falls erforderlich) einen Tastaturhandler haben.

Wenn CustomManageChildren aktiv ist

Im Modus customManageChildren hat der Programmierer volle Kontrolle darüber, welches Objekt sich wo befindet. Die Objekte können sich beliebig überlappen. R-BASIC stellt überlappende Objekte so dar, dass die "oben" liegenden Objekte auch die Mausereignisse erhalten. Es **wird daher dringend empfohlen** in diesem Modus alle VisObj-Objekte als direktes Child des VisContent einzubinden.

Natürlich können Sie auch hier Objekt-Trees verwenden, wenn es sinnvoll ist. Dabei gelten die oben genannten Regeln weiter, d.h. die Maus-Events werden vom Parent an die Children weitergereicht, bis sie das oben liegende Objekt erreichen. **Sie müssen hier aber selbst dafür sorgen, dass jedes Objekt vollständig innerhalb der Grenzen seines Parents dargestellt wird**, sonst erhält es möglicherweise Mausklicks nicht oder nur teilweise. Im Kapitel 5.3.2.2 (Wenn sich die Children überlappen) finden Sie ein Beispiel, das diese Situation erläutert.

5.5.3.3 Focus und Target

Im Allgemeinen ist die Arbeit mit Focus und Target etwas für fortgeschrittene Programmierer. Sie können allerdings die Focus-Hierarchie einfach benutzen, um festzustellen, welches Objekt als letztes angeklickt wurde, also das "aktive" Objekt ist. Das Beispiel "VisObj Keyboard Demo" zeigt, wie man das macht.

Dieser Abschnitt richtet sich an erfahrene Programmierer und zeigt, wie man in die Zuweisung von Focus und Target eingreifen kann. In den meisten Fällen sind die Default-Einstellungen aber völlig ausreichend und angemessen. Eine ausführliche Beschreibung zu den Themen Focus und Target finden Sie in den Kapiteln 12 (Focus und Target) und 13 (Spezielle Menüs) im Handbuch "Spezielle Themen".

Spezielle Instance-Variablen:

Variable	Syntax im UI-Code	Im BASIC-Code
grabFocusOnMouseEvents	grabFocusOnMouseEvents = **numWert**	lesen, schreiben
grabTargetOnMouseEvents	grabTargetOnMouseEvents = **numWert**	lesen, schreiben

Per Default bekommt ein VisObj-Objekt bei einem Klick mit der linken Maustaste das Target und den Focus (d.h. alle Tastatureingaben gehen an dieses Objekt, siehe Handbuch Spezielle Themen, Kapitel 12). Beim Loslassen der Maustaste behält es Focus und Target. Ein Rechtsklick auf ein anderes Objekt ändert Focus und Target auch nicht. Es sind nun Situationen denkbar, in denen dieses Verhalten nicht adäquat ist. Z.B. könnte es erwünscht sein, dass ein Linksklick auf ein bestimmtes Objekt Focus und/oder Target nicht ändert. Oder ein Objekt soll auch bei einem Rechtsklick den Focus und/oder oder das Target bekommen. Für diese Fälle bietet R-BASIC die Instancevariablen **grabFocusOnMouseEvents** und **grabTargetOnMouseEvents**. Per Default haben beide den gleichen Wert, nämlich ME_LEFT_DOWN. d.h. Focus und Target gehen an das Objekt, wenn der Nutzer mit der linken Maustaste darauf klickt. Das ist unabhängig davon, ob das Objekt einen OnMouseButton-Handler hat, oder nicht. Sie können diese Instancevariablen mit einer anderen ME_-Konstante oder eine Kombination von ME_-Konstanten belegen, z.B. mit ME_RIGHT_DOWN + ME_LEFT_DOWN. Dann bekommt das Objekt auch dann den Focus bzw. das Target, wenn der Nutzer mit der rechten Maustaste auf das Objekt klickt. Belegen Sie diese Instancevariablen mit Null, bekommt das Objekt bei einem Linksklick den Focus bzw. das Target nicht mehr.

grabFocusOnMouseEvents, grabTargetOnMouseEvents.

Syntax	UI-Code:	**grabFocusOnMouseEvents** = **value**
		value: Kombination von ME_-Konstanten
	Lesen:	**<numVar>** = **<obj>**.**grabFocusOnMouseEvents**
	Schreiben:	**<obj>**.**grabFocusOnMouseEvents** = **value**

Syntax	UI-Code:	**grabTargetOnMouseEvents** = **value**
		value: Kombination von ME_-Konstanten
	Lesen:	**<numVar>** = **<obj>**.**grabTargetOnMouseEvents**
	Schreiben:	**<obj>**.**grabTargetOnMouseEvents** = **value**

Einschränkungen:

Die Verwendung anderer Mausereignisse als den Linksklick, um Focus und Target zu ändern, ist mit bestimmten Einschränkungen verbunden. Insbesondere muss das Top-Objekt des visual Tree, das VisContent, bereits den Focus und/oder das Target haben. Das ist automatisch der Fall, wenn der Nutzer bereits mit der linken Maustaste auf das VisContent oder eins seiner Children geklickt hat. Als Programmierer kann man stattdessen die globalen Variablen Focus und Traget mit fraglichen VisContent Objekt belegen, z.B. im OnStartup-Handler.
Das Belegen der globalen Variablen Focus und Traget funktioniert in bestimmten Situationen auch innerhalb eines Maushandlers.
Unter Umständen müssen Sie auch den gesamten Focus- bzw. Target-Pfad anpassen.
Die Instancevariablen defaultFocus und defaultTarget sind leider nicht verwendbar, da sie nur für GenericClass Objekte definiert sind.

5.5.4 Spezielle Fähigkeiten und Tools

5.5.4.1 Rahmen und Anfasser

In bestimmten Situationen ist es sinnvoll, den Umriss des VisObj-Objekts oder kleine Vierecke an den Ecken des Objekts zu zeichnen, die der Nutzer direkt anklicken kann. Meist ist es so, dass diese Dinge wieder vom Schirm gelöscht werden müssen, ohne dass der Hintergrund verändert wurde. R-BASIC unterstützt beides und verwendet dabei dem MixMode MM_INVERT. In diesem Modus invertiert jede Zeichenoperation Farbpixel auf dem Schirm, unabhängig von der verwendeten Zeichenfarbe. Dadurch ist zum einen die gezeichnete Figur immer zu sehen, egal welche Hintergrundfarbe vorhanden ist und zum anderen stellt eine zweite Zeichenoperation den Hintergrund exakt wieder her.

Methoden:

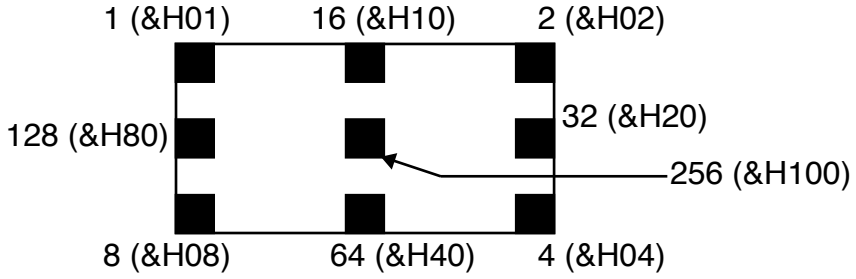
Methode	Aufgabe
DrawHandles	Anfasser zeichnen
TestHandles	Prüfen, ob Position innerhalb eines Anfassers liegt
DrawFrame	Rahmen um das Objekt zeichnen
DrawInverse	Bereich des Objekts invertiert zeichnen

DrawHandles

Diese Methode zeichnet kleine Quadrate ("Anfasser") an den Ecken oder Kanten des Objekts im MixMode MM_INVERT. Dadurch löscht eine zweite Zeichen-operation die Quadrate wieder.
Die Anfasser werden immer vom Rand in das Innere des Objekts gezeichnet, niemals über die Grenzen des Objekts hinaus. Das ist erforderlich, weil Objekte Mausklicks anfangs nur innerhalb der Grenzen des Objekts erkennen.

Syntax: **<obj>.DrawHandles hanBits** [, **hanSize**]
hanBits: Bitflags, welche Anfasser zu zeichnen sind. Siehe Grafik unten.
hanSize: Größe der Anfasser. Der Defaultwert ist 9.

Der Parameter hanBits ist eine Summe aus den Werten, die in der folgenden Grafik angegeben sind. Dabei ist für jeden Anfasser genau ein Bit gesetzt, so dass Sie die Werte beliebig kombinieren können.



Per Default sind die Anfasser 9 Pixel groß. Geben Sie einen Wert für hanSize an, wenn Sie eine andere Größe wünschen.

Beispiel: Zeichnen der 4 Anfasser an den Ecken sowie den in der Mitte

```
myObj.DrawHandles 15 + 256       ' = &H10F
```

TestHandles

Die Methode TestHandles prüft, ob die übergebene Position einem der Anfasser entspricht oder nicht. Es wird nicht geprüft, ob die Anfasser gezeichnet sind, sondern es erfolgt nur ein Positionsvergleich.
TestHandles gibt die Nummer des Anfassers entsprechend dem Bild oben zurück, oder Null, wenn die Position keinem der Anfasser entspricht.

Syntax: **<numVar>** = **<obj>**.**TestHandles** **xPos** , **yPos** [, **hanSize**]
xPos, yPos: zu prüfende Position, i.A. Mauskoordinaten
hanSize: Größe der Handles

Wenn Sie bei DrawHandles einen Parameter hanSize übergeben haben, müssen Sie den gleichen Wert auch an Testhandles übergeben.

DrawFrame

Die Methode DrawFrame zeichnet einen (gestrichelten) Rahmen um das Objekt im MixMode MM_INVERT. Dadurch löscht eine zweite Zeichenoperation den Rahmen wieder.

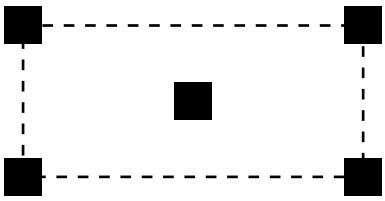
Syntax: **<obj>**.**DrawFrame** [**border** [, **lineStyle**]]
border: Abstand zum Objektrand
lineStyle: Linienstil. Der Defaultwert ist LS_DASHED

Wenn Sie einen Wert für den Parameter border angeben wird der Rahmen um die entsprechende Anzahl an Pixeln im Inneren des Objekts gezeichnet. Um einen Wert für lineStyle anzugeben (z.B. LS_DOTTED), müssen Sie ebenfalls einen Wert für border angeben, z.B. Null.

Beispiel: Zeichnen von 4 Anfassern und einem Rahmen, der sich 4 Pixel innerhalb der Grenzen des Objekts befindet. Dadurch sieht es so aus, als würden die Anfasser über das Objekt hinausragen.

```
MyObj.DrawHandles 1 + 2 + 3 + 4
MyObj.DrawFrame 4
```

Es ergibt sich das folgende Bild.



DrawInverse

Die Methode DrawInverse zeichnet ein Rechteck im Bereich des Objekts im MixMode MM_INVERT. Dadurch werden die Farben auf dem Schirm invertiert und eine zweite Zeichenoperation löscht das Rechteck wieder.

Syntax: **<obj>.DrawInverse** [**border**]
border: Abstand zum Objektrand

Wenn Sie einen Wert für den Parameter border angeben, wird an den Seiten des Objekts ein entsprechend breiter Bereich nicht invertiert.

5.5.4.2 Dragging

Ein Objekt mit der Maus auf dem Bildschirm zu platzieren ist ein häufiger Einsatzfall von VisObj-Objekten. Dazu muss im zugehörigen VisContent die Instancevariable customManageChildren auf TRUE gesetzt sein.

R-BASIC unterstützt das "Dragging" genannte Ziehen eines Objekts über den Bildschirm sehr komfortabel. Dabei wird bei Bedarf der Umriss des Objekts an der aktuellen Mausposition gezeichnet. Das ist per Default ein Rechteck, Sie können aber auch einen eigenen Umriss definieren.

Spezielle Instancevariablen:

Variable	Syntax im UI-Code	Im BASIC-Code
isDragging	—	nur lesen
dragGString	—	lesen, schreiben
dragPosition	—	nur Lesen
dragOffset	—	nur Lesen

Methoden:

Methode	Aufgabe
DragStart	Drag-Modus initialisieren
DragMoveTo	Umriss im Drag-Modus bewegen
DragEnd	Drag-Modus beenden, Objekt neu positionieren
DragAbort	Drag-Modus abbrechen

isDragging

Diese Instancevariable enthält die Information, ob sich das Objekt gerade im Drag-Modus befindet (TRUE) oder nicht (FALSE). Sie kann nur gelesen werden.

Syntax: **<numVar>** = **<obj>.isDragging**

dragGString

Per Default wird als Umriss beim Bewegen des Objekts über den Bildschirm ein gestricheltes Rechteck in der Größe des Objekts verwendet. Wenn Sie einen anderen Umriss verwenden wollen, müssen Sie die Instancevariable dragGString mit einer entsprechenden Grafik belegen. Sie sollten den GString im MixMode MM_INVERT zeichnen, weil die Dragging-Methoden des VisObj voraussetzen, dass ein zweimaliges Zeichnen des Umrisses ihn wieder vollständig vom Bildschirm entfernt.
Um zu verhindern, dass überhaupt ein Umriss gezeichnet wird, müssen Sie dragGString mit einem "leeren" GString, also einem GString ohne Zeichenoperationen, belegen.
DragGString kann nicht im UI-Code verwendet werden.

Syntax	Schreiben:	**<obj>.dragGString** = **<gs>** <gs>: Variable oder Ausdruck vom Typ Handle
	Lesen	**<gsVar>** = **<obj>.dragGString** <gsVar>: Variable vom Typ Handle

Das folgende Codefragment stammt aus dem OnMouseButton-Handler des Beispiels "VisObj Dragging Demo". Wesentliche Elemente sind zur Erhöhung der Übersichtlichkeit farblich hervorgehoben. Wichtig ist, dass die globalen Variablen MaxX und MaxY nach dem Aufruf von StartRecordGS() geändert sind, dass man den Mixmode MM_INVERT einstellt und dass man den GString am Schluss wieder freigibt. Das Belegen von dragGString mit NullHandle() stellt sicher, dass dragGString nicht auf nicht mehr existierende Daten verweist.

```
ON event SWITCH
CASE ME_LEFT_DOWN:
  sender.GrabMouse

  ' MaxX und MaxY sind nach StartRecordGS() immer 1920 x 1024
  ' Wir sichern sie in lokalen Variablen
  mx = MaxX
  my = MaxY

  ' GString aufzeichnen. Der MixMode MM INVERT ist zwingend,
  ' damit der Umriss gezeichnet werden kann
  gs = StartRecordGS()
  graphic.mixmode = MM_INVERT
```

```
    graphic.linestyle = LS_DOTTED
    Rectangle 0, 0, mx, my
    Line 0, 0, mx, my
    Line mx, 0, 0, my

    ' Aufzeichnung beenden
    EndRecordGS(gs)

    ' GString zuweisen und Dragging starten
    sender.dragGString = gs
    sender.DragStart xPos, yPos

    End CASE

  CASE ME_LEFT_UP:
    ' Dragging beenden und GString freigeben
    sender.DragEnd xPos, yPos
    FreeGS( sender.dragGString)
    sender.dragGString = NullHandle()

    sender.ReleaseMouse
    End CASE

  End SWITCH
```

dragPosition

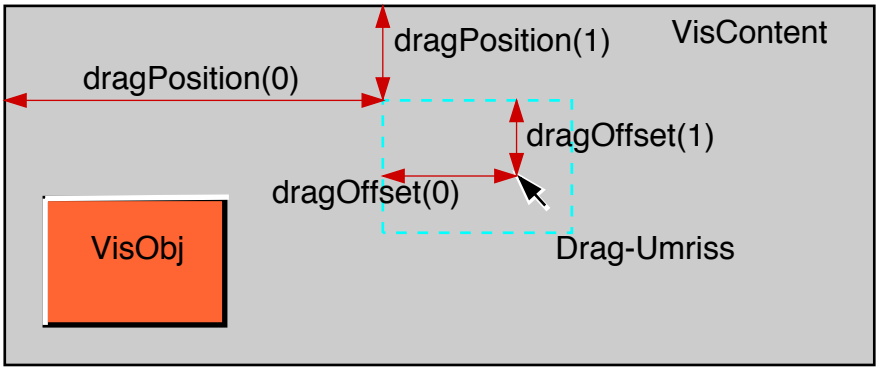
Diese Instancevariable enthält die absoluten Koordinaten (d.h. relativ zum VisContent) der linken oberen Ecke des Drag-Umrisses. Das ist im nächsten Bild dargestellt. DragPosition kann nur gelesen werden und auch nur, solange der Drag-Modus aktiv ist. Ansonsten kommt es zu einem Laufzeitfehler.

Syntax: **<numVar>** = **<obj>**.**dragPosition** (n)
n = 0: x-Koordinate lesen
n = 1: y-Koordinate lesen

dragOffset

Diese Instancevariable enthält die Position der Maus, relativ zur linken oberen Ecke des Drag-Umrisses. Dieser Wert ist konstant und wird beim Aufruf von DragStart festgelegt. Das ist im nächsten Bild dargestellt. DragOffset kann nur gelesen werden und auch nur, solange der Drag-Modus aktiv ist. Ansonsten kommt es zu einem Laufzeitfehler.

Syntax: **<numVar>** = **<obj>**.**dragOffset** (n)
n = 0: x-Offset lesen
n = 1: y-Offset lesen



DragStart

Die Methode DragStart initialisiert den Dragging-Modus für das Objekt. Dabei wird eine interne Datenstruktur initialisiert und der Umriss wird erstmalig gezeichnet. Wenn Sie also einen eigenen Umriss verwenden wollen muss die Instancevariable dragGString **vorher** belegt worden sein.
DragStart wird üblicher Weise im OnMouseButton-Handler als Reaktion auf das Ereignis ME_LEFT_HOLD oder ME_LEFT_DOWN gerufen.

Syntax: **<obj>.DragStart xPos, yPos**
xPos, yPos: aktuelle Position der Maus

DragMoveTo

Die Methode DragMoveTo bewegt dem Umriss zu einer neuen Position. Dazu wird der Umriss zunächst an der alten Position gezeichnet - womit er vom Bildschirm verschwinden sollte - und dann an der neuen Position.
DragMoveTo wird üblicher Weise im OnMouseMove-Handler gerufen. Die Methode prüft selbständig, ob sich das Objekt gerade im Drag-Modus befindet, so dass auf die Abfrage der Instancevariable isDragging verzichtet werden kann.

Syntax: **<obj>.DragMoveTo xPos, yPos**
xPos, yPos: neue Position der Maus

DragEnd

Die Methode DragEnd beendet den Drag-Modus und verschiebt das Objekt an die neue Position. Dazu wird zunächst der Umriss gezeichnet (also gelöscht), das Objekt neu positioniert und abschließend der visual Tree upgedatet.
DragStart wird üblicher Weise im OnMouseButton-Handler als Reaktion auf das Ereignis ME_LEFT_UP gerufen. Die Methode prüft selbständig, ob sich das

Objekt gerade im Drag-Modus befindet, so dass auf die Abfrage der Instancevariable isDragging verzichtet werden kann.
Falls Sie einen eigenen Umriss verwenden (Instancevariable dragGString), sollten Sie den zugehörigen GString jetzt vernichten (falls Sie ihn nicht mehr anderweitig brauchen).

Syntax: **<obj>**.**DragEnd** **xPos**, **yPos**
xPos, yPos: aktuelle Position der Maus

DragAbort

Die Methode DragAbort bricht den Drag-Modus ab, ohne das Objekt zu verschieben und ohne den visual Tree upzudaten. Es wird nur der Umriss gezeichnet (also gelöscht).
Ansonsten gelten die Hinweise, die bei DragEnd gegeben wurden.

Syntax: **<obj>**.**DragAbort**

Ein Code-Beispiel für die Verwendung von DragAbort finden Sie in der Beispiel-Datei "VisObj Level Editor Demo".

5.6 Erweiterte Möglichkeiten für SDK-Programmierer

Die Fähigkeiten der VisualClass-Objekte sind sehr komplex. R-BASIC führt die wichtigsten Fähigkeiten und Eigenschaften heraus. Erfahrene SDK-Programmierer kennen vielleicht weitere Varianten oder es fehlen ihnen bestimmte Einstellmöglichkeiten. Deswegen gibt es einige R-BASIC-Instancevariablen bzw. Methoden, die den direkten Zugriff auf die SDK-Ebene erlauben.

Instance-Variablen und Methoden für SDK-Programmierer:

Variable/Methode	Syntax im UI-Code	Im BASIC-Code
visClassAttrs	visClassAttrs = **toSet**, **toClear**	lesen, schreiben
visCompGeoAttrs	visCompGeoAttrs = **toSet**, **toClear**	lesen, schreiben
visCompDimensionAttrs	visCompDimensionAttrs = **toSet**, **toClear**	lesen, schreiben
contentAttrs	contentAttrs = **toSet** , **toClear**	lesen, schreiben
MarkInvalid2	— (Methode)	nur schreiben

Nicht alle hier besprochenen Instancevariablen und Methoden funktionieren mit allen VisualClass Objekten. Die Zuordnung finden Sie in der folgenden Tabelle

Instance-Variablen und Methoden für SDK-Programmierer:

Variable/Methode	Zugehöige Klassen
visClassAttrs	VisContent, VisObj, BitmapContent, VisText, LargeText
visCompGeoAttrs	VisContent, VisObj, BitmapContent
visCompDimensionAttrs	VisContent, VisObj, BitmapContent
contentAttrs	VisContent, BitmapContent, GenContent
MarkInvalid2	VisContent, VisObj, BitmapContent, VisText, LargeText

visClassAttrs

Die Instancevariable visClassAttrs spricht direkt die SDK-Instancevariable VI_attrs der VisClass an. Sie haben damit Zugriff auf die Bits, die von R-BASIC nicht herausgeführt werden. Der beim Schreiben benötigte Parameter updateMode sollte einer der Werte 0 (VUM_MANUAL), 1 (VUM_NOW), 2 (VUM_DELAYED_VIA_UI_QUEUE) oder 3 (VUM_DELAYED_VIA_APP_QUEUE) sein.

Syntax UI-Code: **visClassAttrs** = **attrsToSet** , **attrsToClear**
Lesen: **<numVar>** = **<obj>**.**visClassAttrs** (0)
Schreiben:
<obj>.**visClassAttrs** = **attrsToSet** , **attrsToClear** , **updateMode**
attrsToSet: zu setzende Attribute
attrsToClear: zu setzende Attribute

visCompGeoAttrs

Die Instancevariable visCompGeoAttrs spricht direkt die SDK-Instancevariable VCI_geoAttrs der VisCompClass an. Sie können alle dort definierten Bits lesen und schreiben.

Syntax UI-Code: **visCompGeoAttrs** = **attrsToSet** , **attrsToClear**
Lesen: **<numVar>** = **<obj>**.**visCompGeoAttrs** (0)
Schreiben: **<obj>**.**visCompGeoAttrs** = **attrsToSet** , **attrsToClear**
attrsToSet: zu setzende Attribute
attrsToClear: zu setzende Attribute

visCompDimensionAttrs

Die Instancevariable visCompDimensionAttrs spricht direkt die SDK-Instancevariable VCI_geoDimensionAttrs der VisCompClass an. Sie können alle dort definierten Bits lesen und schreiben.

Syntax UI-Code: **visCompDimensionAttrs** = **attrsToSet** , **attrsToClear**

Lesen: **<numVar>** = **<obj>**.**visCompDimensionAttrs** (0)
Schreiben:
<obj>.**visCompDimensionAttrs** = **attrsToSet** , **attrsToClear**
attrsToSet: zu setzende Attribute
attrsToClear: zu setzende Attribute

contentAttrs

Die Instancevariable contentAttrs spricht direkt die SDK-Instancevariable VCNI_attrs der VisContentClass an. Sie können alle dort definierten Bits verwenden. Das gilt ebenso für GenContent-Klasse. Hier heißt die Instancevariable im SDK GCI_attrs.

Syntax	UI-Code:	**contentAttrs** = **attrsToSet** , **attrsToClear**
	Lesen:	**<numVar>** = **<obj>**.**contentAttrs** (0)
	Schreiben:	**<obj>**.**contentAttrs** = **attrsToSet** , **attrsToClear**
	attrsToSet:	zu setzende Attribute
	attrsToClear:	zu setzende Attribute

MarkInvalid2

Die Methode MarkInvalid2 ist eine Erweiterung der Methode MarkInvalid. Sie ruft direkt die SDK-Message MSG_VIS_MARK_INVALID auf. Sie können genau die Flags übergeben, die Sie möchten. Die Methode MarkInvalid hingegen übergibt immer die Flags VOF_GEOMETRY_INVALID und VOF_IMAGE_INVALID.
Der Parameter updateMode sollte einer der Werte 0 (VUM_MANUAL), 1 (VUM_NOW), 2 (VUM_DELAYED_VIA_UI_QUEUE) oder 3 (VUM_DELAYED_VIA_APP_QUEUE) sein.

Syntax:	**<obj>**.**MarkInvaild2**	**<visOptFlags>**, **updateMode**