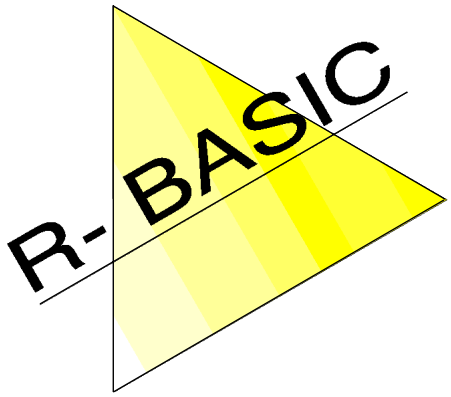
R-BASIC

Einfach unter PC/GEOS programmieren



Objekt-Handbuch

Volume 6
Text-Objekte, FileSelector

Version 1.0

(Leerseite)

Inhaltsverzeichnis

(Leerseite)

4.10 Text-Objekte

4.10.1 Überblick

Die R-BASIC Textobjekte erlauben das einfache Eingeben von Text, ohne dass Sie als Programmierer sich um irgendwelche Details kümmern müssen. Die Textobjekte behandeln Tastatur- und Mausereignisse selbständig. Sie registrieren, ob der Text vom Nutzer verändert wurde und können bei Bedarf Messages aussenden, um den Rest des Programms über bestimmten Ereignissen zu informieren.

In R-BASIC stehen vier Textobjekt Klassen zur Verfügung. Die Klasse **Memo** und **InputLine** sind GenericClass Objekte und sehr einfach zu benutzen. **VisText** und **LargeText** sind VisualClass Objekte. Sie müssen mit einem VisContent in einem View verwendet werden. Diesen Klassen ist ein eigenes Kapitel (4.10.9) gewidmet.

Memo Die Memo-Klasse stellt einen einfachen Texteditor bereit. Sie unterstützt einen automatischen Zeilenumbruch und die Entertaste beginnt einen neuen Absatz. Bei Bedarf wird ein vertikaler Rollbalken erzeugt.

InputLine Die Klasse InputLine ist für die Eingabe einzeiliger Texte, z.B. von Dateinamen, gedacht. Die Entertaste löst bei InputLine-Objekten den Apply-Handler des Objekts aus.

VisText Objekte der Klasse VisText müssen wie ein VisObj-Objekt in einem visual Tree verwendet werden. Details zu diesem Thema finden Sie im Kapitel 5.5. des Objekthandbuchs.

LargeText Der Vorteil der LargeText Objekte ist, dass sie beliebig viel Text speichern können (theoretisch bis zu 2 GByte), während die anderen Textobjekte auf 4 kByte begrenzt sind.

Die meisten Instancevariablen und Methoden sind für alle Textobjektklassen identisch, Ausnahmen sind unten erwähnt. Keins der R-BASIC Text-Objekte unterstützt die Formatierung einzelner Buchstaben, Worte oder Absätze. Alle Formatinformationen (z.B. Font, Textgröße, Textstil, Ausrichtung usw.) gelten immer für den gesamten Text des Objekts.

Da die Textobjekte Memo und InputLine von der GenericClass abstammen erben sie alle Eigenschaften, Hints und Fähigkeiten dieser Klasse. Für diese Textobjekte sind besonders die Fähigkeiten zum Geometrie-Management (Kapitel 3.3) von Bedeutung.
Die Geometrie von VisText- und LargeText-Objekten wird vom VisContent bzw. dem dazugehörigen View bestimmt, in dem sie sich befinden.

Arbeit mit dem Clipboard

Alle Textobjekte können mit der Zwischenablage (Clipboard) kommunizieren. Die Methoden (Objektanweisungen) **ClpTestCopy**, **ClpTestPaste**, **ClpCopy** und **ClpPaste** werden unterstützt. Eine detaillierte Beschreibung dieser Methoden finden Sie im Kapitel "Arbeit mit der Zwischenablage" (Kapitel 5 im Handbuch "Spezielle Themen"). Die Textobjekte verhalten sich dabei genauso wie Sie es von Textobjekten in anderen Programmen gewohnt sind. Die wichtigsten Punkte sind:

- **ClpTestCopy** liefert TRUE, wenn Text selektiert ist. Ist nichts selektiert liefert es FALSE.
- Die Methode **ClpCopy** kopiert den aktuell selektierten Text in die Zwischenablage. Ist nichts selektiert wird auch nichts in die Zwischenablage kopiert. Die globale Variable **clipboardError** wird gesetzt (TRUE: Text wurde ins Clipboard kopiert, FALSE: kein Text wurde ins Clipboard kopiert).
- Die Methode **ClpPaste** ersetzt den aktuell selektierten Text. Ist nichts selektiert wird der Text an der Cursorposition eingefügt. **Achtung! ClpPaste** setzt die globale Variable **clipboardError immer** auf FALSE. PC/GEOS liefert für Textobjekte keine Rückmeldung, ob die Operation erfolgreich war oder nicht.
- **ClpPaste** akzeptiert nur "reine" Texte. Eventuelle Formatierungen gehen verloren. Enthält der Text zu viele Zeichen oder Grafiken, so erzeugt das Objekt eine Fehlermeldung. Ist kein Text in der Zwischenablage wird die Operation ignoriert.

Keyboard-Handler

Sie können in die Behandlung von Tastaturereignissen eingreifen indem Sie einen Tastaturhandler für das Textobjekt schreiben. Dazu werden die folgenden Instancevariablen unterstützt:

Actionhandler	Instancevariablen	Methoden
OnKeyPressed	inputFlags	—

Der **OnKeyPressed**-Handler wird gerufen, wenn das Textobjekt den Focus hat und der Nutzer drückt eine Taste oder lässt sie los. Er muss als **KeyboardAction** deklariert sein. Dabei steuert **inputFlags**, ob das Objekt selbst oder der entsprechende BASIC Handler das Ereignis bearbeitet. Eine ausführliche Beschreibung der Zusammenhänge finden Sie im Kapitel 14 (Arbeit mit der Tastatur) im Handbuch "Spezielle Themen".
Insbesondere ist es wichtig zu wissen, dass das Textobjekt zunächst jedes Tastaturereignis selbst behandelt, bevor es den entsprechenden BASIC Handler aufruft. Im Kapitel 14.4 (Filtern von Tastaturereignissen) des Handbuchs "Spezielle Themen" ist beschrieben, wie man das umgehen kann.

Focus und Target

Textobjekte interagieren mit der Focus- und Target-Hierarchie. Es ist möglich zu überwachen, ob ein Text-Objekt den Focus oder das Target hat, indem man einen Focus- bzw. Target-Handler schreibt. Dazu werden die folgenden Actionhandler, Instance- und Systemvariablen unterstützt.

Actionhandler	Instancevariablen (nur Memo und InputLine)	Systemvariable
OnFocusChanged OnTargetChanged	defaultFocus targetable defaultTarget	Focus Target

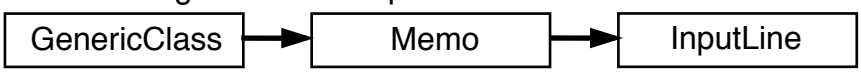
Die notwendigen Details zur Arbeit mit Focus und Target finden Sie im Kapitel 12 (Focus und Target) des Handbuchs "Spezielle Themen". Das Arbeiten mit Focus und Target ist etwas für erfahrene Programmierer und nur in wenigen Fällen notwendig. Eine Ausnahme bildet die Implementation von speziellen Menüs wie dem "Bearbeiten" Menü. Diesem Thema ist deswegen ein eigenes Kapitel ("Spezielle Themen", Kapitel 13) gewidmet.

Hinweise zu Focus und Target bei Text-Objekten:

- Die Verwendung des OnFocusChanged Handler ist nur sehr selten nötig. In den meisten Fällen, in denen man zunächst an den Focus-Handler denkt, ist es sinnvoller den OnTargetChanged Handler zu verwenden.
- Wenn Sie zum Beispiel mehr als ein Textobjekt haben wird der OnTarget-Changed Handler oft benutzt um die UI entsprechend den Attributen (Font, Textgröße, Farben usw.) anzupassen, die im Textobjekt dargestellt werden, mit dem der Nutzer gerade interagiert.
  Ein entsprechendes Beispiel finden Sie im Kapitel 4.10.6 (Text-Objekt Actions).

Die folgenden Kapitel verwenden meist Memo oder InputLine Objekte als Beispiele. Die Aussagen gelten aber immer für alle Textobjekte. Ausnahmen sind explizit erwähnt.
Außerdem wird davon ausgegangen, dass die Textobjekte im normalen Modus (nicht im sogenannten "Delayed Mode") arbeiten. Der Delayed Mode ist nur für GenericClass Objekte (Memo und InputLine) verfügbar und ausführlich im Kapitel 3.4.2 (Delayed Mode und Statusmessage) beschrieben.

Abstammung Memo und InputLine:



Abstammung VisText und LargeText:

Spezielle Instance-Variablen für alle (*) Textobjekt Klassen:

Variable	Syntax im UI-Code	Im BASIC-Code
maxLen (*)	maxLen = **numWert**	lesen, schreiben
textLen	—	nur lesen
text$	text$ = **"text"**	lesen, schreiben
cursorPos	—	lesen, schreiben
selectionEnd	—	lesen, schreiben
selectionLen	—	lesen, schreiben
fontID	fontID = **numWert**	lesen, schreiben
fontSize	fontSize = **numWert**	lesen, schreiben
textStyle	textStyle = **numWert**	lesen, schreiben
textColor	textColor = **numWert**	lesen, schreiben
backColor	backColor = **numWert**	lesen, schreiben
justifyText	justifyText = **numWert**	lesen, schreiben
lineSpacing	lineSpacing = **numWert**	lesen, schreiben
margins	margins = **left**, **right** [, **first**]	lesen, schreiben
topSpace	topSpace = **numWert**	lesen, schreiben
textFilter	textFilter = **numWert**	lesen, schreiben
textAttrs	textAttrs = **numWert**	lesen, schreiben
modified	modified = **numWert**	lesen, schreiben
OnModified	OnModified = **<Handler>**	nur schreiben
OnSelectionChanged	OnSelectionChanged = **<Handler>**	nur schreiben

(*) maxLen wird von LargeText Objekten nicht unterstützt. Die Textgröße von LargeText Objekten ist prinzipiell unbegrenzt.

Spezielle Instance-Variablen nur für GenericClass Objekte Memo und InputLine:

Variable	Syntax im UI-Code	Im BASIC-Code
TextFrame	TextFrame	—
TextNoFrame	TextNoFrame	—
SelectableIfRO	SelectableIfRO	—
ApplyHandler	ApplyHandler = **<Handler>**	nur schreiben
StatusHandler	StatusHandler = **<Handler>**	nur schreiben

Spezielle Instance-Variablen nur für LargeText:

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

Methoden für alle Textobjekt Klassen:

Methode	Aufgabe
SelectedText$	Liefert den aktuell selektierten Text
TextRange$	Liefert einen bestimmten Textbereich
Append	Text anhängen
Insert	Text einfügen
ReplaceSelection	Selektierten Text ersetzen
DeleteSelection	Selektierten Text löschen
DeleteRange	Textbereich löschen
ShowCursor	Zur Cursorposition scrollen
Suspend	Verhindert die Neudarstellung des Objekts
EndSuspend	Erlaubt die Neudarstellung des Objekts wieder

Methoden nur für Memo und InputLine:

Methode	Aufgabe
ScrollUp	Hoch scrollen
ScrollDown	Herunter scrollen
SendStatus	Status-Handler aufrufen

Action-Handler-Typen:

Handler-Typ	Parameter
TextAction	(sender as object, isModified as integer, textLen as word, selectionLen as word)

Beispiel: Ein typisches Memo Text Objekt

```
Memo Memo1
  Caption$ = "Note:"
  justifyCaption = J_TOP
  text$ ="Enter some text here ..."
  maxlen = 100
  fixedSize = 30 + ST_AVG_CHAR_WIDTH, 5 + ST_LINES_OF_TEXT
  END Object
```

Beispiel: Ein typisches InputLine Text Objekt

```
InputLine NameText
  Caption$ = "Name:"
  text$ ="Setag, Llib"
  maxLen = 100
  ExpandWidth
  ApplyHandler = ApplyNameText
  END Object
```

4.10.2 Arbeit mit dem Text

Hinweis: Stringvariablen nehmen standardmäßig nur bis zu 128 Zeichen auf, mit String(n) vereinbarte Stringvariablen können bis zu 1024 Zeichen speichern. Achten Sie beim Lesen von Text aus einem Textobjekt darauf, dass Textobjekte bis zu 4096 Zeichen enthalten können, LargeText Objekte sogar noch wesentlich mehr. Verwenden Sie die Instancevariable maxLen, um sicher zu sein, bzw. fragen Sie die Instancevariable textLen ab, wenn sie unsicher sind.

Instancevariable	Syntax im UI-Code	Im BASIC-Code
maxLen (*)	maxLen = **numWert**	lesen, schreiben
textLen	—	nur lesen
text$	text$ = **"text"**	lesen, schreiben
cursorPos	—	lesen, schreiben
selectionEnd	—	lesen, schreiben
selectionLen	—	lesen, schreiben

(*) maxLen wird von LargeText Objekten nicht unterstützt.

Methoden:

Methode	Aufgabe
SelectedText$	Liefert des aktuell selektierten Text
TextRange$	Liefert einen bestimmten Textbereich
Append	Text anhängen
Insert	Text einfügen
ReplaceSelection	Selektierten Text ersetzen
DeleteSelection	Selektierten Text löschen
DeleteRange	Textbereich löschen
ShowCursor	Zur Cursorposition scrollen
ScrollUp	Hoch scrollen
ScrollDown	Herunter scrollen
Suspend	Verhindert die Neudarstellung des Objekts
EndSuspend	Erlaubt die Neudarstellung des Objekts wieder

text$

Die Instance-Variable **text$** enthält den eigentlichen Text des Objekts. Sie kann gelesen und geschrieben werden. Das Textobjekt stellt den neuen Text automatisch dar, wenn sie der Instance-Variable **text$** einen Wert zuweisen.

Syntax	UI-Code:	**text$** = **"text"**
	Lesen:	**<stringVar>** = **<obj>**.**text$**
	Schreiben:	**<obj>**.**text$** = **"text"**
Kompatibilität:	alle Textobjekte	

maxLen

Die Instance-Variable **maxLen** enthält die maximale Länge des Textes, den das Objekt verwalten kann. Der Default-Wert liegt bei 1024, das ist die maximale Größe, die eine String-Variable in R-BASIC speichern kann. Erlaubt sind Werte von 1 bis 4096. Für LargeText Objekte kann die Textlänge nicht begrenzt werden, maxLen wird nicht unterstützt.

Syntax	UI-Code:	**maxLen** = **numWert**
	Lesen:	**<numVar>** = **<obj>**.**maxLen**
	Schreiben:	**<obj>**.**maxLen** = **numWert**
Kompatibilität:	Memo, InputLine, VisText (nicht LargeText)	

- Setzen Sie maxLen auf einen Wert, der kleiner als die aktuelle Textlänge ist, so wird der Text abgeschnitten.
- Der Nutzer kann nicht mehr Text eingeben, als durch maxLen festgelegt ist.
- Es ist immer eine gute Idee maxLen so klein wie nur möglich zu wählen. Beispielsweise ist zur Eingabe von GEOS-Dateinamen ein Wert von 32 für maxLen vernünftig, da GEOS-Dateinamen nicht länger als 32 Zeichen werden können.
- Der Text wird immer gemeinsam mit dem Textobjekt in den Speicher geladen wird. Wenn Sie maxLen zur Laufzeit (!) drastisch vergrößern und auch entsprechend viel Text abspeichern kann es in sehr ungünstigen Fällen zur Meldung "Hauptspeicher voll" kommen. Das Problem kann nicht auftreten, wenn Sie maxLen im UI-Code auf einen großen Wert setzen. Der Compiler verteilt die Textobjekte dann auf mehrere Objektblöcke. Im Kapitel 2.1.4 finden Sie Details zu diesem Problem.

textLen

Die Instance-Variable textLen enthält die Länge des aktuellen Textes.

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

Kompatibilität: Memo, InputLine, VisText (nicht LargeText)

CursorPos

Die Instancevariable cursorPos enthält die aktuelle Position der Schreibmarke (Cursor). Sie entspricht er Zeichenposition, hinter dem sich der Cursor befindet. Die Position Null entspricht dem Textanfang. CursorPos kann gelesen und geschrieben, aber nicht im UI-Code verwendet werden. Wenn Text selektiert ist enthält cursorPos die Anfangsposition des selektierten Texts. Schreiben der Cursorposition hebt eine vorhandene Textselektion auf.

Syntax Lesen: **<numVar>** = **<obj>**.**cursorPos**
Schreiben: **<obj>**.**cursorPos** = **wert**

wert: Neue Cursorposition. Null entspricht dem Textanfang.

Kompatibilität: alle Textobjekte

SelectionEnd

Die Instancevariable selectionEnd enthält die Endposition des aktuell selektierten Texts. SelectionEnd ist immer größer oder gleich cursorPos. Ist kein Text selektiert enthält selectionEnd den gleichen Wert wie cursorPos. SelectionEnd kann gelesen und geschrieben, aber nicht im UI-Code verwendet werden.

Syntax Lesen: **<numVar>** = **<obj>**.**selectionEnd**
Schreiben: **<obj>**.**selectionEnd** = **wert**
wert: Neue Endposition der Textselektion.

Kompatibilität: alle Textobjekte

SelectionLen

Die Instancevariable selectionLen enthält die Länge des aktuell selektierten Texts. SelectionEnd ist immer größer oder gleich Null. Ist kein Text selektiert enthält selectionLen den Wert Null. SelectionLen kann gelesen und geschrieben, aber nicht im UI-Code verwendet werden.

Syntax	Lesen:	**<numVar>** = **<obj>**.**selectionLen**
	Schreiben:	**<obj>**.**selectionLen** = **wert**
	wert:	Neue Länge der Textselektion. Negative Werte selektieren den Text links vom Cursor.
Kompatibilität:		alle Textobjekte

SelectedText$

Die Methode SelectedText$ liefert den aktuell selektierten Text. Ist nichts selektiert liefert sie einen leeren String. SelectedText$ kann nur gelesen werden. Um den selektierten Text zu ersetzen verwenden Sie die Methode ReplaceSelection.

Syntax	Lesen:	**<stringVar>** = **<obj>**.**SelectedText$**
Kompatibilität:		alle Textobjekte

TextRange$

Die Methode TextRange$ liefert einen bestimmten Textbereich. Der Bereich wird dabei durch die Cursorpositionen links vom zu lesenden Bereich (pos1) und die Cursorpositionen rechts vom zu lesenden Bereich. Dadurch ergeben sich die folgenden Zusammenhänge:

- Pos1 entspricht der Anzahl der zu übergehenden Zeichen links vom zu lesenden Bereich.
- Pos1 = 0 entspricht dem Lesen ab dem Textanfang, pos1 = N bedeutet: Lesen ab dem N+1. Zeichen.
- Die Anzahl der gelesenen Zeichen ergibt sich zu pos2 - pos 1.

Außerdem gilt folgendes:

- Ist pos2 größer als die Textlänge so liefert TextRange$ entsprechend weniger Zeichen.
- Sollte pos1 > pos2 sein so vertauscht R-BASIC die Werte automatisch.

Syntax	Lesen:	**<stringVar>** = **<obj>**.**TextRange$** (pos1, pos2)
	pos1, pos2:	Textbereich der gelesen werden soll.
Kompatibilität:		alle Textobjekte

Beispiele. Beachten Sie die Werte in der zweiten Zeile!

```
a$ = TextObj.TextRange$ ( 0, 10)     ' die ersten 10 Zeichen
a$ = TextObj.TextRange$ (10, 20)     ' die nächsten 10 Zeichen
a$ = TextObj.TextRange$ (100, 5000) ' alles ab Zeichen 101
```

Append

Die Methode Append hängt Text an den vorhandenen Text an.

Syntax Schreiben:	**<TextObj>.Append <StringExpression>**
StringExpression:	ein beliebiger Stringausdruck
Kompatibilität:	alle Textobjekte

Insert

Die Methode Insert fügt Text an der aktuellen Cursorposition ein. Ist Text selektiert wird am Anfang des selektierten Bereichs eingefügt. Die Selektion wird dabei aufgehoben.

Syntax Schreiben:	**<TextObj>.Insert <StringExpression>**
StringExpression:	ein beliebiger Stringausdruck
Kompatibilität:	alle Textobjekte

ReplaceSelection

Die Methode ReplaceSelection ersetzt den aktuell selektierten Text. Ist nichts selektiert wird der Text an der Cursorposition eingefügt.

Syntax Schreiben:	**<TextObj>.ReplaceSelection <StringExpression>**
StringExpression:	ein beliebiger Stringausdruck
Kompatibilität:	alle Textobjekte

DeleteSelection

Die Methode DeleteSelection löscht den aktuell selektierten Text. Ist nichts selektiert passiert nichts.

Syntax Schreiben: **<TextObj>.DeleteSelection**

Kompatibilität: alle Textobjekte

DeleteRange

Die Methode DeleteRange löscht einen bestimmten Textbereich. Der Bereich wird dabei durch die Cursorpositionen vor (pos1) und nach dem gewünschten Bereich (pos2) bestimmt. Weitere Hinweise dazu finden Sie bei der Methode TextRange$.

Syntax Lesen: **<stringVar>** = **<obj>.DeleteRange** (**pos1**, **pos2**)

pos1, pos2: Textbereich der gelöscht werden soll.

Kompatibilität: alle Textobjekte

Beispiele:

```
TextObj.DeleteRange 0, 10        ' die ersten 10 Zeichen
TextObj.DeleteRange 100, 5000    ' alles ab Zeichen 101
```

ShowCursor

Die Methode ShowCursor scrollt den Text so, dass die aktuelle Cursorposition für den Nutzer sichtbar ist.

Syntax: **<TextObj>.ShowCursor**

Kompatibilität: alle Textobjekte

ScrollDown

Die Methode ScrollDown scrollt den Text nach unten.

Syntax: **<TextObj>.ScrollDown**

Kompatibilität: GenericClass Text-Objekte: Memo, InputLine

ScrollUp

Die Methode ScrollUp scrollt den Text nach oben.

Syntax: **<TextObj>.ScrollUp**

Kompatibilität: GenericClass Text-Objekte: Memo, InputLine

Hinweis: VisText Objekte sind nicht scrollbar. Für LargeText-Objekte können Sie die Methode ScrollCmd des zugehörigen Views benutzen.

Suspend

Die Methode Suspend verhindert eine Neudarstellung, solange bis die Methode EndSuspend aufgerufen wurde. Das ist sinnvoll, wenn man mehrere Änderungen (Font, Größe usw.) vornehmen will, bzw. Text Stück für Stück hinzufügt. Dadurch wird ein Flackern verhindert.

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

Kompatibilität: alle Textobjekte

EndSuspend

EndSuspend hebt den mit Suspend gesetzten Zustand wieder auf.

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

Kompatibilität: alle Textobjekte

4.10.3 Zeichenattribute

Instancevariable	Syntax im UI-Code	Im BASIC-Code
fontID	fontID = **numWert**	lesen, schreiben
fontSize	fontSize = **numWert**	lesen, schreiben
textStyle	textStyle = **numWert**	lesen, schreiben
textColor	textColor = **numWert**	lesen, schreiben

Per Default lesen die Textobjekte Font und Größe des darzustellenden Textes aus der GEOS.INI (Kategorie [ui], Einträge "editableTextFontID" und "editable-TextFontsize"). Damit passen sie sich der GEOS-Installation des Nutzers an.

Es ist jedoch möglich, den Texten eine bestimmte Schriftart, Größe, Farbe und Stil (sogenannte Zeichenattribute) zuzuweisen. Damit werden die INI-Einträge vom Textobjekt ignoriert und der Text wird bei allen Nutzern auf die geforderte Weise dargestellt.

Dabei ist es nötig, folgendes zu wissen:

1. Sie können immer nur dem ganzen Text bestimmte Zeichenattribute zuweisen. Alle Zeichen werden dann auf diese Weise dargestellt. Die Zuweisung verschiedener Schriftarten oder Schriftgrößen zu einzelnen Teilen des Textes ist also nicht möglich.

2. Textobjekte speichern die Zeichenattribute nicht einzeln, sondern in einer gemeinsamen Datenstruktur. Weisen Sie auch nur ein einziges Zeichenattribut zu (z.B. die Textfarbe) so wird die Datenstruktur mit allen Zeichenattributen angelegt. Dabei gelten die folgenden Standardwerte für die nicht explizit zugewiesenen Attribute:
   Font: URW Sans
   Größe: 12
   Stil: normal
   Farbe: schwarz

Beispiel:

```
Memo    NotesText
  text$ = "Noch keine Notizen eingetragen."
  fontID = FID_MONO
    ' fontSize nicht gesetzt --> 12 pt wird verwendet,
    ' egal was in der INI steht
  textStyle = TS_BOLD + TS_ITALIC
  textColor = BLUE
  END OBJECT
```

fontID

Syntax	UI-Code:	**fontID** = **idCode**
	Lesen:	**<numVar>** = **<obj>**.**fontID**
	Schreiben:	**<obj>**.**fontID** = **idCode**
	idCode:	Numerischer Wert, der den Textfont bestimmt
	Kompatibilität:	alle Textobjekte

PC/GEOS identifiziert Schriften (Fonts) über eine sogenannte Font-ID-Nummer. Details dazu finden Sie im Kapitel 2, insbesondere Kapitel 2.2, des Handbuchs "Spezielle Themen".
Bitte beachten Sie, dass fontID, fontSize, textStyle und textColor vom Textobjekt in einer gemeinsamen Datenstruktur gespeichert werden. Für nicht explizit belegte Zeichenattribute wird ein Standardwert verwendet.

Namentlich verfügbare Font-ID's in R-BASIC

Name der Konstante	Wert	GEOS-Name
FID_BISON	2560	Bison (1)
FID_UNIVERSITY	513	University (1)
FID_BERKELEY	514	Berkeley (1)
FID_MONO	6656	URW Mono
FID_SANS	4608	URW Sans
FID_ROMAN	4096	URW Roman
FID_CRANBROOK	4097	Cranbrook
FID_SYMBOLPS	6144	URW SymbolPS

(1)Hinweis: Die Fonts mit den ID's FID_BISON, FID_UNIVERSITY und FID_BERKELEY sind Bitmap-Fonts, die sich nicht zur Ausgabe auf den Drucker eignen. Achtung! Die Kombination von Bitmap-Fonts mit bestimmten, nicht von diesem Font unterstützten Stilkombinationen kann zum Systemabsturz führen.

Einige weitere Font-ID's ohne vordefinierten Namen in R-BASIC:

1563	LED	(Bitmap-Font)
53006	Fat Fracture	
5632	Superb	
4612	Sather Gothic	
5123	Shattuck Avenue	

fontSize

FontSize stellt die Schriftgröße ein. Zulässig sind Werte von 4 bis 792.

Syntax	UI-Code:	**fontSize** = **size**
	Lesen:	**<numVar>** = **<obj>**.**fontSize**
	Schreiben:	**<obj>**.**fontSize** = **size**
	size:	Numerischer Wert, Schriftgröße
Kompatibilität:		alle Textobjekte

Bitte beachten Sie, dass fontID, fontSize, textStyle und textColor vom Textobjekt in einer gemeinsamen Datenstruktur gespeichert werden. Für nicht explizit belegte Zeichenattribute wird ein Standardwert verwendet.

textStyle

TextStyle stellt den Textstil ein.

Syntax	UI-Code:	**textStyle** = **style**
	Lesen:	**<numVar>** = **<obj>**.**textStyle**
	Schreiben:	**<obj>**.**textStyle** = **style**
	size:	Numerischer Wert, Textstil Kombination von TS_ Konstanten, siehe Tabelle
Kompatibilität:		alle Textobjekte

Bitte beachten Sie, dass fontID, fontSize, textStyle und textColor vom Textobjekt in einer gemeinsamen Datenstruktur gespeichert werden. Für nicht explizit belegte Zeichenattribute wird ein Standardwert verwendet.

Textstile zur Benutzung mit textStyle

Textstil	Wert	Bedeutung
TS_UNDERLINE	1	unterstrichene Schrift
TS_STRIKE_THRU	2	durchgestrichene Schrift
TS_SUBSCRIPT	4	tiefgestellte Schrift $_{\text{Schrift}}$
TS_SUPERSCRIPT	8	hochgestellte Schrift $^{\text{Schrift}}$
TS_ITALIC	16	*kursive Schrift*
TS_BOLD	32	**fette Schrift**
TS_OUTLINE	64	Wenn der Font sowohl Bitmap- und als auch Outline-Schrift enthält: Verwendung der Outline Schrift erzwingen

textColor

TextColor stellt die Textfarbe ein. Zulässig sind nur Indexfarben (0 bis 255).

Syntax	UI-Code:	**textColor** = **farbwert**
	Lesen:	**<numVar>** = **<obj>**.**textColor**
	Schreiben:	**<obj>**.**textColor** = **farbwert**
	farbwert:	Numerischer Wert, Textfarbe (0 ... 255)
Kompatibilität:		alle Textobjekte

Bitte beachten Sie, dass fontID, fontSize, textStyle und textColor vom Textobjekt in einer gemeinsamen Datenstruktur gespeichert werden. Für nicht explizit belegte Zeichenattribute wird ein Standardwert verwendet.

4.10.4 Absatzattribute

Unter Absatzattributen versteht man die Merkmale Texthintergrundfarbe, Textausrichtung, Zeilenabstand, Ränder und Absatzzwischenräume. Sie können immer nur dem ganzen Text bestimmte Absatzattribute zuweisen. Alle Absätze werden immer auf diese Weise dargestellt.

Folgende Instancevariablen stehen für die Absatzattribute zur Verfügung:

Instancevariable	Syntax im UI-Code	Im BASIC-Code
backColor	backColor = **numWert**	lesen, schreiben
justifyText	justifyText = **numWert**	lesen, schreiben
lineSpacing	lineSpacing = **numWert**	lesen, schreiben
margins	margins = **left**, **right** [, **first**]	lesen, schreiben
topSpace	topSpace = **numWert**	lesen, schreiben

Beispiel:

```
Memo    NotesText
  text$ = "Noch keine Notizen eingetragen."
  backColor = WITHE
  justifyText = J_CENTER
  topSpace = 10
  END OBJECT
```

backColor

BackColor stellt die Hintergrundfarbe des Textobjekts ein. Zulässig sind nur Indexfarben (0 bis 255).
Ist kein Wert für backColor gesetzt so wird die System-Hintergrundfarbe (z.B. Grau) verwendet. <obj>.backColor liefert dann -1.

Syntax	UI-Code:	**backColor** = **farbwert**
	Lesen:	**<numVar>** = **<obj>**.**backColor**
	Schreiben:	**<obj>**.**backColor** = **farbwert**
	farbwert:	Numerischer Wert, Hintergrundfarbe (0 ... 255)
Kompatibilität:		alle Textobjekte

justifyText

JustifyText stellt die Textausrichtung ein. Der Defaultwert ist J_LEFT.

Syntax	UI-Code:	**justifyText** = **jText**
	Lesen:	**<numVar>** = **<obj>**.**justifyText**
	Schreiben:	**<obj>**.**justifyText** = **jText**
	jText:	Numerischer Wert, Ausrichtung Zulässige Wert: Siehe Tabelle
	Kompatibilität:	alle Textobjekte

Zulässige Werte für justifyText:

Konstante	Wert	Bedeutung
J_CENTER	1	Text zentrieren
J_LEFT	2	Text linksbündig
J_RIGHT	4	Text rechtsbündig
J_FULL	32	Blocksatz

lineSpacing

LineSpacing stellt den Zeilenabstand ein. Der Defaultwert (kein Wert für lineSpacing gesetzt) ist 1.

Syntax	UI-Code:	**lineSpacing** = **abstand**
	Lesen:	**<numVar>** = **<obj>**.**lineSpacing**
	Schreiben:	**<obj>**.**lineSpacing** = **abstand**
	abstand:	Numerischer Wert vom Typ Real z.B. 1 (einzeilig), 1.5 oder 2 (zweizeilig)
	Kompatibilität:	alle Textobjekte

Hinweis: Wenn Sie "krumme" Werte (z.B. 1.15) für lineSpacing verwenden, kann es passieren, dass ein falscher Bereich selektiert wird, wenn Sie eine Textzeile mit einem Dreifachklick selektieren. Das liegt nicht an R-BASIC. Selektieren Sie dann den gewünschten Text durch Ziehen mit der Maus. Wenn die Funktion für Sie wichtig ist, müssen Sie probieren, ob der eingestellte Wert funktioniert. Für read-only Texte gibt es keine Einschränkungen.

margins

Margins stellt die Ränder für den Text ein. Dabei kann der linke Einzug für die erste Zeile unabhängig vom linken Rand gewählt werden. Lassen Sie den Parameter first weg, gilt first=left, d.h. die erste Zeile beginnt dort, wo auch alle anderen Zeilen beginnen.
Die Maßeinheit für die Ränder ist 1/12 Punkt, also sehr klein. Um einen linken Rand von 12 Punkt Breite zu erzeugen müssen Sie also einen Wert von 144 angeben.

Syntax	UI-Code:	**margins** = **left**, **right** [, **first**]
	Lesen:	**<numVar>** = **<obj>**.**margins** (n)
		n = 0: linker Rand
		n = 1: rechter Rand
		n = 2: Einzug erste Zeile
	Schreiben:	**<obj>**.**margins** = **left**, **right** [, **first**]
		left: linker Rand
		right: rechter Rand
		first: Einzug 1. Zeile (Default: wie left)
Kompatibilität:		alle Textobjekte

Beispiel:

```
Memo    NotesText
  text$ = "Noch keine Notizen eingetragen."
  margins = 144, 144  ' je 12 Punkt
  END OBJECT
```

topSpace

TopSpace (engl.: Oberer Abstand, hier Abstand über dem Absatz) stellt den zusätzlichen Platz zwischen zwei durch einen Zeilenumbruch (Entertaste) getrennten Absätzen ein.
Die Maßeinheit hierfür ist 1/12 Punkt, also sehr klein. Um einen Abstand von 12 Punkt zu erzeugen müssen Sie also einen Wert von 144 angeben.

Syntax	UI-Code:	**topSpace** = **anstand**
	Lesen:	**<numVar>** = **<obj>**.**topSpace**
	Schreiben:	**<obj>**.**topSpace** = **abstand**
		abstand: Numerischer Wert, Absatzabstand.
Kompatibilität:		alle Textobjekte

4.10.5 Textfilter

Mit Hilfe der Textfilter können Sie festlegen, dass der Nutzer nur Zeichen eingeben kann, die bestimmten Kriterien genügen. Das ist z.B. sinnvoll, wenn der Nutzer einen Dateinamen eingeben soll. Der entsprechende Textfilter stellt sicher, dass nur Zeichen eingegeben werden können, die in Dateinamen auch erlaubt sind.

Instancevariable	Syntax im UI-Code	Im BASIC-Code
textFilter	textFilter = **numWert**	lesen, schreiben

Wichtig! Textfilter wirken nur auf Zeichen, die über die Tastatur eingegeben werden. Weder der bereits vorhandene Text noch Änderungen der Instancevariable **text$** werden beeinflusst!

Syntax UI-Code: **textFilter** = **wert**
Lesen: **<numVar>** = **<obj>**.**textFilter**
Schreiben: **<obj>**.**textFilter** = **wert**

wert: Einer der Filterwerte entsprechend der Tabelle, gegebenenfalls kombiniert mit einem der Modifier-Bits entsprechend der zweiten Tabelle.

Kompatibilität: alle Textobjekte

Die folgenden Textfilter stehen zur Verfügung:

Konstante	Wert
TF_NONE	0
TF_NORMAL_ASCII	1
TF_DOS_CHARACTER_SET	2
TF_ALPHA	3
TF_ALPHA_NUMERIC	4
TF_DASHED_ALPHA_NUMERIC	5
TF_NUMERIC	6
TF_SIGNED_NUMERIC	7
TF_SIGNED_DECIMAL	8
TF_FLOAT_DECIMAL	9
TF_LEGAL_FILENAMES	10
TF_LEGAL_DOS_FILENAMES	11
TF_LEGAL_DOS_PATH	12
TF_LEGAL_DOS_VOLUME_NAMES	13
TF_DATE	14
TF_TIME	15

Zusätzlich gibt es Filter-Modifier Bits. Diese können gemeinsam mit den anderen Textfiltern verwendet werden um deren Eigenschaften zu modifizieren.

Konstante	Wert (hex)	Wert (dez)
TF_MAKE_UPPERCASE	&h100	512
TF_ALLOW_SPACES	&h200	1024
TF_NO_SPACES	&h400	2048

Beispiel:

```
InputLine NameText
  textFilter = TF_ALPHA + TF_MAKE_UPPERCASE + TF_NO_SPACES
  End Object
```

Wirkung der einzelnen Filter

TF_NONE
Das ist der Default Wert. Es ist kein spezieller Filter gesetzt.

TF_NORMAL_ASCII
Dieser Filter erlaubt nur normale ASCII-Zeichen. Erweiterte ASCII-Zeichen, also auch Umlaute, sind nicht erlaubt.

TF_DOS_CHARACTER_SET
Dieser Filter erlaubt nur Zeichen aus dem Standard-DOS-Zeichensatz.

TF_ALPHA
Dieser Filter erlaubt nur Buchstaben und Leerzeichen.

TF_ALPHA_NUMERIC
Dieser Filter erlaubt nur Buchstaben, Zahlen und Leerzeichen.

TF_DASHED_ALPHA_NUMERIC
Dieser Filter erlaubt nur Buchstaben, Zahlen und Leerzeichen sowie den Bindestrich

TF_NUMERIC
TF_SIGNED_NUMERIC
Diese Filter erlauben nur Zeichen, die in den entsprechenden numerischen Werten vorkommen können. Leerzeichen sind erlaubt.

TF_SIGNED_DECIMAL
TF_FLOAT_DECIMAL
Diese Filter erlauben nur Zeichen, die in den entsprechenden numerischen Werten vorkommen können. Leerzeichen sind nicht erlaubt. Der Dezimaltrenner hängt von den Einstellungen des Systems ab, in Deutschland ist es üblicher

Weise das Komma. Das weicht von R-BASIC Konventionen ab, hier ist der Dezimaltrenner immer ein Punkt.

TF_LEGAL_FILENAMES
Dieser Filter erlaubt nur Zeichen, die in GEOS Dateinamen vorkommen können.

TF_LEGAL_DOS_FILENAMES
Dieser Filter erlaubt nur Zeichen, die in DOS Dateinamen vorkommen können.

TF_LEGAL_DOS_PATH
Dieser Filter erlaubt nur Zeichen, die in DOS Pfaden vorkommen können. Er kann mit TF_ALLOW_SPACES kombiniert werden.

TF_LEGAL_DOS_VOLUME_NAMES
Dieser Filter erlaubt nur Zeichen, die in DOS Volumenamen vorkommen können.

TF_DATE
TF_TIME
Dieser Filter erlaubt nur Zeichen, die in Datum und Uhrzeit vorkommen können sowie Leerzeichen. Wichtig! Welche Zeichen das sind, hängt von den lokalen Einstellungen ab.

Bedeutung der Filter-Modifier Bits

TF_MAKE_UPPERCASE
Dieser Filter bewirkt, wenn er gemeinsam mit einem der anderen Filter angewendet wird, dass alle Buchstaben in Großbuchstaben umgewandelt werden. Er wirkt nicht auf Umlaute und Sonderzeichen.

TF_ALLOW_SPACES
TF_NO_SPACES
Diese Filter bestimmen, ob Leerzeichen erlaubt sind oder nicht.
TF_ALLOW_SPACES wird von VisText und LargeText nicht unterstützt.

4.10.6 Textobjekt Actions

Instancevariable	Syntax im UI-Code	Im BASIC-Code
modified	modified = **numWert**	lesen, schreiben
ApplyHandler (*)	ApplyHandler = **<Handler>**	nur schreiben
OnModified	OnModified = **<Handler>**	nur schreiben
OnSelectionChanged	OnSelectionChanged = **<Handler>**	nur schreiben

(*) ActionHandler stehen nur für die GenericClass Objekte Memo und InputLine zur Verfügung.

Action-Handler-Typen:

Handler-Typ	Parameter
TextAction	(sender as object, isModified as integer, textLen as word, selectionLen as word)

Anmerkungen zu TextAction

Die Parameter **isModified**, **textLen** und **selectionLen** sind normalerweise eine Kopie der entsprechenden Instancevariablen des Objekts. Der Zugriff auf die Parameter ist wesentlich schneller als der direkte Zugriff auf die zugehörigen Instancevariablen.
Die Ausnahme sind LargeText Objekte. Weil textLen und selectionLen vom Typ word sind, können sie in vielen Fällen die korrekten Werte für ein LargeText-Objekt nicht aufnehmen. Deswegen sind diese beiden Parameter immer Null, wenn der Handler von einem LargeText-Objekt gerufen wurde.

Anmerkung zu Focus und Target

Textobjekte gehören zu den seltenen Fällen, in denen der Programmierer die Focus- und Targethierarchie benutzt. Deswegen wegen finden Sie am Ende des Kapitels ein entsprechendes Beispiel.

Beschreibung der Instancevariablen

modified

Die Instance-Variable **modified** enthält die Information, ob der Text seit dem letzten Aussenden der Apply-Message (Aufruf des Apply-Handlers) vom Nutzer modifiziert wurde (**modified**=TRUE) oder nicht (**modified**=FALSE). Textobjekte eines neu gestarteten Programms sind zunächst ebenfalls "nicht modified".

Syntax	UI-Code:	**modified** = TRUE I FALSE
	Lesen:	**<numVar>** = **<obj>.modified**
	Schreiben:	**<obj>.modified** = TRUE I FALSE
Kompatibilität:		alle Textobjekte

Beachten Sie, dass ein Verändern des Textes vom BASIC-Code aus (z.B. Belegen der Instance-Variable **text$**), den Text 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.
Das Aussenden der Apply-Message setzt den Modified-Status zurück (**modified** = FALSE).

ApplyHandler

Die Instance-Variable **ApplyHandler** enthält den Namen des Action-Handlers, der aufgerufen wird, wenn der Text sein Änderungen anwenden will (engl. to apply: Anwenden). **Apply-Handler** müssen als **TextAction** deklariert sein.

Syntax	UI- Code:	**ApplyHandler** = **<Handler>**
	Schreiben:	**<obj>.ApplyHandler** = **<Handler>**
Kompatibilität:		Nur GenericClass Textobjekte Memo und InputLine

Beachten Sie: Der Apply-Handler wird nur aufgerufen, wenn der Text "modified", d.h. vom Nutzer verändert ist. Dies passiert automatisch, wenn der Nutzer den Text ändert, Sie können es aber auch vom BASIC Code aus machen, indem Sie die Instance-Variable modified mit TRUE belegen.

- **InputLine**-Objekte haben im Allgemeinen einen Apply-Handler. Er wird aufgerufen, wenn der Nutzer nach Eingabe eines Textes im Eingabefeld auf die Entertaste drückt.

- **Memo**-Objekte haben häufig keinen Apply-Handler.

- **VisText** und **LargeText-**Objekte können keinen Apply-Handler haben.

- Sie können ein GenericClass Textobjekt (Memo und InputLine) veranlassen, seinen Apply-Handler aufzurufen, indem Sie die von der GenericClass geerbte Methode **Apply** verwenden. Das Objekt muss aber als "modified" markiert (siehe oben). Alternativ könnte man dem Objekt auch den Hint *ApplyEvenIfNotModifie*d geben. Eine ausführliche Beschreibung dazu finden Sie im Kapitel 3.4 (Die "Apply"-Message) dieses Handbuchs.

```
SUB ForceApply ( obj as OBJECT )
  obj.modified = TRUE ' zu Sicherheit!
  obj.Apply
  END SUB
```

OnModified

Gelegentlich benötigt man eine Information, wenn ein Textobjekt durch eine Nutzereingabe vom "nicht modified" in den "modified" Zustand übergeht. Die Instance-Variable **OnModified** enthält den Namen des Action-Handlers, der aufgerufen wird, wenn das Textobjekt erstmalig nach Aussenden der letzten Apply-Message vom Nutzer verändert wird. **OnModified**-Handler müssen als **TextAction** deklariert sein.

Syntax	UI- Code:	**OnModified** = **<Handler>**
	Schreiben:	**<obj>.OnModified** = **<Handler>**
Kompatibilität:		alle Textobjekte

Das Aussenden der Apply-Message setzt den "modified" Zustand des Textobjekt zurück. Gibt der Nutzer nun Text ein, so wird der **OnModified**-Handler aufgerufen. Das heißt im Umkehrschluss, dass der **OnModified**-Handler in folgenden Fällen nicht gerufen wird:

- Setzen der **modified** Instance-Variable vom BASIC Code aus.
- Verändern des Textes vom BASIC-Code aus (z.B. Belegen der Instance-Variable **text$**).
- Die Instance-Variable modified steht bereits auf TRUE, z.B. weil sie vom BASIC-Code aus gesetzt wurde, bevor der Nutzer etwas eingegeben hat.

Beispiel: Textobjekt mit Apply- und OnModified-Handler. Der Nutzer soll einen Dateinamen eingeben und ihn durch Drücken der Entertaste im Textobjekt "anwenden" können. Oder er soll einen extra Button dazu verwenden. Dieser soll aber erst aktiv sein, nachdem der Nutzer etwas eingegeben hat. Die SUB "DoSaveFile" muss natürlich irgendwo definiert sein und wird hier nicht mit aufgeführt.

UI-Code

```
Button  SaveFileButton
  Caption$ = "Save File"
  ActionHandler = ButtonSaveFile
  enabled = FALSE     ' Zunächst inaktiv
  END Object

InputLine FileNameText
  maxLen = 32     ' max. 32 Zeichen sinnvoll
  textFilter = TF_LEGAL_FILENAMES ' ungültige Zeichen blocken
  ApplyHandler = TextSaveFile
  OnModified = TextIsModified
  END Object
```

BASIC-Code

```
ButtonAction ButtonSaveFile
  DoSaveFile ' macht die Arbeit
  END Action

TextAction TextSaveFile
  DoSaveFile ' macht die Arbeit
  END Action

TextAction TextIsModified
  SaveFileButton.enabled = TRUE    ' Button freischalten
  END Action
```

OnSelectionChanged

Wenn man ein "Edit" Menü implementieren will benötigt man die Information, ob der Nutzer Text selektiert hat oder nicht und ob er den Selektionsstatus verändert. Die Instancevariable **OnSelectionChanged** enthält den Namen des Actionhandlers, der aufgerufen wird, wenn der Nutzer zwischen "nichts selektiert" und "etwas selektiert" wechselt. **OnSelectionChanged** Handler müssen als **TextAction** deklariert sein. Um herauszufinden, ob Text selektiert ist oder nicht sollten Sie den Parameter "selectionLen" abfragen. Bei LargeText Objekten müssen Sie die Instancevariable selectionLen direkt abfragen, weil der an den Handler übergebene Parameter selectionLen hier immer Null ist.

Syntax	UI- Code:	**OnSelectionChanged** = **<Handler>**
	Schreiben:	**<obj>**.**OnSelectionChanged** = **<Handler>**
Kompatibilität:		alle Textobjekte

Beispiel: Einen "Kopieren" Button im Edit Menü verwalten.

UI-Code

```
Button  CopyButton
  Caption$ = "Kopieren"
  ActionHandler = DoCopyText  ' woanders implementiert
  enabled = FALSE     ' Anfangs inaktiv
  END Object

Memo InfoText
  OnSelectionChanged = HandleSelection
  END Object
```

BASIC-Code

```
TextAction HandleSelection
  IF selectionLen THEN            ' d.h. selectionLen <> 0
    CopyButton.enabled = TRUE
  ELSE
    CopyButton.enabled = FALSE
  END IF
END Action
```

Beispiel für die Verwendung des OnTargetChanged-Handlers

Wenn das Programm wissen muss, welches Text-Objekt gerade aktiv ist, bietet sich der OnTargetChanged-Handler an. Im Beispiel wird ein Number-Objekt verwendet, um die Größe des Fonts im aktiven Text-Objekt anzuzeigen.

UI-Code:

```
Memo    Text1
  fontSize = 14 : fontID = FID_MONO
  defaultFocus        ' ja, Focus
  OnTargetChanged = HandleTarget
End Object

Memo    Text2
  fontSize = 24 : fontID = FID_SANS
  OnTargetChanged = HandleTarget
End Object

Number PointInfoNumber
  Caption$ = "Aktuelle Font Größe:"
End Object
```

BASIC-Code

```
TargetAction HandleTarget
  if state = FALSE THEN RETURN  ' Target verloren? Ignorieren.
  PointInfoNumber.value = sender.fontSize  ' UI updaten
End Action
```

4.10.7 Verhalten und Aussehen von Textobjekten

Um das Aussehen und das Verhalten der GenericClass Textobjekte Memo und InputLine in bestimmten Situationen zu steuern, stehen die folgenden Instancevariablen zur Verfügung.
VisText und LargeText Objekte werden in einem visual Tree verwaltet. Für diese Objekte steht nur textAttrs mit genau einem Attribut zur Verfügung.

Instancevariable Memo, InputLine	Syntax im UI-Code	Im BASIC-Code
TextFrame	TextFrame	—
TextNoFrame	TextNoFrame	—
SelectableIfRO	SelectableIfRO	—
textAttrs (*)	textAttrs = **numWert**	lesen, schreiben

(*) textAttrs steht mit Einschränkungen auch für VisText und LargeText-Objekte zur Verfügung.

TextFrame

Der Hint TextFrame bewirkt, dass das Textobjekt immer mit einem Rahmen gezeichnet wird. Read-Only Textobjekte haben normalerweise keinen Rahmen.

Syntax UI-Code: **TextFrame**

Kompatibilität: Nur GenericClass Textobjekte Memo und InputLine

TextNoFrame

Der Hint TextNoFrame bewirkt, dass das Textobjekt immer ohne Rahmen gezeichnet wird. Editierbare Texte haben normalerweise einen Rahmen.

Syntax UI-Code: **TextNoFrame**

Kompatibilität: Nur GenericClass Textobjekte Memo und InputLine

SelectableIfRO

Der Hint SelectableIfRO (selektierbar, auch wenn Read-Only) bewirkt, dass der Nutzer read-only Texte mit der Maus selektieren kann. Drag und Drop ist dann möglich.

Syntax UI-Code: **SelectableIfRO**

Kompatibilität: Nur GenericClass Textobjekte Memo und InputLine

textAttrs

Die Instancevariable textAttrs enthält eine Reihe von Flagbits, die das Verhalten und die Eigenschaften des Textobjekts in bestimmten Situationen bestimmen. Jedes Bit hat eine eigene Bedeutung. Die verschiedenen Werte können mit + oder OR verknüpft werden. Die folgende Tabelle enthält die für textAttrs definierten Bitwerte. Hier nicht angegebene Werte sind intern reserviert und sollten nicht benutzt werden. Per Default ist das Bit TA_USE_TAB_FOR_NAVIGATION für Memo und InputLine gesetzt.
VisText und LargeText unterstützen nur genau ein Attribut, nämlich TA_USE_TAB_FOR_NAVIGATION. Dieses Bit ist für VisText und LargeText per Default aber **nicht** gesetzt.

Hinweis: R-BASIC unterstützt die im PC/GEOS-SDK definierten Attribute. Ob ein bestimmtes Attribut oder eine Attributkombination im konkreten Fall wirkt hängt manchmal von den konkreten Umständen ab. Beispielsweise setzen einige Flags (z.B. TA_SELECT_TEXT) setzen voraus, dass der Hint defaultFocus gesetzt ist. Oder der Hint fixedSize verhindert trotz gesetzten TA_NEVER_SCOLLABLE, dass das Textobjekt seine Größe an den aktuellen Text anpasst.

Konstante	Wert (hex)	Wert (dez)
TA_DONT_SCROLL_TO_CHANGES	&h02	2
TA_TAIL_ORIENTED	&h04	4
TA_ALLOW_TEXT_OFF_END	&h08	8
TA_NO_WORD_WRAPPING	&h10	16
TA_INIT_SCROLLING	&h20	32
TA_USE_TAB_FOR_NAVIGATION	&h40	64
TA_NEVER_SCROLLABLE	&h100	256
TA_SELECT_TEXT	&h200	512
TA_CURSOR_AT_START	&h400	1024
TA_CURSOR_AT_END	&h800	2048

Syntax UI-Code: **textAttrs** = **wert**
Lesen: **<numVar>** = **<obj>**.**textAttrs**
Schreiben: **<obj>**.**textAttrs** = **wert**

wert: Kombination aus den Bitwerten entsprechend der Tabelle. Für VisText und LargeText wird nur das Bit TA_USE_TAB_FOR_NAVIGATION unterstützt. Alle anderen Werte werden ignoriert.

Kompatibilität: alle Textobjekte

Beispiele:

```
Memo InfoText
  textAttrs = TA_INIT_SCROLLING OR TA_USE_TAB_FOR_NAVIGATION
    ' Statt + kann man auch die logische Operation OR
    ' verwenden
  End Object

InputLine OtherText
  defaultFocus
  textAttrs = TA_SELECT_TEXT
    ' Weil TA_USE_TAB_FOR_NAVIGATION nicht angegeben ist
    ' wird die Tabulatortaste nicht zum Anspringen des
    ' nächsten UI-Objekts verwendet.
  End Object
```

Bedeutung der einzelnen Flagbits:

TA_DONT_SCROLL_TO_CHANGES
Per Default scrollen Textobjekte automatisch zu der Position, an der Änderungen am Text, z.B. Einfügen oder Löschen, vorgenommen werden. Dieses Flag schaltet dieses Verhalten aus.

TA_TAIL_ORIENTED
Dieses Flag sollte gesetzt sein, wenn Sie möchten, dass das Objekt das Textende anzeigt (engl. tail: Schwanz), statt den Beginn des Textes. In einer scrollbaren Textbox bewirkt es, dass Text, der am Ende angefügt wird, immer angezeigt wird.

TA_ALLOW_TEXT_OFF_END
Dieses Flag ermöglicht es, dass der Text größer ist, als von der Textbox dargestellt werden kann, ohne dass die Textbox vertikale oder horizontale Rollbalken erzeugt.

TA_NO_WORD_WRAPPING
Dieses Flag schaltet den Wortweisen Zeilenumbruch (Word wrapping) aus. Stattdessen erfolgt ein Zeilenumbruch auch mitten im Wort.

TA_INIT_SCROLLING
Dieses Flag bewirkt, dass der Text immer Rollbalken hat, auch wenn dies nicht erforderlich ist.

TA_USE_TAB_FOR_NAVIGATION
Dieses Flag zeigt an, dass die Tabulatortaste zum Anspringen des nächsten UI-Objekts benutzt werden soll, statt ein Tabulatorzeichen im Text zu speichern. Für Memo und InputLine ist es das einzige Flag, dass per Default gesetzt ist.
Für VisText und LargeText ist es das einzige unterstütze Flag und es ist per Default nicht gesetzt.

TA_NEVER_SCROLLABLE
Dieses Flag bewirkt, dass das Textobjekt bei länger werdendem Text seine Größe ändern soll, statt Rollbalken zu erzeugen.

TA_SELECT_TEXT
Dieses Flag bewirkt, dass Textobjekte, die den Hint **defaultFocus** gesetzt haben, ihren Text zum Programmstart komplett selektieren sollen. Das erleichtert es dem Nutzer, ihn komplett zu ersetzen.

TA_CURSOR_AT_START
Dieses Flag bewirkt, dass Textobjekte, die den Hint **defaultFocus** gesetzt haben, den Cursor am Textanfang positionieren.

TA_CURSOR_AT_END
Dieses Flag bewirkt, dass Textobjekte, die den Hint **defaultFocus** gesetzt haben, den Cursor am Textende positionieren.

4.10.8 Textobjekte im Delayed Mode

Die Textobjekte Memo und InputLine 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>**

Kompatibilität: Nur GenericClass Textobjekte Memo und InputLine

Der **StatusHandler** wird im Delayed Mode statt des ApplyHandlers gerufen, wenn der Nutzer z.B. nach Eingabe eines Textes in einem InputLine-Objekt auf die Entertaste drückt. 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 der Text vom User modifiziert wurde, der ApplyHandler aber noch nicht gerufen wurde. Der Aufruf des ApplyHandlers setzt auch im Delayed Mode den modified-Status zurück.

Methode	Aufgabe
SendStatus	Status-Handler aufrufen

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

Kompatibilität: Nur GenericClass Textobjekte Memo und InputLine

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

4.10.9 VisText und LargeText

Die VisualClass Objekte VisText und LargeText müssen als Children eines VisContent in einem View verwendet werden. VisText-Objekte dienen dazu, Text gemeinsam mit Grafikelementen anzuzeigen oder einzugeben. LargeText-Objekte müssen Sie verwenden, wenn Sie große Textmengen (mehr als 4000 Zeichen) darstellen oder bearbeiten wollen.
Für alle anderen Zwecke sollten Sie die GenericClass-Objekte Memo oder InputLine verwenden.

4.10.9.1 VisText

VisText-Objekte werden ähnlich wie VisObj-Objekte verwendet. Die notwendigen Informationen dazu finden Sie in den Kapiteln 5.3 (VisGroup), 5.4 (VisContent) und 5.5 (VisObj). Sie können Objekte der Klassen VisObj und VisText innerhalb eines Visual Tree beliebig mischen. Allerdings können VisText-Objekte keine Children haben. Deswegen werden die diesbezüglichen Instancevariablen von VisText-Objekten nur unterstützt, wenn sie die eigene Größe betreffen.

Zusätzlich zu den Text-spezifischen Instancevariablen unterstützen VisText-Objekte die folgenden Instancevariablen:

Eigene Instancevariablen

Variable	Syntax im UI-Code	Im BASIC-Code
visTextFrame	visTextFrame = width [, col [, dis [, st]]]	lesen, schreiben
visTextFrameOptions	visTextFrame = yAdd [, fill]	lesen, schreiben

Einige der bei der VisGroup Class (Kapitel 5.3) bzw. der VisObj Class (Kapitel 5.5) beschriebenen Instancevariablen sind auch für VisText-Objekte verfügbar. Ihre wesentlichen Eigenschaften werden im Folgenden kurz beschrieben.

Variable	Syntax im UI-Code	Im BASIC-Code
drawable	drawable = TRUE | FALSE	lesen, schreiben
detectable	detectable = TRUE | FALSE	lesen, schreiben
managed	managed = TRUE | FALSE	lesen, schreiben
visPosition	visPosition = xPos, yPos	lesen, schreiben
xPosition, yPosition	—	nur lesen
visSize	visSize = width, height	lesen, schreiben
xSize, ySize	—	nur lesen

visTextFrame

Die Instancevariable visTextFrame legt fest, ob ein Rahmen um das Objekt gezeichnet werden soll. Das entspricht in etwa der Instancevariablen TextFrame bei GenericClass Textobjekten.

Syntax UI-Code: **visTextFrame** = **width** [, **color** [, **dist** [, **style**]]
Lesen: **<numVar>** = **<obj>**.**visTextFrame** (0) ' width
<numVar> = **<obj>**.**visTextFrame** (1) ' color
<numVar> = **<obj>**.**visTextFrame** (2) ' dist
<numVar> = **<obj>**.**visTextFrame** (3) ' style
Schreiben: **<obj>**.**visTextFrame** = **width** [, **color** [, **dist** [, **style**]]

width: Breite des Rahmens. Defaultwert: 0 (kein Rahmen)
color: Farbe des Rahmens. Defaultwert: 0 (BLACK)
dist: Abstand des Rahmens vom Text-Objekt. Defaultwert: 0
style: Linienstil, siehe Tabelle. Defaultwert: LS_SOLID

Alle Parameter sind vom Datentyp Byte. Beachten Sie, dass der Rahmen außerhalb des Objekts gezeichnet wird. Ein 2 Pixel breiter Rahmen mit einem Abstand (dist) von 1 vergrößert den Platzbedarf des Objekts um 3 Pixel in jede Richtung.

Erlaubte Linienstile für den Parameter style:

Wert	Konstante	Bedeutung
0	LS_SOLID	durchgehend
1	LS_DASHED	gestrichelt
2	LS_DOTTED	gepunktet
3	LS_DASHDOT	Strich-Punkt
4	LS_DASDDOT	Strich-Doppelpunkt

visTextFrameOptions

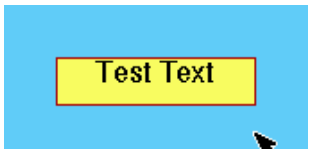
Die Instancevariable visTextFrameOptions modifiziert den Rahmen (visText-Frame) um ein VisText Objekt.
Mit dem Parameter yAdd wird die obere Kante des Rahmens um yAdd Pixel nach oben verschoben. Damit können Sie einem umrahmten Text ein gefälligeres Aussehen geben. Der Rahmen wird dadurch höher, die untere Kante wird nicht verschoben. yAdd ist vom Datentyp Byte.
Wenn Sie den Parameter fill auf TRUE setzen, dann wird der Bereich zwischen dem Objekt und dem Rahmen mit der Text-Hintergrundfarbe gefüllt. Ansonsten ist dieser Bereich transparent. Die Einstellung fill = TRUE ist nur sinnvoll, wenn mit visTextFrame oder mit dem Parameter yAdd ein Abstand zwischen dem Rahmen und dem Objekt eingestellt wurde und sich das Objekt vor einem Hintergrund mit einer abweichenden Farbe befindet.

Syntax	UI-Code:	**visTextFrameOptions** = **yAdd** [, **fill**]
	Lesen:	**<numVar>** = **<obj>**.**visTextFrameOptions** (0) ' yAdd
		<numVar> = **<obj>**.**visTextFrameOptions** (1) ' fill
		<numVar> = **<obj>**.**visTextFrame** (2) ' dist
		<numVar> = **<obj>**.**visTextFrame** (3) ' style
	Schreiben:	**<obj>**.**visTextFrame** = **yAdd** [, **fill**]

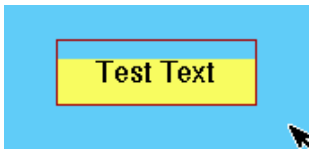
yAdd: Verschiebung der oberen Rahmenkante nach oben.
Defaultwert: 0 (keine Verschiebung)

fill: Füllen des Bereichs zwischen Objekt und Rahmen mit der Text-Hintergrundfarbe
Defaultwert: FALSE (keine Füllung)

Beispiele für visTextFrame und visTextFrameOptions:



visTextFrame = 1, RED



visTextFrame = 1, RED
visTextFrameOptions = 10

visTextFrame = 1, RED
visTextFrameOptions = 10, TRUE

drawable

Die Instancevariable drawable bestimmt, ob das Objekt auf den Bildschirm gezeichnet wird oder nicht. Das entspricht in etwa der Instancevariablen visible bei GenericClass Objekten. Allerdings wird das Objekt weiterhin bei der Berechnung der Geometrie berücksichtigt, auch wenn drawable auf FALSE gesetzt ist.
Siehe auch: Kapitel 5.5.2 VisObj: Grundlegende Fähigkeiten

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

detectable

Die Instancevariable detectable bestimmt, ob das Objekt auf Maus- und Tastaturereignisse reagieren soll, oder nicht. Das entspricht in etwa der Instancevariablen readOnly bei GenericClass Objekten.
Siehe auch: Kapitel 5.5.2 VisObj: Grundlegende Fähigkeiten

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

managed

Die Instancevariable managed legt fest, ob die Position des Objekts vom Geometriemanager verwaltet werden soll, oder nicht. 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 auf TRUE lassen, wenn das zugehörige VisContent im Modus customManageChildren arbeitet.
Siehe auch: Kapitel 5.5.2 VisObj: Grundlegende Fähigkeiten

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

visPosition

Die Instance-Variable visPosition enthält die aktuelle Position des Objekts, relativ zu seinem VisContent.
Siehe auch: 5.3.2.1 VisGroup: Größe und Position

Syntax	UI-Code:	**visPosition** = **xPos**, **yPos**
		xPos: x-Position
		yPos: y-Position
	Lesen:	**<numVar>** = **<obj>**.**visPosition**(0) ' xPos
		<numVar> = **<obj>**.**visPosition**(1) ' yPos
	Schreiben:	**<obj>**.**visPosition** = **xPos**, **yPos** [, autoRedraw]
		autoRedraw:
		FALSE (Default): keine sofortige Neudarstellung
		TRUE: sofortige Neudarstellung (Move-To-Funktion)

visSize

Die Instance-Variable visSize enthält die aktuelle Größe des Objekts.

Wichtig: **Sie müssen einen Wert für visSize festlegen.**
VisText-Objekte haben immer eine feste Größe. Sie verfügen nicht über die Fähigkeit, ihre eigene Größe der eingegebenen Textmenge anzupassen. Sie erzeugen auch keine Rollbalken. Wenn Sie mehr Text eingeben oder anzeigen, als in die vorgegebene Größe passt, wird der überschüssige Text nicht dargestellt.
Siehe auch: 5.3.2.1 VisGroup: Größe und Position

Syntax UI-Code: **visSize** = **width**, **height**
width: Breite
height: Höhe
Lesen: **<numVar>** = **<obj>**.**visSize**(0) ' Breite
<numVar> = **<obj>**.**visSize**(1) ' Höhe
Schreiben: **<obj>**.**visSize** = **width**, **height** [, autoRedraw]
autoRedraw:
FALSE (Default): keine sofortige Neudarstellung
TRUE: sofortige Neudarstellung

xPosition, yPosition

Diese Werte liefern die aktuelle Position des Objekts.
Siehe auch: 5.3.2.1 VisGroup: Größe und Position

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

xSize, ySize

Diese Werte liefern die aktuelle Größe des Objekts in Pixeln.
Siehe auch: 5.3.2.1 VisGroup: Größe und Position

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

Das folgende Codefragment aus dem Beispiel "VisText Demo 2" im Ordner "Beispiel\Objekte\VisText und LargeText" zeigt, wie man einen VisText in einen VisualTree einbinden kann.

```
View DemoView
  Content = DemoContent
  initialSize = 400, 220
  < weitere Instancevariablen >
End OBJECT

VisContent DemoContent
  Children = VisObj1, VisText1, VisObj2, VisText2
  < weitere Instancevariablen >
End OBJECT

VisObj VisObj1
  visSize = 120, 120
  OnDraw = VisObjDraw     ' muss irgendwo implementiert sein
End OBJECT
```

```
VisText VisText1
  text$ = "Raumschiff\rEnterprise"
  visSize = 150, 50
  fontID = FID_UNIVERSITY
  fontSize = 16
End OBJECT
```

4.10.9.2 LargeText

LargeText Objekte bringen nur eine einzige eigene Instancevariable mit.

Variable	Syntax im UI-Code	Im BASIC-Code
approxSize	approxSize = numWert	lesen, schreiben

Diese Variable ist per Default so vorbelegt, dass sie nur in sehr seltenen Fällen geändert werden muss.

View- und Content Setup

Um mit einem LargeText zusammen zu arbeiten, müssen sowohl das VisContent als auch das zugehörige View auf spezielle Weise initialisiert werden. Hierfür wird die Instancevariable holdsLargeText auf den Wert TRUE gesetzt.
Ein LargeText muss das einzige Child des VisContent sein, sonst wird es eventuell nicht angezeigt.

Ein typisches Setup für eine View/Content-Kombination mit einem LargeText sieht so aus:

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

  holdsLargeText=TRUE

  ' nicht erforderlich, aber häufig verwendet
  defaultTarget
  defaultFocus
  targetable = TRUE
  viewAttrs = VA_CONTROLLED, 0     ' Setzen, wenn ein ViewControl
                                   ' verwendet werden soll
End OBJECT

VisContent DemoContent
  holdsLargeText=TRUE
  Children = MyLargeText
End OBJECT

LargeText MyLargeText
  text$ = "Ich kann ganz viel Text aufnehmen."
  fontID = FID_SANS
  fontSize = 14
End OBJECT
```

holdsLargeText

Die Instancevariable holdsLargeText initialisiert ein View bzw. ein VisContent Objekt so, dass sie mit einem LargeText Objekt zusammenarbeiten können.

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

Die Instancevariable kann nicht gelesen werden, weil intern eine ganze Palette von Werten geändert werden. Dazu gehört, dass in den Instancevariablen viewAttrs (View) bzw. contentAttrs (VisContent) einzelne Bits gesetzt werden. Sie können diese Instancevariablen trotzdem "ganz normal" verwenden, da die Bits unabhängig voneinander behandelt werden. Ausnahme: die Instancevariable inputOptions des View sollten Sie nicht verändern.

LargeText Instance Variablen

LargeText Objekte arbeiten intern mit 32-Bit Koordinaten. Das hat Auswirkungen auf die Verwendbarkeit bzw. Bedeutung einiger geerbter Instancevariablen bzw. Handler-Parameter vom Typ WORD bzw. INTEGER (16 Bit).

- Die von der VisTextClass geerbten Instancevariablen **drawable**, **detectable**, **managed**, **visPosition** und **visSize** werden nicht unterstützt, da sie entweder intern verwaltet werden oder vom Datentyp WORD sind.
- Die für alle Klassen definierten word-großen Instancevariablen xPosition, yPosition, xSize und ySize liefern immer den Wert Null.
- Die Parameter **textLen** und **selectionLen** der OnModified- und OnSelection-Changed- Handler sind vom Typ INTEGER und daher bedeutungslos. Sie werden immer mit dem Wert Null belegt.

approxSize

Die Instancevariable approxSize enthält die ungefähre Textmenge in Kilobyte, die zu erwarten ist. Der Wert ist nicht kritisch, er kann problemlos überschritten werden. Der Defaultwert beträgt 400 (Kilobyte) und sollte für die meisten Anwendungen ausreichend sein.

Syntax	UI-Code:	**approxSize** = **numWert**
	Schreiben:	**<obj>.approxSize** = **numWert**
	Lesen:	**numVar** = **<obj>.approxSize**
	numWert:	Ungefähr zu erwartenden Textmenge in Kilobyte. Der Defaultwert ist 400. Verwenden Sie Wert 10000 wenn Sie 10 MB oder mehr benötigen.

Hinweise

- LargeText Objekte speichern ihren Text in einer von R-BASIC verwalteten temporären Datei. Da sowohl einige andere Objekte als auch GStrings temporäre Dateien benutzen, verwendet R-BASIC diesen Wert, um bei Bedarf eine weitere Datei anzulegen.
- Wenn Sie den Wert 10000 (entsprechend 10 MB) verwenden, legt R-BASIC eine Datei exklusiv für dieses Objekt an. Damit kann der Text theoretisch bis 2 Gigabyte groß werden.
- Es wird empfohlen, den Wert, wenn überhaupt, nur im UI-Code zu verwenden. Das Ändern von approxSize zur Laufzeit ist ein aufwändiger Prozess.

Hintergrundinformationen

Die folgenden Informationen sind für die normale Verwendung eines LargeText Objekts nicht erforderlich.
Im Detail passiert beim Belegen der Instancevariablen holdsLargeText folgendes:

View-Objekt

Die Instancevariablen werden so gesetzt, als würde im UI-Code folgendes stehen:

```
viewAttrs = VA_DRAG_SCROLLING , 0
inputOptions = VIO_DONT_SEND_KBD_RELEASES
focusable = TRUE
```

Außerdem wird in den viewAttrs das in R-BASIC nicht verfügbare Bit &h100 (im SDK: GVA_WINDOW_COORDINATE_MOUSE_EVENTS) gesetzt. Wenn Sie holdsLargeText zur Laufzeit auf FALSE setzen wird nur dieses Bit zurückgesetzt, die anderen Instancevariablen werden nicht verändert.

Hinweis: Zusätzliche viewAttrs können Sie im UI-Code oder zur Laufzeit an beliebiger Stelle setzen, da die Bits immer einzeln behandelt werden. Zusätzliche inputOptions sollten Sie nicht setzen.

VisContent-Objekt

Die Instancevariablen werden so gesetzt, als würde im UI-Code folgendes stehen:

```
contentAttrs = CA_SAME_WIDTH_AS_VIEW , 0
customManageChildren = TRUE
```

Außerdem werden in den contentAttrs die in R-BASIC nicht mit einer Konstante belegten Bits &h20 (im SDK: VCNA_LARGE_DOCUMENT_MODEL) und &h10 (im SDK: VCNA_WINDOW_COORDINATE_MOUSE_EVENTS) gesetzt.
Darüber hinaus werden weitere, in R-BASIC nicht verfügbare, Instancevariablen verändert. Wenn Sie holdsLargeText zur Laufzeit auf FALSE setzen werden sowohl die internen Instancevariablen zurückgesetzt, also auch die contentAttrs &h20 und &h10. Die anderen Instancevariablen werden nicht geändert.

Hinweis: Zusätzliche contentAttrs können Sie im UI-Code oder zur Laufzeit an beliebiger Stelle setzen, da die Bits immer einzeln behandelt werden. Beachten Sie, dass die oben genannten Bits der contentAttrs auch von R-BASIC aus verändert werden können.

4.10.10 Text in Dateien speichern

Alle Textobjekte können ihren Text direkt in eine Datei schreiben und direkt aus einer Datei lesen. Das ist insbesondere für LargeText-Objekte interessant. Dazu werden die folgenden Methoden unterstützt.

Methode	Aufgabe
WriteToFile	Text in eine Datei schreiben
ReplaceFromFile	Text aus einer Datei ersetzen
InsertFromFile	Text aus einer Datei einfügen
FileTextSize	Textumfang in einer Datei ermitteln
WriteToVMFile	Text in eine VM-Datei schreiben
ReplaceFromVMFile	Text aus einer VM-Datei ersetzen
InsertFromVMFile	Text aus einer VM-Datei einfügen
VMFileTextSize	Textumfang in einer VM-Datei ermitteln

Die Methoden WriteToFile, ReplaceFromFile, InsertFromFile und FileTextSize arbeiten mit normalen DOS-Dateien bzw. gleichwertig mit GEOS-DATEN-Dateien. Dabei kann gleichzeitig der Zeichensatz konvertiert werden (z.B. GEOS nach Windows), da im Hintergrund die Convert$-Funktion aufgerufen wird.

Die Methoden WriteToVMFile, ReplaceFromVMFile, InsertFromVMFile und VM-FileTextSize arbeiten mit GEOS VM-Dateien. VM-Dateien sollten Sie verwenden, wenn Sie mehr als nur einen Text (z.B. mehrere unabhängige Texte oder Text und Bilder) in einer einzigen Datei speichern wollen. Alle "großen" Applikationen (z.B. GeoWrite, GeoDraw, R-BASIC) speichern ihre Dokumente in VM-Dateien.
Um diese Methoden verwenden zu können, müssen Sie die Library "VMFiles" includen. Diese Library kann separat von der R-BASIC Webseite heruntergeladen werden.

WriteToFile

WriteToFile schreibt den vom Textobjekt dargestellten Text in eine offene DOS- oder GEOS-Daten-Datei. Sie können auswählen, ob die Zeichen dabei in einen anderen Zeichensatz (z.B. DOS oder HTML) konvertiert werden sollen und ob der gesamte Text oder nur Teile davon in der Datei gespeichert werden sollen.
WriteToFile schreibt immer ab der aktuellen Dateiposition, vorhandene Daten werden überschrieben. Bei Bedarf wird die Datei verlängert.

Syntax: **<obj>**.**WriteToFile** fh [, convertMode [, start [, end]]]

fh: Variable oder Ausdruck vom Typ FILE. Die Datei muss offen sein.
convertMode: Bestimmt, zwischen welchen Zeichensätzen konvertiert werden soll. Siehe unten.
Defaultwert: Null (keine Konvertierung)
start: Cursorposition, ab der geschrieben werden soll. Die Zählung beginnt bei Null.
Defaultwert: Null (von Anfang an)
end: Cursorposition, bis zu der geschrieben werden soll. Die Zählung beginnt bei Null. End darf größer als die Textlänge sein.
Defaultwert: 4 294 967 294 (alles)

Hinweise:

- Für den Parameter convertMode sind alle Werte zugelassen, die auch für die Funktion Convert$ zugelassen sind. Als Ersatzzeichen für nicht konvertierbare Zeichen wird immer der Unterstrich '_' verwendet.
  Eine Beschreibung der Convert$-Funktion finden Sie im Kapitel 2.4.3 (Konvertierungsfunktionen) des Programmierhandbuchs.
- WriteToFile verschiebt den Dateizeiger hinter den geschriebenen Bereich.
- WriteToFile schreibt keine Ende-Null in die Datei. Verwenden Sie die Routine **FileWrite**, wenn Sie eine Ende-Null schreiben wollen.
- Ist die Datei größer, als der geschriebene Text, so bleiben die nicht überschriebenen Daten erhalten. Verwenden Sie die Routine **FileTruncate**, wenn Sie die Datei nach dem Schreiben an der aktuellen Position abschneiden wollen. Diese Routinen und die Arbeit mit Dateien sind im Handbuch "Spezielle Themen", Kapitel 9, beschrieben.
- WriteToFile setzt die globale Variable fileError. Ist der Parameter start größer als verfügbarer Text wird kein Laufzeitfehler erzeugt, sondern fileError auf ERROR_TEXT_TOO_SHORT (-22) gesetzt.
- Ist der Parameter end kleiner als der Parameter start, so wird ein Laufzeitfehler erzeugt und das Programm beendet.

Beispiele:

```
' Kompletten Text in eine (offene) Datei schreiben.
' Zeichen in den Windows-Zeichensatz konvertieren
' GEOS-Zeilenumbrüche durch DOS-Zeilenumbrüche ersetzen

  MyTextObj.WriteToFile fh, GEOS_TO_WIN + CR_TO_CRLF
```

```
' 200 Zeichen mit HTML Codierung schreiben
' "trw" öffnet evtl. vorhandene Dateien und schneidet sie ab.

DIM fh as FILE

  fh = FileCreate "FILE.TXT", "trw"
  MyText.WriteToFile fh, GEOS_TO_HTML + CR_TO_CRLF, 0, 200
  FileClose fh
```

ReplaceFromFile

ReplaceFromFile ersetzt den aktuellen Text durch den in einer Datei (DOS-Datei oder GEOS-Daten-Datei) enthaltenen Text. Dabei wird der Text bis zum Dateiende oder bis zum Auftreten einer Ende-Null eingelesen. Eine Prüfung auf ungültige Zeichen erfolgt nicht.
Sie können die zu Textmenge begrenzen und festlegen ob der Text in einen anderen Zeichensatz (z.B. von HTML nach GEOS) konvertiert werden soll.
ReplaceFromFile liest immer ab der aktuellen Dateiposition und verschiebt den Dateizeiger hinter den gelesenen Text.

Syntax: **<obj>.ReplaceFromFile** fh [, convertMode [, maxLen]]

fh: Variable oder Ausdruck vom Typ FILE. Die Datei muss offen sein.
convertMode: Bestimmt, zwischen welchen Zeichensätzen konvertiert werden soll. Siehe unten.
Defaultwert: Null (keine Konvertierung)
maxLen: Maximale Anzahl zu lesender Zeichen. MaxLen bezieht sich auf die Datei, die Anzahl der der erzeugten (d.h. an das Textobjekt übergebenen) Zeichen kann je nach convertMode abweichen.
Defaultwert: 4 294 967 294 (alles)

Hinweise:

- Für den Parameter convertMode sind alle Werte zugelassen, die auch für die Funktion Convert$ zugelassen sind. Als Ersatzzeichen für nicht konvertierbare Zeichen wird immer der Unterstrich '_' verwendet.
  Eine Beschreibung der Convert$-Funktion finden Sie im Kapitel 2.4.3 (Konvertierungsfunktionen) des Programmierhandbuchs.

Warnung

Die Werte GEOS_TO_HTML, GEOS_TO_HTML_BR, GEOS_TO_UTF8 sowie das Flag CR_TO_CRLF für convertMode können den Text verlängern, so dass ein Textobjekt, z.B. ein Memo, ihn nicht mehr aufnehmen kann. Beim Lesen von Text ist es allerdings selten, dass man diese Werte verwendet.
Für LargeText-Objekte existiert dieses Problem nicht.

Beispiele:

```
' Den kompletten Text aus einer (offenen) DOS-Datei ersetzen

  MyTextObj.ReplaceFromFile fh, DOS_TO_GEOS + CRLF_TO_CR
```

```
' Den Text durch maximal 200 Zeichen aus einer (offenen) HTML-
' Datei ersetzen. Die 200 Zeichen beziehen sich auf die Datei.

  MyTextObj.ReplaceFromFile fh, HTML_TO_GEOS + CRLF_TO_CR, 200
```

InsertFromFile

InsertFromFile fügt Text aus einer Datei an der aktuellen Cursorposition ein. Wenn etwas selektiert ist, wird der neue Text hinter dem selektierten Bereich eingefügt. Ansonsten gelten die bei ReplaceFromFile angegebenen Hinweise.

Syntax: **<obj>.InsertFromFile** fh [, convertMode [, maxLen]]

FileTextSize

FileTextSize ermittelt die Anzahl der Zeichen, die aus einer offenen DOS- oder GEOS-Daten-Datei mit ReplaceFromFile oder InsertFromFile maximal gelesen werden können. Dabei wird ab der aktuellen Dateiposition begonnen und am Dateiende bzw. der nächsten Ende-Null abgebrochen. Die Ende-Null wird (wenn vorhanden) nicht mitgezählt.
Wird ein Parameter für convertMode angegeben, so wird die Anzahl der Zeichen nach der Konvertierung zurückgeliefert.

Syntax: **<numVar>** = **<obj>.FileTextSize** (fh [, convertMode])

fh: Variable oder Ausdruck vom Typ FILE. Die Datei muss offen sein.
convertMode: Bestimmt, zwischen welchen Zeichensätzen konvertiert werden soll. Siehe ReplaceFromFile.
Defaultwert: Null (keine Konvertierung)

Anmerkung: FileTextSize ist zwar eine Textobjekt-Methode, verwendet aber die Eigenschaften des Textobjekts selbst nicht.

Beispiel

```
DIM size

  size = DemoMemo.FileTextSize(fh, GEOS_TO_HTML)
  IF size <= 4000 THEN
    DemoMemo.ReplaceFromFile fh , GEOS_TO_HTML
  ELSE
   MsgBox "Der Text wäre" + Str$(size-4000) + " Zeichen zu lang."
  End IF
```

WriteToVMFile

WriteToVMFile schreibt den Text als VMArray in eine offene VM-Datei und liefert das VMBlock-Handle des VMArrays zurück.
Um WriteToVMFile verwenden zu können, müssen Sie die Library "VMFiles" includen. Diese kann separat von der R-BASIC Webseite heruntergeladen werden.

Syntax: **<hanVar>** = **<obj>**.**WriteToVMFile** fh [, start [, end]]

hanVar:	Variable vom Typ Handle
fh:	Variable oder Ausdruck vom Typ FILE. Die Datei muss offen und eine VM-Datei sein.
start:	Cursorposition, ab der geschrieben werden soll. Die Zählung beginnt bei Null. Defaultwert: Null (von Anfang an)
end:	Cursorposition, bis zu der geschrieben werden soll. Die Zählung beginnt bei Null. End darf größer als die Textlänge sein. Defaultwert: 4 294 967 294 (alles)

Hinweise:

- WriteToVMFile speichert den Text immer so, wie er im Text-Objekt angezeigt wird. Eine Konvertierung in andere Zeichensätze ist nicht möglich.
- WriteToVMFile schreibt ein "Standard" VMArray mit einer Elementgröße von 1 Byte. Es enthält mindestens eine Ende-Null. Es ist erlaubt, das VMArray mit den VMArray-Routinen der VMFiles-Library zu manipulieren. Die Ende-Null darf aber nicht entfernt werden.
- WriteToVMFile legt immer ein neues VMArray an. Arrays, die Sie nicht mehr brauchen, müssen Sie mit **VMArrayDestroy** vernichten.
- WriteToVMFile setzt die globale Variable fileError. Ist der Parameter start größer als verfügbarer Text wird kein Laufzeitfehler erzeugt, sondern fileError auf ERROR_TEXT_TOO_SHORT (-22) gesetzt.
- Ist der Parameter end kleiner als der Parameter start, so wird ein Laufzeitfehler erzeugt und das Programm beendet.

Beispiele: Siehe unten.

ReplaceFromVMFile

ReplaceFromVMFile ersetzt den aktuellen Text durch den in einer VM-Datei enthaltenen Text. Um ReplaceFromVMFile verwenden zu können, müssen Sie die Library "VMFiles" includen.

Syntax: **<obj>.ReplaceFromVMFile** fh, block

fh: Variable oder Ausdruck vom Typ FILE. Die Datei muss offen und eine VM-Datei sein.
block: Handle auf ein von WriteToVMFile geschriebenes VMArray.

Hinweise:
- ReplaceFromVMFile beeinflusst die globale Variable fileError nicht.

InsertFromVMFile

InsertFromVMFile fügt den in einer VM-Datei enthaltenen Text an der aktuellen Cursorposition ein. Ist etwas selektiert, so wird der neue Text hinter dem selektierten Bereich eingefügt.
Ansonsten gelten die bei ReplaceFromVMFile angegebenen Hinweise.

Syntax: **<obj>.ReplaceFromVMFile** fh, block

fh: Variable oder Ausdruck vom Typ FILE. Die Datei muss offen und eine VM-Datei sein.
block: Handle auf ein von WriteToVMFile geschriebenes VMArray.

Beispiele

Die Routine SaveToVMFile speichert den Text eines Textobjekts in einer Datei. Der Parameter "t" bei VMOpen sorgt dafür, dass die Datei nach dem Öffnen abgeschnitten wird, also leer ist. Das neue VMArray wird als "Mapblock" gesetzt, damit man später einfach darauf zugreifen kann.

```
SUB SaveToVMFile()
DIM fh AS FILE
DIM blk AS HANDLE

  fh = VMOpen("VMTextFile", "trw")
  blk = DemoLargeText.WriteToVMFile(fh)
  VMSetMapBlock(fh, blk)
  VMClose(fh)

End SUB
```

Die Routine LoadFormVMFile liest den Text aus der von SaveToVMFile angelegten Datei. Bei VMOpen darf der Parameter "t" nicht angegeben werden, damit die Daten beim Öffnen der Datei erhalten bleiben.
Das VMArray mit dem Text wurde als "Mapblock" gesetzt, so kann man wieder darauf zugreifen.

```
SUB LoadFromVMFile()
DIM fh AS FILE
DIM blk AS HANDLE

  fh = VMOpen("VMTextFile", "rw")
  blk = VMGetMapBlock(fh)
  DemoLargeText.InsertFromVMFile(fh, blk)
  VMClose(fh)

End SUB
```

Da WriteToVMFile jeweils ein neues VMArray anlegt, ist das "Ersetzen" eines Texts in einem VMArray nicht möglich. Also muss man das "alte" VMArray manuell vernichten, und stattdessen das neue VMArray verwenden.

```
FUNCTION ReplaceVMArray(fh AS FILE) AS HANDLE
DIM oldArray, newArray AS HANDLE

  ' Neues VMArray anlegen und als MapBlock setzen
  ' danach altes Array vernichten (nur wenn existent!)

  oldArray = VMGetMapBlock(fh)
  newArray = DemoMemo.WriteToVMFile(fh)
  VMSetMapBlock(fh, newArray)
  IF oldArray <> NullHandle() THEN VMArrayDestroy (fh, oldArray)

  RETURN newArray

End FUNCTION
```

Ein Beispiel, wie man mehrere unabhängige Texte in einer einzigen VM-Datei speichert, finden Sie in der Beispieldatei "Text Speichern, komplex, VM-Datei" im Ordner "Beispiel\Objekte\Text". Dabei wird der Text nicht mehr direkt als Mapblock gesetzt.

VMFileTextSize

VMFileTextSize ermittelt die Anzahl der Zeichen eines in einer VM-Datei gespeicherten Textes. Die Ende-Null wird nicht mitgezählt.

Syntax: **<numVar>** = **<obj>**.**VMFileTextSize** (fh, block)

fh: Variable oder Ausdruck vom Typ FILE. Die Datei muss offen und eine VM-Datei sein.
block: Handle auf ein von WriteToVMFile geschriebenes VMArray.

Anmerkungen:

- VMFileTextSize ist zwar eine Textobjekt-Methode, verwendet aber die Eigenschaften des Textobjekts selbst nicht.
- VMFileTextSize macht im Kern nichts anderes, als die Routine VMArrayGetCount() aus der "VMFiles" Library. Sie könnten also auch "VMArrayGetCount(fh, block) - 1" verwenden.

Beispiel

```
DIM size
DIM fh as FILE
DIM block as HANDLE

  < fh und block belegen >

  size = DemoMemo.VMFileTextSize(fh, block)
  MsgBox "Die Datei enthält " + Str$(size) + " Zeichen."
```

Tipps und Tricks: Wie kann man ...

... Text anhängen?
Man muss den Cursor ans Ende setzen (obj.cursorPos = obj.textLen) und dann InsertFromFile bzw. InsertFromVMFile rufen.

... den selektierten Text ersetzen?
Man muss den selektierten Text löschen (obj.DeleteSelection) und dann InsertFromFile bzw. InsertFromVMFile rufen.

... Teile eines Textes aus einer DOS-Datei lesen?
Man positioniert den Dateizeiger am Anfang des zu lesenden Bereichs und übergibt InsertFromFile bzw. ReplaceFromFile als dritten Parameter die Anzahl der zu lesenden Zeichen. Geben Sie für convertMode Null an, wenn der Zeichensatz nicht konvertiert werden soll.

... Teile eines Textes aus einer VM-Datei lesen?
Man muss den Text aus der VM-Datei vollständig lesen. Mit VMFileTextSize kann man die Anzahl der gelesenen Zeichen ermitteln. Die überflüssigen Teile löscht man anschließend manuell mit obj.DeleteRange. Dabei sollte man hinten anfangen, um sich Berechnungen zu ersparen.

... den selektierten Text in eine Datei schreiben?
Man übergibt WriteToFile bzw. WriteToVMFile den selektierten Bereich als start- bzw. end-Parameter, z.B.

```
obj.WriteToVMFile fh , obj.cursorPos , obj.selectionEnd
```

... Text beim Lesen / Schreiben in eine VM-Datei in einen anderen Zeichensatz konvertieren?
Warum sollte man das tun? Am besten, Sie suchen eine andere Lösung.
Ansonsten: Schreiben Sie den Text konvertiert in eine temporäre DOS-Datei und lesen ihn von dort ohne erneute Konvertierung wieder ein. Dann können Sie ihn in einer VM-Datei speichern. Beim Lesen gehen Sie umgekehrt vor.
Vorsicht! bei der Konvertierung ins UTF8-Format könnten Codes unter 32(dez) entstehen, die beim Einlesen in ein GEOS-Text-Objekt nicht angezeigt werden können und sogar das System crashen könnten.

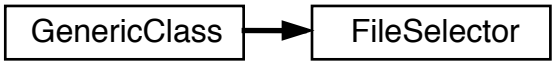
4.11 FileSelector

4.11.1 Überblick

Ein FileSelector stellt die UI bereit, die notwendig ist, um eine Datei oder ein Verzeichnis auszuwählen. Klickt der Nutzer auf einen Eintrag sendet der FileSelector eine Notification-Message aus und der Programmierer kann entscheiden was passiert. Häufig wird die entsprechende Datei dann geöffnet. Die Navigation durch die Verzeichnisse handelt der FileSelector dabei selbständig.

Welche Dateien in der Liste angezeigt werden kann über diverse Kriterien, z.B. das Token der Datei oder eine Dateimaske eingestellt werden.
Häufig ist der FileSelector Teil eines Dialogs, in dem sich noch andere Objekte, z.B. ein "Öffnen" und ein "Abbrechen"-Button befinden.

Abstammung:



Der FileSelector erbt alle Eigenschaften und Fähigkeiten der GenericClass.

Spezielle Instance-Variablen:

Instancevariable	Syntax im UI-Code	Im BASIC-Code
NotificationHandler	NotificationHandler = **<Handler>**	nur schreiben
initialPath	initialPath = **StdPath**, **"<subDir>"**	—
showFilesDisabled	showFilesDisabled = **TRUE | FALSE**	lesen, schreiben
numFilesToShow	numFilesToShow = **anzahl**	lesen, schreiben
fileListWidth	fileListWidth = **anzahl**	lesen, schreiben
selection$	—	lesen, schreiben
path$	—	lesen, schreiben
fullPath$	—	lesen, schreibe
entryFlags	—	nur lesen
matchToken	matchToken = "**TCHR**", **manufID**	lesen, schreiben
hasMatchToken	—	nur lesen
matchCreator	matchCreator = "**TCHR**", **manufID**	lesen, schreiben
hasMatchCreator	—	nur lesen
matchMask$	matchMask$ = "**fileMask**"	lesen, schreiben
matchCriteria	matchCriteria = **numVal**	lesen, schreiben

Action-Handler-Typen:

Handler-Typ	Parameter
FileAction	(sender as object, entryFlags as WORD, selection$ as String)

Methoden:

Methode	Aufgabe
ClearMatchToken	Entfernt das matchToken aus den Instancevariablen
ClearMatchCreator	Entfernt das matchCreator-Token aus den Instancevariablen
Suspend	Verhindert ein Neueinlesen des angezeigten Ordners
EndSuspend	Erlaubt das Neueinlesen des angezeigten Ordners wieder
Rescan	Liest den angezeigten Ordner neu ein
UpDirectory	Wechselt in das Parent-Verzeichnis
OpenEntry	Öffnet das selektierte Verzeichnis

4.11.2 Konfigurieren des FileSelectors

Meistens wird der FileSelector im UI-Code konfiguriert. Dazu stehen die folgenden Instancevariablen zur Verfügung:

Instancevariable	Syntax im UI-Code	Im BASIC-Code
initialPath	initialPath = **StdPath**, **"<subDir>"**	—
showFilesDisabled	showFilesDisabled = **TRUE I FALSE**	lesen, schreiben
numFilesToShow	numFilesToShow = **anzahl**	lesen, schreiben
fileListWidth	fileListWidth = **anzahl**	lesen, schreiben
matchToken	matchToken = "**TCHR**", **manufID**	lesen, schreiben
matchCreator	matchCreator = "**TCHR**", **manufID**	lesen, schreiben
matchMask$	matchMask$ = "**fileMask**"	lesen, schreiben
matchCriteria	matchCriteria = **numVal**	lesen, schreiben

Ein typisches FileSelector-Objekt sieht wie folgt aus. Im Beispiel werden die R-BASIC Font-Dateien zur Anzeige gebracht.

```
FileSelector TestFileselector
  Caption$ = "Bitte Datei auswählen"
  justifyCaption = J_TOP
  initialPath = SP_USER_DATA, "R-BASIC\\FONT"
  NotificationHandler = FileSelected
  numFilesToShow = 12
  matchMask$ = "*.RBF"
  matchCriteria = FMC_DIRS + FMC_DOS_FILES
End OBJECT
```

Zur Auswahl, welche Dateien in der Liste angezeigt werden sollen, stehen die Instancevariablen **matchToken** (engl. to match = übereinstimmen), **match-Creator**, **matchMask$** und **matchCriteria** zur Verfügung. Die Defaultwerte sind so gewählt, dass alle Dateien und alle Verzeichnisse angezeigt werden. Geben Sie mehrere Match-Kriterien an so müssen diese gleichzeitig erfüllt sein. Widersprüchliche Kriterien z.B. das Token von GeoWrite-Dateien und das Creator-Token von GeoDraw führen dazu, dass keine Dateien angezeigt werden.

initialPath

InitialPath legt den anfangs angezeigten Pfad fest. Das kann nur im UI-Code geschehen. Wird kein initialPath gesetzt so wird das GEOS Hauptverzeichnis angezeigt.

Syntax UI-Code:	**initialPath** = **stdPath**, **"Subdir"**
stdPath:	Eine StandardPath Konstante
"Subdir":	Unterverzeichnis. Das kann auch ein Leerstring sein.

Beispiele

```
initialPath = SP_DOCUMENT, "R-BASIC\\Beispiele"
initialPath = SP_USER_DATA, ""
```

showFilesDisabled

ShowFilesDisabled legt fest, ob Dateien in der Liste als "disabled" angezeigt werden. Der Nutzer sieht die Dateinamen hellgrau, kann sie aber nicht anwählen. Das wird z.B. im Dialog "Speichern unter ..." verwendet.

Syntax	UI- Code:	**showFilesDisabled** = **TRUE** (Defaultwert: FALSE)
	Lesen:	**<numVar>** = **<Obj>**.**showFilesDisabled**
	Schreiben:	**<Obj>**.showFilesDisabled = TRUE I FALSE

numFilesToShow

NumFilesToShow legt die Anzahl der gleichzeitig angezeigten Listeneinträge fest.

Syntax	UI- Code:	**numFilesToShow** = **numWert**
	Lesen:	**<numVar>** = **<Obj>**.**numFilesToShow**
	Schreiben:	**<Obj>**.**numFilesToShow** = **numWert**
		numWert: numerischer Ausdruck, bestimmt die Anzahl

fileListWidth

FileListWidth legt die Breite der Liste fest. Maßeinheit ist die "mittlere Zeichenbreite". Der Defaultwert ist Null, d.h. die Liste bestimmt die ihre Breite selbst. FileListWidth wird selten verwendet, weil der Defaultwert im Allgemeinen passt.

Syntax	UI- Code:	**filesListWidth** = **numWert**
	Lesen:	**<numVar>** = **<Obj>**.**filesListWidth**
	Schreiben:	**<Obj>**.**filesListWidth** = **numWert**
		numWert: Breite der Liste, Maßeinheit: "mittlere Zeichenbreite". Null: Defaultwert verwenden.

matchToken

Nur GEOS-Dateien mit dem passenden Token werden angezeigt. DOS-Dateien werden nicht mehr angezeigt. Das Token bestimmt das Icon, mit dem die Datei im GeoManager angezeigt wird.

Syntax	UI- Code:	**matchToken** = **"tchr"**, **manufID**
	Lesen:	**<tok>** = **<obj>**.**matchToken**
	Schreiben:	**<obj>**.**matchToken** = **"tchr"**, **manufID**
		"tchr" : 4 Buchstaben, "TokenChars"
		manufID: Manufacturer ID (numerischer Wert)
		<tok>: Variable vom Typ GeodeToken

Per Default ist kein Wert für matchToken gesetzt und alle Dateien werden angezeigt.

Beispiel: GeoDraw-Dateien anzeigen

```
matchToken = "DDAT", 0
```

Will man nur Dateien anzeigen, deren Token aus lauter Nullen besteht (das ist z.B. bei den VM-Dateien der Fall, die vom Icon-Editor geschrieben werden), muss man als tokenChars einen Leerstring zuweisen:

```
matchToken = "", 0
```

Das Token "", 0 ist nicht identisch mit dem Zustand "kein Token gesetzt". Im ersten Fall werden nur Dateien mit einem Token, das aus lauter Nullen besteht angezeigt, im zweiten Fall wird das Token ignoriert (die Dateien werden nicht nach ihrem Token gefiltert). Um ein gesetztes Token wieder zu entfernen verwenden Sie die Methode **ClearMatchToken**, die weiter unten beschrieben ist.

matchCreator

Nur GEOS-Dateien mit dem passenden Creator-Token werden angezeigt. DOS-Dateien werden nicht mehr angezeigt. Das Creator-Token bestimmt die Applikation, mit der die Datei erstellt wurde. Bei Programmen ist das "GEOS",0.

Syntax	UI- Code:	**matchCreator** = **"tchr"**, **manufID**
	Lesen:	**<tok>** = **<obj>.matchCreator**
	Schreiben:	**<obj>.matchCreator** = **"tchr"**, **manufID**
		"tchr" 4 Buchstaben, "TokenChars"
		manufID Manufacturer ID (numerischer Wert)
		<tok> Variable vom Typ GeodeToken

Per Default ist kein Wert für matchCreator gesetzt und alle Dateien werden angezeigt.

Beispiel: GeoWrite-Dateien anzeigen

```
matchCreator = "WP00", 0
```

Will man nur Dateien anzeigen, deren Creator-Token aus lauter Nullen besteht (das ist z.B. bei den VM-Dateien der Fall, die vom Icon-Editor geschrieben werden), muss man als tokenChars einen Leerstring zuweisen:

```
matchCreator = "", 0
```

Das Creator-Token "", 0 ist nicht identisch mit dem Zustand "kein Token gesetzt". Im ersten Fall werden nur Dateien mit einem Token, dass aus lauter Nullen besteht angezeigt, im zweiten Fall wird das Creator-Token ignoriert (die Dateien werden nicht nach ihrem Creator-Token gefiltert). Um ein gesetztes Creator-Token wieder zu entfernen verwenden Sie die Methode **ClearMatchCreator**, die weiter unten beschrieben ist.

matchMask$

Nur Dateien, die der übergebenen Namensmaske entsprechen, werden angezeigt. Dabei gelten die GEOS-Namenskonventionen.

- Wildcards "*" und "?" sind zulässig
  * (Sternchen): beliebige Anzahl (oder Null) Zeichen oder Ziffern
  ? Genau ein Zeichen oder eine Ziffer
  : und \ sind nicht zulässig
- Die Groß- und Kleinschreibung spielt eine Rolle
- Für DOS-Dateien: Großbuchstaben verwenden

Per Default ist kein Wert für matchMask$ gesetzt und alle Dateien werden angezeigt.

Syntax	UI- Code:	**matchMask$** = **"maskString"**
	Lesen:	**<stringVar>** = **<obj>**.**matchMask$**
	Schreiben:	**<obj>.matchMask$** = **"maskString"**
		"maskString" Eine Filtermaske für die Dateien, z.B. "*.PCX"

Beispiele:

"D*"	Alle Dateien, die mit einem großen D anfangen.
"?a*"	Alle Dateien, deren zweiter Buchstabe ein a ist.
"*a*"	Alle Dateien, die ein a im Namen enthalten. Das a darf auch am Anfang oder am Ende stehen.
"*.*"	Alle Dateien, die einen Punkt im Namen enthalten.
"*.PCX"	Alle Dateien, die auf Punkt-PCX enden.
"*"	Alle Dateien

Wenn keine Maske gesetzt ist liefert matchMask$ einen Leerstring. Das Schreiben eines Leerstrings löscht eine gesetzte Maske. Funktionell sind "keine Maske gesetzt" und "*" identisch, "keine Maske gesetzt" ist aber schneller, weil der FileSelector keine Vergleiche ausführt.

matchCriteria

MatchCriteria ist ein numerischer Wert, der bestimmt, welche Art von Dateien angezeigt werden sollen.

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

Der Defaultwert ist FMC_ALL_FILES (= FMC_DIRS + FMC_DOS_FILES + FMC_GEOS_EXEC + FMC_GEOS_DATA). Per Default werden alle Dateien und Ordner angezeigt.

Erlaubte Werte:

Konstante	Wert	(hex)	Bedeutung
FMC_DIRS	32768	&h8000	Ordner anzeigen
FMC_DOS_FILES	16384	&h4000	Nicht-GEOS-Dateien anzeigen
FMC_GEOS_EXEC	8192	&h2000	GEOS Programme und Libraries anzeigen
FMC_GEOS_DATA	4096	&h1000	GEOS VM- und Daten-Dateien anzeigen
FMC_ALL_FILES	61440	&hF000	Alle Dateien und Ordner anzeigen. Das entspricht der Summe der vier Werte von oben.
FMC_MASK_CASE_INSENSITIVE	2048	&h0800	Masken unterscheiden nicht zwischen Groß- und Kleinbuchstaben
FMC_USE_MASK_FOR_DIRS	128	&h0080	Masken auch auf Ordner anwenden

Die hier nicht angegebenen Bits sind intern verwendet und werden von R-BASIC nicht unterstützt. Sie müssen Null sein.

Beispiele

```
DIM criteria as word ' oder as real
  matchCriteria = FMC_DIRS + FMC_GEOS_DATA
  MyObj.matchCriteria = FMC_DIRS + FMC_GEOS_EXEC
  criteria = MyObj.matchCriteria
```

Der Wert von matchCriteria besteht aus einzelnen Bits, die jedes eine bestimmte Bedeutung haben (sog. BitFlags). Die Abfrage erfolgt mit der logischen Operation **AND**, das setzen mit der logischen Operation **OR**. Das Löschen eines Bits erfordert die Operation "**AND** (**NOT** bit_zu_löschen)".

Beispiel:
FMC_DIRS abfragen

```
  IF FSel.matchCriteria AND FMC_DIRS THEN ...
```

Beispiel:
FMC_MASK_CASE_INSENSITIVE setzen ohne die anderen Flags zu ändern

```
  FSel.matchCriteria =
    FSel.matchCriteria OR FMC_MASK_CASE_INSENSITIVE
```

Beispiel:
FMC_DOS_FILES löschen ohne die anderen Flags zu ändern

```
  FSel.matchCriteria =
    FSel.matchCriteria AND (NOT FMC_DOS_FILES)
```

4.11.3 Arbeit mit Token und Creator

Instancevariable	Syntax im UI-Code	Im BASIC-Code
hasMatchToken	—	nur lesen
hasMatchCreator	—	nur lesen

Methode	Aufgabe
ClearMatchToken	Entfernt das matchToken aus den Instancevariablen
ClearMatchCreator	Entfernt das matchCreator-Token aus den Instancevariablen

Die Instancevariablen **matchToken** und **matchCreator** haben die Besonderheit, dass die dazugehörigen Daten (das jeweilige GeodeToken) vorhanden sein kann oder auch nicht. Ist das Token vorhanden werden alle Dateien entsprechend dem Token (bzw. Creator-Token) gefiltert und nur die Dateien angezeigt, die eine Übereinstimmung aufweisen. Ist das Token nicht vorhanden werden die Dateien nicht gefiltert, ihr Token (bzw. Creator-Token) ist egal.

Per Default sind keine Werte für matchCreator und matchToken gesetzt, die Dateien werden also nicht gefiltert. Die Filterung wird durch Setzen eines Wertes für dies Instancevariablen aktiviert. Es ist nun nicht möglich durch Zuweisen eines "Leer"-Token zu den Instancevariablen matchToken bzw. matchCreator das jeweilige Token zu löschen. Die Filterung nach Tokens kann auf diese Weise nicht aufgehoben werden. Diesem Zweck dienen die in diesem Kapitel beschriebenen Methoden ClearMatchToken und ClearMatchCreator.

hasMatchToken

HasMatchToken enthält die Information, ob mit "matchToken" ein Wert gesetzt wurde oder nicht. Der Wert kann nur gelesen werden und liefert TRUE oder FALSE.

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

hasMatchCreator

HasMatchCreator enthält die Information, ob mit "matchCreator" ein Wert gesetzt wurde oder nicht. Der Wert kann nur gelesen werden und liefert TRUE oder FALSE.

Syntax Lesen: **<numVar>** = **<Obj>.hasMatchCreator**

ClearMatchToken

Die Methode ClearMatchToken löscht den mit "matchToken" gesetzten Wert.

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

ClearMatchCreator

Die Methode ClearMatchCreator löscht den mit "matchCreator" gesetzten Wert.

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

4.11.4 Behandlung der Notification-Message

Klickt der Nutzer auf einen Eintrag in der Liste des FileSelectors wird der Notification-Handler des FileSelectors aufgerufen. Das kann ein Einfachklick oder ein Doppelklick sein. Außerdem ist es möglich weitere Informationen über den aktuell selektierten Eintrag, z.B. den Pfad zum dargestellten Verzeichnis oder ob es sich um eine Ordner oder eine Datei handelt, zu erhalten. Dazu stehen die folgenden Instancevariablen zur Verfügung:

Instancevariable	Syntax im UI-Code Im BASIC-Code	
NotificationHandler	NotificationHandler = **<Handler>**	nur schreiben
path$	—	lesen, schreiben
fullPath$	—	lesen, schreibe
selection$	—	lesen, schreiben
entryFlags	—	nur lesen

Handler-Typ	Parameter
FileAction	(sender as object, entryFlags as WORD, selection$ as String)

NotificationHandler

Die Instance-Variable **NotificationHandler** enthält den Namen des Handlers, der gerufen wird, wenn der Nutzer auf einen Eintrag in der Liste klickt. Das kann ein Einfachklick oder ein Doppelklick sein. Der Wert wird üblicherweise im UI-Code gesetzt, bei Bedarf kann er auch zur Laufzeit (im BASIC-Code) gesetzt werden.

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

Wichtig! FileSelectoren können nicht in Blocking-Dialogen (vgl. Kapitel 4.6.7, DialogObj.attrs = DA_BLOCKING) verwendet werden. Blocking-Dialoge blockieren den BASIC-Thread so lange, bis der Dialog geschlossen wird. In dieser Zeit kann der Notification-Handler nicht ausgeführt werden.

NotificationHandler müssen als FileAction deklariert sein.

Der Parameter **selection$** enthält den aktuell ausgewählten Eintrag (Dateiname oder Ordnername). Wenn der erste Eintrag selektiert ist (aktuelles Verzeichnis oder Wurzelverzeichnis) enthält selection$ den Text "." (Der Text besteht nur aus nur einem Punkt.)

Der Parameter **entryFlags e**nthält Informationen über den aktuell selektierten Eintrag. Jedes Bit hat eine eigene Bedeutung. Die Abfrage der Bits erfolgt mit der logischen Operation AND.

Folgende Werte und Konstanten sind definiert.

Konstante	Wert	hex	Bedeutung
—	32768	&h8000	intern verwendet
FEF_SUBDIR	16348	&h4000	Ein Verzeichnis ist selektiert
FEF_OPEN	8192	&h2000	Ein Doppelklick wurde ausgeführt
FEF_NO_ENTRY	4096	&h1000	Die Dateiliste ist leer
FEF_ERROR	2048	&h800	Es gab einen Fehler
FEF_TEMPLATE	1024	&h400	Die Datei ist ein "Muster"
—	512	&h200	Die Datei ist "shared-multiple"
—	256	&h100	Die Datei ist "shared-single"
FEF_READ_ONLY	128	&h80	Es ist eine "Nur Lesen" Datei
FEF_PARENT_DIR	64	&h40	Der erste Eintrag in der Liste ist selektiert.
—	32	&h20	Der Eintrag ist disabled

Die in der Tabelle nicht aufgeführten Bits sind nicht definiert und sollten nicht verwendet werden. "Shared-multiple" bzw. "shared-single" bedeutet, dass die Datei im Netzwerk von mehreren Nutzern gleichzeitig geöffnet werden kann. Der Nutzer kann das z.B. in GeoWrite über den "Dokument-Typ" im Menü "Datei"->"Sonstiges" festlegen.

Die typische Reaktion auf einen Doppelklick auf eine Datei besteht darin, in den vom FileSelector angezeigten Pfad zu wechseln (Mithilfe der Instancevariable path$) und dann die selektierte Datei zu öffnen.

Beispiel: Ein typischer Notification-Handler. Wir setzen voraus, dass es einen Öffnen-Button gibt, der nur Enabled werden soll, wenn eine Datei angewählt ist. Ein Doppelklick auf eine Datei soll diese öffnen. Der Handler implementiert das typische Vorgehen dazu. Ein Doppelklick auf ein Verzeichnis wird vom Handler ignoriert. Der FileSelector kümmert sich selbst darum, das entsprechende Verzeichnis zu öffnen und anzuzeigen.
Die Routine *DoSomeThingsWithFile* muss natürlich auch irgendwo definiert sein.

```
FileAction FileSelected
DIM fh as FILE

IF entryFlags AND FEF_SUBDIR THEN
  OpenButton.enabled = FALSE
  RETURN ' Verzeichnis selektiert
   ELSE
  OpenButton.enabled = TRUE   ' Datei ist ausgewählt
  End IF
```

```
IF entryFlags AND FEF_OPEN THEN    ' Doppelklick
  SetCurrentPath sender.path$
  fh = FileOpen selection$
  IF  fh <> NullFile() THEN
    DoSomeThingsWithFile(fh)
    FileClose fh
    End IF
  End IF

End ACTION
```

path$

Die Instancevariable path$ enthält den angezeigten Pfad, ohne Selektion. Sie kann gelesen und geschrieben, aber nicht im UI-Code verwendet werden. Um den anzuzeigenden Pfad im UI-Code zu setzen verwenden Sie die Instancevariable initialPath.

Syntax Lesen: **<stringVar>** = **<obj>.path$**
Schreiben: **<obj>.path$** = **<pathExpression>**
<pathExpression>: Stringausdruck, zu setzender Pfad.
Es kann ein relativer Pfad oder ein absoluter Pfad (mit Laufwerksbuchstabe, z.B. "C:\\DOS") sein.

Beispiel: Anzeige eines absolut angegebenen Pfades.

```
MyObj.path$ = "C:\\Bilder\\Kinder"
```

Beispiel: Anzeige des Unterverzeichnisses "Arbeit" des aktuell angezeigten Verzeichnisses.

```
MyObj.path$ = "Arbeit"
```

Beispiel: Wechseln in das vom FileSelector angezeigte Verzeichnis

```
SetCurrentPath MyObj.path$
```

Hinweise
Schreiben in die Instancevariable path$:

- Schreiben von path$ setzt die globale Variable **fileError** (Null wenn OK, ERROR_PATH_NOT_FOUND wenn der Pfad invalid ist.
- Um in das Root eines Laufwerks zu wechseln verwendet man z.B. "C:\\"
- Bei Pfadangaben spielt die Groß-/Kleinschreibung eine Rolle. Für reine DOS-Verzeichnisse sollten Sie Großbuchstaben verwenden.

Lesen von path$

- Standard R-BASIC Strings können bis zu 128 Zeichen aufnehmen. Pfade können bis 198 Zeichen lang sein. Variablen, die einen Pfad aufnehmen sollen, sollten als String(198) oder länger definiert sein. Beispiel:

```
DIM pathVar$ as STRING(200)
```

fullPath$

Die Instancevariable fullPath$ enthält den angezeigten Pfad einschließlich des selektierten Eintrags. Sie kann gelesen und geschrieben, aber nicht im UI-Code verwendet werden.

Syntax Lesen: **<stringVar>** = **<obj>.fullPath$**
Schreiben: **<obj>.fullPath$** = **<pathExpression>**
<pathExpression>: Stringausdruck, zu setzender Pfad.
Es kann ein relativer Pfad oder ein absoluter Pfad sein.

Schreiben in die Instancevariable fullPath$:

- Das letzte Pfadelement wird als zu selektierende Eintrag interpretiert, auch wenn es ein Verzeichnis ist. Beispiel:

```
MyObj.fullPath$ = "C:\\GEOS\\DOCUMENT"
```

zeigt das GEOS Verzeichnis an, wobei DOCUMENT selektiert ist.
Um das Root eines Verzeichnisses zu selektieren hängt man einen Backslash an:

```
MyObj.fullPath$ = "C:\\GEOS\\DOCUMENT\\"
```

zeigt das GEOS\DOCUMENT Verzeichnis an. Oder man verwendet **path$**.
- Schreiben von fullPath$ setzt die globale Variable **fileError** (Null wenn OK, ERROR_PATH_NOT_FOUND wenn der Pfad invalid ist.
- Um in das Root eines Laufwerks zu selektieren verwendet man z.B. "C:\\"
- Bei Pfadangaben spielt die Groß-/Kleinschreibung eine Rolle. Für reine DOS-Verzeichnisse sollten Sie Großbuchstaben verwenden.

Lesen von fullPath$

- Standard R-BASIC Strings können bis zu 128 Zeichen aufnehmen. Pfade können bis 198 Zeichen lang sein. Hinzu kommt die Selektion (max. 32 Zeichen) und der Backslash. Variablen, die einen Pfad aufnehmen sollen, sollten als String(231) oder länger definiert sein. Beispiel:

```
DIM pathVar$ as STRING(235)
```

selection$

Die Instancevariable selection$ enthält den aktuell ausgewählten Eintrag (Dateiname oder Ordnername). Sie kann gelesen und geschrieben, aber nicht im UI-Code verwendet werden.

Syntax Lesen: **<stringVar>** = **<obj>.selection$**
Schreiben: **<obj>.selection$** = **<stringExpression>**
<stringExpression>: Stringausdruck, zu selektierender Eintrag

---

Wenn der erste Eintrag selektiert ist (aktuelles Verzeichnis oder Wurzelverzeichnis) enthält selection$ den Text "." (Der Text besteht nur aus nur einem Punkt.)

Hinweise:
- Schreiben von selection$ setzt die globale Variable **fileError** (Null wenn OK, ERROR_PATH_NOT_FOUND wenn der Eintrag nicht existiert.
- Um den ersten Eintrag zu selektieren kann man "." oder einen Leerstring übergeben.

Beispiel: Lesen der Selektion

```
DIM sel$ as String(32)    ' Das reicht
  sel$ = MyObj.selection$
```

Beispiel: Selektieren des Unterverzeichnisses "Arbeit". Das Verzeichnis wird nur selektiert, der FileSelector wechselt nicht in das Verzeichnis

```
  MyObj.selection$ = "Arbeit"
```

entryFlags

EntryFlags enthält die Informationen über den aktuell selektierten Eintrag, die auch an den Parameter "entryFlags" des FileSelector NotificationHandlers übergeben werden. Eine Beschreibung der einzelnen Flagbits finden Sie dort. Der Wert kann nur gelesen werden.

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

Beispiel: Abfrage ob eine Datei selektiert ist

```
DIM flags
  flags = MyObj.entryFlags
  IF  (flags  AND  FEF_SUBDIR) = 0  THEN
    MsgBox "Eine Datei ist selektiert"
    END IF
```

Beispiel: Ein "Open"-Handler
Wir setzen einen FileSelector (DemoFileSelector) voraus, der den "Öffnen"-Button nicht enabled oder disabled. Der Buttonhandler muss daher entscheiden ob eine Datei selektiert ist oder ein Verzeichnis und entsprechend reagieren.

```
ButtonAction OpenFileOrFolder
DIM flags as word

  flags = DemoFileSelector.entryFlags

  IF flags AND FEF_SUBDIR THEN
    DemoFileSelector.OpenEntry    ' Verzeichnis öffnen
  ELSE
    SetCurrentPath DemoFileSelector.path$
    MsgBox "Eine Datei ist selektiert"
    ' Hier die Datei öffnen ...
  End IF

End Action
```

4.11.5 Weitere Fähigkeiten

Dieser Abschnitt beschreibt einige Fähigkeiten des FileSelectors die gelegentlich benötigt werden. Dafür sind die folgenden Methoden implementiert:

Methode	Aufgabe
Suspend	Verhindert ein Neueinlesen des angezeigten Ordners
EndSuspend wieder	Erlaubt das Neueinlesen des angezeigten Ordners
Rescan	Liest den angezeigten Ordner neu ein
UpDirectory	Wechselt in das Parent-Verzeichnis
OpenEntry	Öffnet das selektierte Verzeichnis

Suspend

Die Methode Suspend verhindert ein Rescan (erneutes Einlesen des angezeigten Ordners) solange bis die Methode EndSuspend aufgerufen wurde. Das ist sinnvoll, wenn man mehrere match-Attribute ändern will.

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

EndSuspend

EndSuspend hebt den mit Suspend gesetzten Zustand wieder auf.

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

Rescan

Veranlasst den FileSelector das Verzeichnis neu einzulesen. Der FileSelector führt auch im "Suspend" Zustand einen Rescan aus, hebt den "Suspend" Zustand aber nicht auf.

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

UpDirectory

Die Methode UpDirectory wechselt in das übergeordnete Verzeichnis.

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

OpenEntry

Die Methode OpenEntry wechselt in das selektierte Verzeichnis und zeigt dieses an. Ist er erste Eintrag im FileSelector selektiert (Aktuelles Verzeichnis, kein Unterverzeichnis) wird ins übergeordnete Verzeichnis gesprungen. Ist eine Datei selektiert passiert nichts.

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

(Leerseite)