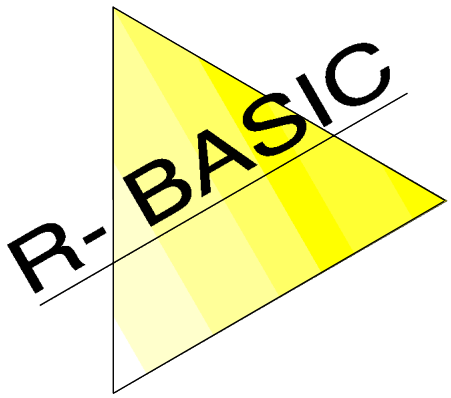
R-BASIC

Einfach unter PC/GEOS programmieren



Objekt-Handbuch

Volume 8
Canvas, Image, Display, DisplayGroup,
DisplayControl

Version 1.0

(Leerseite)

Inhaltsverzeichnis

(Leerseite)

4.16 Canvas

Überblick

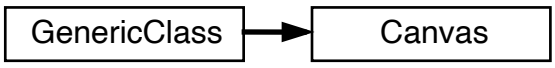
Ein Canvasobjekt (engl: Leinwand) dient dazu, eine Grafik anzuzeigen. Diese Grafik liegt nicht als vorgefertigtes Bild vor, sondern wird zur Laufzeit gezeichnet. Bei Bedarf kann sich die vom Canvasobjekt angezeigte Grafik also ändern. Dazu verfügt das Canvas Objekt über folgende Fähigkeiten:

- OnDraw Handler: Dieser Handler wird gerufen, wenn sich das Objekt auf dem Bildschirm darstellt. Der OnDraw Handler zeichnet die Grafik.
- Gepufferte Darstellung: Wenn gewünscht kann das Objekt die Grafik, die der OnDraw Handler gezeichnet hat, intern abspeichern. Sie kann dann abgerufen werden ohne den OnDraw Handler erneut auszuführen. Das ist wesentlich schneller und geht auch, wenn Ihr Programm gerade mit anderen Operationen beschäftigt ist.

Innerhalb des OnDraw Handlers wird das Canvasobjekt automatisch zu Screen, so dass Grafik und Text einfach ausgegeben werden können. Für viele Zwecke ist das nicht nur ausreichend, sondern auch die bessere Wahl. In Vergleich zu einem BitmapContent-Objekt benötigt das Canvasobjekt viel weniger Speicher, da es keine Bitmap im Hintergrund verwaltet. Und es benötigt kein Viewobjekt als Partner, ist also einfacher zu handhaben.

Verwenden Sie ein Canvasobjekt, wenn Sie eine einfache Grafik, z.B. ein Logo, ein Schema oder einen grafisch gestalteten Text, darstellen wollen, die sich im Programmablauf nicht oder nur selten ändern. Für grafische Ausgaben, die während des Programmablaufs ständig angepasst werden müssen, wie Statusmeldungen (z.B. "Schritt 5 von 10") oder bewegte Grafiken ist das Canvasobjekt nicht optimal. Beispiele zur Verwendung des Canvas-Objekts finden Sie im Ordner "Beispiel\Objekte\Grafik".

Abstammung:



Das Canvasobjekt erbt alle Eigenschaften und Fähigkeiten der GenericClass. Von besonderer Bedeutung sind dabei die Fähigkeiten zum Geometriemanagement (siehe Kapitel 3.3). Insbesondere die Hints **fixedSize**, **initialSize**, **ExpandWidth** und/oder **ExpandHeight** werden häufig genutzt, da das Canvas Objekt "von sich aus" keine vorgegebene Größe hat und sonst möglicherweise unsichtbar klein ist.

Spezielle Instance-Variablen:

Variable	Syntax im UI-Code	Im BASIC-Code
OnDraw	OnDraw = **<Handler>**	nur schreiben
defaultColor	defaultColor = **fg**, **bg**	lesen, schreiben
buffered	buffered = TRUE	lesen, schreiben
bufferedDataSize	bufferedDataSize = **<Wert>**	lesen, schreiben

Methoden:

Methode	Aufgabe
Dirty	Weist das Objekt an, sich neu darzustellen

Action-Handler-Typen:

Handler-Typ	Parameter
DrawAction	(sender as object, width, height as word)

Mausunterstützung

Das Canvasobjekt unterstützt die Behandlung von Mausereignissen. Eine detaillierte Beschreibung der Arbeit mit der Maus finden Sie im Handbuch "Spezielle Themen", Kapitel 17.
Es ist möglich, innerhalb eines Maushandlers Grafiken oder Text auf den Bildschirm auszugeben. Dazu müssen Sie das Objekt explizit zum Screen machen. Häufig werden Sie außerdem die Maus grabben. Ein entsprechendes Beispiel ("Canvas Maus Demo") finden Sie im Ordner "Beispiel\Objekte\Grafik".
Beachten Sie aber, dass das Canvas-Objekt die Grafik- und Textausgaben nicht abspeichert! Sie werden nicht wieder gezeichnet, wenn das Objekt sich selbst neu darstellt.

Der gepufferte Modus

Im gepufferten Modus speichert das Objekt die darzustellende Grafik zwischen. Dadurch muss nicht jedes Mal der OnDraw-Handler gerufen werden, wenn sich das Objekt neu auf dem Bildschirm darstellen muss. Um den gepufferten Modus zu aktivieren verwenden Sie die Instancevariable **buffered**. Sie müssen dem Objekt auch mitteilen, wie groß die zu speichernde Datenmenge ungefähr (!) ist. Dazu verwenden Sie die die Instancevariable **bufferedDataSize**.
Es wird empfohlen, den gepufferten Modus zu verwenden.

Normaler Modus (buffered = FALSE):
Jedes Mal, wenn sich das Objekt auf dem Bildschirm neu darstellen muss, wird der BASIC OnDraw Handler gerufen. Das ist der Standardmodus. Für viele, insbesondere einfache Anwendungen ist er ausreichend.

Gepufferter Modus (buffered = TRUE)
Wenn sich das Objekt auf dem Bildschirm neu darstellen muss, wird die zwischengespeicherte Grafik dargestellt. Der BASIC OnDraw Handler wird nicht gerufen.

Vorteile des gepufferten Modus
- deutlich schnellere Darstellung

- Darstellung auch dann, wenn das BASIC-Programm "beschäftigt" ist. Wenn Ihr Programm eine hohe "Hintergrundaktivität" hat, z.B. bei einem Spiel, sollten Sie unbedingt den gepufferten Modus verwenden.

Nachteile des gepufferten Modus

- Die Grafik kann sich Größenänderungen des Objekts nicht anpassen. Verkleinert sich das Objekt wird möglicherweise "über den Rand" gezeichnet.

Einschalten des gepufferten Modus

Setzen Sie die Instancevariable buffered auf TRUE.
Das Objekt fordert Speicher zum Zwischenspeichern der Grafikbefehle an. Dann wird der OnDraw Handler gerufen. Die Grafikausgaben gehen aber nicht nur auf den Schirm, sondern werden parallel dazu vom Objekt "mitgeschnitten".

Sie können die Variablen **buffered** und **bufferedDataSize** auch schon im UI-Code setzen. Dann wird die gepufferte Grafik gleich beim Programmstart aufgebaut.

Ausschalten des gepufferten Modus

Setzen Sie die Instancevariable buffered auf FALSE.
Das Objekt gibt den angeforderten Speicher frei. Anschließend wird der OnDraw Handler gerufen um das Objekt neu darzustellen.

Verwenden des OnDraw Handlers

Der OnDraw-Handler übernimmt die Darstellung der vom Objekt angezeigten Grafik. Das gilt sowohl für den normalen als auch für den gepufferten Modus, im gepufferten Modus wird der OnDraw-Handler genau einmal gerufen.
Während der **OnDraw** Handler läuft, wird das Canvas Objekt zum **Screen**, das heißt, alle Grafikausgaben gehen über dieses Objekt direkt auf den Bildschirm. Sie können alle Grafikbefehle verwenden, Farben ändern, Texte ausgeben usw., selbst Manipulationen des Koordinatensystems (siehe Kapitel 2.3.3) sind möglich.
Falls ein globaler Screen gesetzt ist (siehe Kapitel 2.3) wird dessen Status vorher gesichert und anschließend wieder hergestellt, so dass es keine gegenseitige Beeinflussung geben kann. **Mit einer Ausnahme**: Alle Einstellungen rund um den Block-Grafik-Modus (siehe Handbuch "Spezielle Themen", Kapitel 2.5) sind immer **global**. Ändern Sie hier innerhalb eines OnDraw Handlers etwas (z.B. durch Laden eines anderen Zeichensatzes) wirkt sich das auf alle anderen Teile des Programms aus.

Beispiel UI Code:

```
Canvas DemoCanvas
  fixedsize = 300, 200
  OnDraw = DemoDraw
END OBJECT
```

Beispiel Handler:

```
DRAWACTION DemoDraw
  Paper 203
  CLS
  graphic.linewidth = 5
  Circle 150, 100,50, LIGHT_BLUE
  Print atxy 110,100;"Hallo"
END ACTION
```

Das Canvasobjekt führt ein automatisches Clipping aus, d.h. die Grafikteile, die über den Rand des Objekts ragen, werden nicht gezeichnet.
Sie können die aktuelle Größe des Zeichenbereichs ermitteln, indem Sie die Systemvariablen **MaxX** und **MaxY** abfragen. Da ist insbesondere dann hilfreich, wenn das Objekt seine Größe verändern kann, z.B. wenn die Hints ExpandWidth und ExpandHeight gesetzt sind.

Beispiel UI:

```
Canvas DemoCanvas
  initialSize = 150, 150
  OnDraw = DrawCircleHandler
  ExpandWidth
  ExpandHeight
END OBJECT
```

Beispiel Code. Es entsteht ein Viertelkreis.

```
DRAWACTION DrawCircleHandler
  FillEllipse 0, 0, 2*MaxX, 2*MaxY, LIGHT_BLUE
END ACTION
```

Beschreibung der Instancevariablen

OnDraw

Die Instance-Variable **OnDraw** enthält den Namen des Handlers, der die Grafik zeichnen soll. Dieser muss als **DrawAction** vereinbart sein. Der Wert wird üblicherweise im UI-Code gesetzt.

```
Syntax  UI- Code:     OnDraw = <Handler>
        Schreiben:    <obj>.OnDraw = <Handler>
```

Bei Bedarf kann der OnDraw-Handler auch zur Laufzeit (im BASIC-Code) gesetzt werden. In diesem Fall stellt sich das Objekt automatisch neu dar.
Hinweis: Der neue OnDraw Handler wird erst ausgeführt nachdem der Handler, der die Zuweisung ausgeführt hat beendet ist!

defaultColor

Die Instance-Variable **defaultColor** enthält die Farben, die beim Aufruf des OnDraw Handlers eingestellt werden. Dabei setzt R-BASIC die Farben folgendermaßen:
Hintergrundfarbe: bg
Text-, Linien- und Flächenfarbe: fg
Das ist prinzipiell so, als würde automatisch die Anweisung "COLOR fg, bg" ausgeführt, kostet aber deutlich weniger Zeit.

```
Syntax  UI-Code: defaultColor = fg, bg
                 fg: Vordergrund (foreground)
                 bg: Hintergrund (background)
                    fg und bg müssen Indexfarben sein. RGB-Farben
                    sind nicht zulässig.
        Lesen:   <numVar> = <obj>.defaultColor (0)      ' fg
                 <numVar> = <obj>.defaultColor (1)      ' bg
        Schreiben: <obj>.defaultColor = fg, bg
```

Canvas Objekte ohne die Anweisung **defaultColor** verwenden die Farben "schwarz auf weiß".

buffered

Die Instancevariable **buffered** legt fest, ob das Canvasobjekt die anzuzeigende Grafik zwischenspeichert (buffered = TRUE, "gepufferter" Modus) oder nicht (buffered = FALSE, normaler Modus). FALSE ist der Defaultwert.
Wenn Sie den gepufferten Modus aktivieren sollten Sie ebenfalls die Instance-variable bufferedDataSize (siehe unten) im Blick haben.

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

Ändern Sie den Wert der Instancevariablen buffered von FALSE auf TRUE oder von TRUE auf FALSE, so ruft das Objekt seinen OnDraw-Handler und stellt sich neu auf dem Bildschirm dar.

bufferedDataSize

Im gepufferten Modus fordert das Objekt Speicher (in einer Datei) an, um die darzustellende Grafik zu speichern. BufferedDataSize enthält die Information, wie groß der benötigte Speicher ungefähr (!) ist. Der Wert ist nicht kritisch. Wenn Sie hier einen falschen Wert angeben, passiert im Allgemeinen nichts.

Syntax	UI- Code:	**bufferedDataSize** = **<Wert>**
	Schreiben:	**<obj>.bufferedDataSize** = **<Wert>**
	Lesen:	**<numVar>** = **<obj>.bufferedDataSize**
	<Wert>:	numerische Konstante, siehe aus der Tabelle unten

Der Defaultwert für bufferedDataSize ist DS_TINY. Das ist ein sinnvoller Wert, wenn Sie nur Grafikbefehle verwenden und keine Bitmapgrafiken ausgeben. Die folgende Tabelle enthält die zulässigen Werte:

Konstante	Wert	Zu erwartende Datenmenge
DS_TINY	0	nicht mehr als 10 .. 20 kByte
DS_SMALL	1	nicht mehr als 50 .. 100 kByte
DS_MEDIUM	2	nicht mehr als 500 kByte ... 1 MB
DS_LARGE	3	nicht mehr als 5 MByte
DS_HUGE	4	möglicherweise mehr als 5 MByte

Beispiele, in welchen Situationen welche Datenmengen zu erwarten sind, finden Sie im Programmierhandbuch, Kapitel 2.8.5, bei der Beschreibung des Befehls StartRecordGS.

Dirty

Die Methode Dirty (engl: schmutzig) bewirkt, dass sich das Objekt neu darstellt, indem es seinen **OnDraw** Handler ruft. Verwenden Sie diese Methode wenn sich Daten, die zur Darstellung des Objekts relevant sind, geändert haben. Wenn das Objekt z.B. eine Pyramide darstellt und die Höhe der Pyramide hat sich geändert, dann müssen Sie die Dirty-Methode rufen, damit das Objekt die Pyramide mit der neuen Höhe zeichnet.

Syntax im BASIC Code:		**<obj>.Dirty**

Die Dirty Methode arbeitet auch im gepufferten Modus. Das Objekt gibt die alte gepufferte Grafik automatisch frei und speichert die neue ab.

Tipps und Tricks

- Per Default ist im OnDraw-Handler der LAYOUT Modus eingestellt. Das bedeutet, dass Textausgaben, die über den Rand des Ausgabefensters hinausgehen, keine neue Zeile eröffnen und beliebige, auch negative Cursor-Koordinaten erlaubt sind. Alternativ können Sie im OnDraw-Handler den PAGE Modus aktivieren, bei dem die Textausgabe auf das aktuelle Textfenster begrenzt wird.
  PAGE und LAYOUT Modus werden durch Ausgeben eines speziellen Steuerzeichens mit dem Print-Befehl aktiviert:

PAGE Modus	Print "\17"	oder	Print Chr$(17)
LAYOUT Modus	Print "\19"	oder	Print Chr$(19)

  In der KeyCodes Library sind entsprechende Konstanten definiert.
  Achtung! Der SCROLL-Modus (Print "\18") wird von Canvasobjekten nicht unterstützt und kann zu seltsamen Ergebnissen führen.

- Canvasobjekte unterstützen die Zwischenablage nur im gepufferten Modus. Die Methoden ClpTestCopy und ClpTestPaste fragen daher als erstes ab, ob der gepufferte Modus aktiv ist. ClpCopy funktioniert im gepufferten Modus immer, ClpPaste akzeptiert Bitmaps und GStrings in der Zwischenablage.
  Beachten Sie, dass das Objekt seine Größe nach einer "Paste" Operation nicht an die neue Grafik anpasst. Unter Umständen wird die eingefügte Grafik über den Rand des Objekts hinaus gezeichnet.
  Im normalen Modus wird keine Clipboardarbeit unterstützt. ClpCopy und ClpPaste setzen die globale Variable **clipboardError** auf TRUE.

- Vermeiden Sie die Verwendung des Befehls CLS im OnDraw-Handler. CLS ignoriert eventuelle Koordinatentransformationen und zeichnet ein Rechteck in der aktuellen Hintergrundfarbe. Diese Farbe ist meist nicht identisch mit der "Hintergrundfarbe" ihres GEOS-Systems. Im gepufferten Modus werden zudem vorher gezeichnete Objekte nicht entfernt, sondern nur überdeckt.

- Ein Canvas-Objekt kann mit der Anweisung PrintObj gedruckt (also auf einen Drucker ausgegeben) werden. Diese Anweisung ist im Objekthandbuch, Kapitel 4.14.7 (Drucken spezieller Objekte) beschrieben. Das Canvas-Objekt muss dazu im gepufferten Modus arbeiten.

(Leerseite)

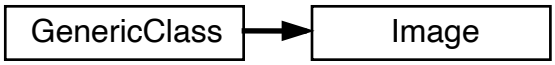
4.17 Image

4.17.1 Überblick

Ein Imageobjekt dient dazu, eine Grafik anzuzeigen, die als fertiges Bild in einer Datei oder in der PictureList vorliegt. Der Programmierer muss nichts weiter tun als die anzuzeigende Grafik zu spezifizieren. Um den Rest kümmert sich das Imageobjekt. Sie können dem Objekt auch jederzeit eine andere Grafik zuweisen, es gibt keinen speziellen Befehl um den von der aktuellen Grafik eventuell belegten Speicher freizugeben. Darum kümmert sich das Objekt automatisch.
Das Imageobjekt kann auch Animationen ohne weiteres Zutun des Programmierers abspielen. Dabei können Sie festlegen, ob die Animation automatisch oder erst durch einen Programmbefehl gestartet werden soll.

Unterstützte Grafik-Formate:
- Bitmap Dateien: JPG, BMP, ICO, PCX, GIF, TGA, RLE, DIB, SCR (BreadBox SplashScreen)
- Animationen: GIF, FLC, FLI, BreadBox QuickCam Format
- Resource-Maker Dateien (Bitmaps und GStrings)
- Sonstige: GEOS Hintergrund-Dateien

Abstammung:



Spezielle Instance-Variablen:

Variable	Syntax im UI-Code	Im BASIC-Code
ImageFile	ImageFile = **stdPath**, **"file"**	nur schreiben
ImagePicture	ImagePicture = **"name"**	nur schreiben
ImageResource	ImageResource = **stdPath**, **"file"**, **"name"**	nur schreiben
numPicts	—	nur lesen
pictNum	pictNum = **n**	lesen, schreiben
imgInfo	—	nur lesen
imgState	—	nur lesen
scale	scale = **scaleX**, **scaleY**	lesen, schreiben
drawPos	drawPos = **x0**, **y0**	lesen, schreiben
borderColor	borderColor = **ul**, **br**	lesen, schreiben
bgColor	bgColor = **col**, **drawMask**	lesen, schreiben
autoSize	autoSize = TRUE | FALSE	lesen, schreiben
autoStart	autoStart = TRUE | FALSE	lesen, schreiben
currentFrame	—	lesen, schreiben
numFrames	—	nur lesen
animationTics	—	lesen, schreiben

Methoden:

Methode	Aufgabe
Redraw	Weist das Objekt an, sich neu darzustellen
AnimationStart	Startet die Animation
AnimationStop	Stoppt die Animation
AnimationNext	Ruft den nächsten Frame der Animation auf

Das Imageobjekt erbt alle Eigenschaften und Fähigkeiten der GenericClass. Von besonderer Bedeutung sind dabei die Fähigkeiten zum Geometriemanagement (siehe Kapitel 3.3). Insbesondere die Hints **fixedSize**, **initialSize**, **ExpandWidth** und/oder **ExpandHeight** werden häufig genutzt, da das Image Objekt "von sich aus" keine vorgegebene Größe hat und sonst möglicherweise unsichtbar klein ist. Beachten Sie in diesem Zusammenhang auch die Instancevariable "**autoSize**".

Mausunterstützung

Das Imageobjekt unterstützt die Behandlung von Mausereignissen. Eine detaillierte Beschreibung der Arbeit mit der Maus finden Sie im Handbuch "Spezielle Themen", Kapitel 17.
Es ist möglich, innerhalb eines Maushandlers Grafiken oder Text auf den Bildschirm auszugeben. Dazu müssen Sie das Objekt explizit zum Screen machen. Häufig werden Sie außerdem die Maus grabben. Ein auf das Imageobjekt übertragbares Beispiel finden Sie hier: "Beispiel\Objekte\Grafik\Canvas Maus Demo". Beachten Sie aber, dass das Imageobjekt die Grafik- und Textausgaben nicht abspeichert! Sie werden nicht wieder gezeichnet, wenn das Objekt sich selbst neu darstellt.

Clipboard

Das Imageobjekt kann die aktuell dargestellte Grafik mit der Methode ClpCopy in die Zwischenablage kopieren. Diese Methode ist für alle Objektklassen definiert. Lesen aus der Zwischenablage wird nicht unterstützt.
Um herauszufinden, ob das Objekt eine Grafik darstellt, die ins Clipboard kopiert werden kann, können Sie die Methode ClpTestCopy verwenden. Diese Methode liefert TRUE wenn keine Grafik spezifiziert wurde, wenn die Grafikdatei nicht gelesen werden konnte oder in jedem anderen Fehlerfall.

Kopieren oder Drucken der Grafik

Um die aktuell dargestellte Grafik auf einen Drucker auszugeben verwenden Sie bitte die Routine PrintObj, die im Kapitel 4.14.7 (Drucken Spezieller Objekte) im Objekthandbuch beschrieben ist. Diese Routine können Sie auch verwenden, um die aktuell angezeigte Grafik auf das aktuelle Screenobjekt zu "drucken". Der Screen kann dabei eine Bitmap, ein GString oder was auch immer sein. Alternativ können Sie die Grafik zunächst ins Clipboard kopieren und vorn dort z.B. mit den Befehlen ClipboardGetGS oder ClipboardGetBitmap holen.

4.17.2 Anzeige von Bildern

Grafiken für das Image-Objekt können aus drei Quellen stammen: aus externen Bilddateien (Instancevariable **ImageFile**), aus der PictureList (Instancevariable **ImagePicture**) oder aus einer Resource-Datei (Instancevariable **ImageResource**). Außerdem haben Sie die Möglichkeit, Informationen über die angezeigte Grafik oder die Grafikdatei zu erhalten (Instancevariablen **numPicts**, **pictNum**, **imgInfo** und **imgState**).
Die Instancevariable ImageFile kann auch eine Animation spezifizieren. Die Details dazu werden im nächsten Abschnitt besprochen.
Die Methode **Redraw** ermöglicht ein Neuzeichnen der angezeigten Grafik.

ImageFile

ImageFile enthält den kompletten Pfad zur anzuzeigenden Datei. Wenn das Programm startet öffnet das Imageobjekt die Datei, lädt (kopiert) die Grafik und schließt die Datei wieder. Um zu prüfen, ob das Objekt ein Bild geladen hat können Sie die Instancevariable "numPicts" abfragen.
Wenn ImageFile eine Animation spezifiziert kann die Animation automatisch gestartet werden (siehe unten, Instancevariable "autoStart"). Während die Animation läuft bleibt die Datei die ganze Zeit offen.

Syntax	UI- Code:	**imageFile** = **stdPath**, **"file"**
	Schreiben:	**<obj>.imageFile** = **stdPath**, **"file"**
	Lesen:	—
	stdPath:	Standardpfad-Konstante oder Null, siehe Tabelle.
	"file"	Name der anzuzeigenden Datei. Pfadangaben sind zulässig. Siehe Tabelle.

Wird der Wert zur Laufzeit zugewiesen stellt sich das Objekt sofort neu dar. Außerdem wird die globale Variable fileError gesetzt.

Zulässige Kombinationen für stdPath und "file"

stdPath	"file"	Anmerkung
Standardpfad Konstante	relativer Pfad zu einer Datei z.B. SP_USER_DATA, "Bild.PCX" SP_DOCUMENT, "Bilder\\Bild.BMP"	
Null	Absoluter Pfad zu einer Datei z.B. 0, "D:\\Bilder\\Tools\\Circle.ICO"	
Null	relativer Pfad zu einer Datei d.h. relativ zum aktuellen Verzeichnis z.B. 0, "Baum.BMP" 0, "Pflanzen\\Baum.BMP"	Nicht im UI Code zulässig

Beispiele

```
Image DemoImage
  imageFile = SP_DOCUMENT, "IMAGES\\SUNSET.JPG"
End Object
```

```
Image DemoImage
  imageFile = SP_USER_DATA, "BACKGRND\\Froggy Bumps"
End Object
```

```
  DemoImage.imageFile = 0, "D:\\Bilder\\TEST3.RLE"
```

ImagePicture

ImagePicture enthält den Namen einer Grafik aus der PictureList. Die Grafik kann eine Bitmap oder ein GString sein.
Die Verwendung der PictureList wird im Programmierhandbuch, Kapitel 2.8.6.2 (Verwendung der "PictureList") beschrieben.

Syntax	UI- Code:	**imagePicture** = **"name"**
	Schreiben:	**<obj>.imagePicture** = **"name"**
	Lesen:	—
	"name"	Name der Grafik in der PictureList

Wird der Wert zur Laufzeit zugewiesen stellt sich das Objekt sofort neu dar. Um zu prüfen, ob das Objekt das Bild in der PictureList gefunden hat können Sie die Instancevariable "numPicts" abfragen.

Beispiele

```
Image DemoImage
  imagePicture = "Segler"
End Object
```

```
  DemoImage.imagePicture = "Diagram"
```

ImageResource

ImageResource ermöglicht dem Imageobjekt Grafiken (Bitmaps und GStrings) aus einer Resource-Maker Datei ohne weitere Unterstützung des Programmierers anzuzeigen. Der Resource-Maker ist © by Rabe-Soft und kann von der Website des Programmierers (www.rbettsteller.de) heruntergeladen werden.

Wenn das Programm startet öffnet das Imageobjekt die Datei, lädt (kopiert) die Grafik und schließt die Datei wieder. Um zu prüfen, ob das Objekt ein Bild geladen hat können Sie die Instancevariable "numPicts" abfragen.

Syntax	UI- Code:	**imageResource** = **stdPath**, **"file"**, **"name"**
	Schreiben:	**<obj>.imageResource** = **stdPath**, **"file"**, **"name"**
	Lesen:	—
	stdPath:	Standardpfad-Konstante oder Null, siehe ImageFile.
	"file"	Name der Resource-Datei. Pfadangaben sind zulässig. Siehe ImageFile.
	"name"	Name des Grafik-Eintrags in der Resource-Datei.

Wird der Wert zur Laufzeit zugewiesen stellt sich das Objekt sofort neu dar. Außerdem wird die globale Variable fileError gesetzt.

Achtung! Falls die spezifizierte Datei keine gültige Resource-Datei ist crasht das System! Sie können im Zweifelsfall das Token abfragen.

Beispiele

```
Image DemoImage
  imageResource = SP_DOCUMENT, "IMAGES\\Test Resource", "Erde"
End Object
```

```
Image DemoImage
  imageResource = SP_USER_DATA,\
             "R-BASIC\\BIN\\Rainer\\SuperGame\\Super ImgResorce",\
             "SiegerAnimation"
End Object
```

```
  DemoImage.imageResource = 0, "D:\\TestResource", "Grafik1"
```

numPicts

Die Instancevariable numPicts enthält die Anzahl der Bilder in der Bilddatei. Kommt das Bild aus einer Resource oder aus der PictureList enthält numPicts den Wert 1. Im Fehlerfall oder wenn noch kein Bild zugewiesen wurde enthält numPicts den Wert Null. Sie können numPicts verwenden, um zu prüfen ob das Objekt ein Bild anzeigt oder nicht. Außerdem können Sie die globale Variable fileError abfragen. Verwenden Sie die Routine ErrorText$() um den Fehlercode in fileError in einen verständlichen Text zu übersetzen.

Syntax	UI- Code:	—
	Schreiben:	—
	Lesen:	**<numVar>** = **<obj>.numPicts**

pictNum

Die Instancevariable pictNum enthält die Nummer des gerade angezeigten Bildes für den Fall, dass die Datei mehr als ein Bild enthält. Das kann z.B. bei ICO Dateien zutreffen. Das erste Bild hat die Nummer Null. Es gilt also immer pictNum < numPicts. Für Bilder aus einer Resource oder aus der PictureList ist der Wert immer Null. Weisen Sie der Instancevariablen einen ungültigen Wert zu, so wird sie automatisch auf Null gesetzt.

Syntax	UI- Code:	**pictNum** = **num**
	Schreiben:	**<obj>**.**pictNum** = **num**
	Lesen:	**<numVar>** = **<obj>**.**pictNum**

ImgInfo

Die Instancevariable imgInfo liefert eine Struktur des Typs GraphicInfo. Diese enthält detaillierte Informationen über das aktuell vom Objekt angezeigte Bild. Die Struktur GraphicInfo ist im Anhang C beschrieben.

Syntax	Lesen:	**<var>** = **<obj>**.**imgInfo**
	var:	Variable vom Typ GraphicInfo

Die Struktur GraphicInfo ist wie folgt definiert:

```
STRUCT GraphicInfo
  sizeX as WORD
  sizeY as WORD
  bitsPerPixel as WORD
  numImages as WORD
End STRUCT
```

Hinweis:
Die von der Instancevariablen imgInfo gelieferten Daten können sich von denen, die von der Routine GetImageInfo geliefert werden (siehe Programmierhandbuch, Kapitel 2.8.6.3 Externe Bilddateien), unterscheiden. Das kann mehrere Gründe haben:

- Bei Dateien, die mehrere Bilder enthalten können (z.B. ICO oder GIF Dateien) liefert GetImageInfo immer die Informationen für das erste Bild, imgInfo jedoch die Informationen für das vom Image-Objekt gerade angezeigte Bild. Dadurch können sich sowohl die Farbtiefe (GraphicInfo Feld bitsPerPixel) als auch die Bildgröße (GraphicInfo Felder sizeX und sizeY) unterschieden.
- Bilder die mit 4 Bit per Pixel oder mit 32 Bit per Pixel kodiert sind werden vom Image-Objekt mit 8 bzw. 24 Bit per Pixel dargestellt. Die Instancevariable imgInfo liefert deswegen niemals eine Farbtiefe von 4 oder 32 Bit, die Routine GetImageInfo hingegen schon.

ImgState

Die Instancevariable imgState enthält die Information, welche Art von Bild das Objekt gerade darstellt. Der Wert kann nur gelesen werden.

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

ImgState liefert einen der folgenden Werte zurück:

Konstante	Wert	Bedeutung
IMGS_NO_IMAGE	0	Kein Bild oder Fehler
IMGS_BITMAP	1	Standbild, Bitmap
IMGS_GSTRING	2	Standbild, GString
IMGS_ANIMATION_RUNNING	3	Laufende Animation
IMGS_ANIMATION_PAUSE	4	Pausierte Animation

Redraw

Die Methode Redraw bewirkt, dass sich das Objekt neu auf dem Bildschirm darstellt. Der Aufruf der Methode ist nur selten notwendig. Ein Beispiel wäre, wenn Sie in einem Maushandler etwas auf den Screen gezeichnet haben und das einfach wieder löschen wollen.

Syntax: **<obj>**.**Redraw** [**drawBackground**]
drawBackground: TRUE | FALSE (Default: FALSE)

Beispiel:

```
DemoImage.Redraw
DemoImage.Redraw  TRUE
```

DrawBackground = TRUE bewirkt, dass das Objekt seinen Hintergrund ebenfalls neu zeichnet. Das kann erforderlich sein, wenn die Grafik transparente Anteile enthält oder Sie die Instancevariablen drawPos, borderColor und / oder bgColor verwendet haben.

4.17.3 Spezielle Attribute

Neben dem "einfachen" Darstellen von Bildern und Animationen können Sie die Eigenschaften des Imageobjekts in gewissen Grenzen einstellen. Insbesondere können Sie festlegen, dass die Grafik vergrößert oder verkleinert dargestellt wird (Instancevariable **scale**), an eine andere Position als die linke obere Ecke gezeichnet wird (Instancevariable **drawPos**), Sie können einen Rahmen um die Grafik zeichnen (Instancevariable **borderColor**) und einen farbigen Hintergrund festlegen (Instancevariable **bgColor**). Außerdem können Sie festlegen, dass das Objekt seine Größe an die Größe der dargestellten Grafik anpassen soll (Instancevariable **autoSize**).

Scale

Scale enthält einen Faktor, um den die Grafik bei der Darstellung gestreckt oder gestaucht wird. In den meisten Fällen brauchen Sie keinen Skalierungsfaktor zu setzen, weil der Defaultwert in x- und in y-Richtung 1 ist.

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

Beispiel: Grafik in doppelter Größe darstellen:

```
Image DemoImage
  imageFile = SP_DOCUMENT, "IMAGES\\SUNSET.JPG"
  scale = 2, 2
End Object
```

DrawPos

Die Instancevariable drawPos enthält die Koordinaten, auf die die linke obere Ecke der Grafik gezeichnet werden soll. Der Defaultwert ist (0; 0). Negative Koordinaten sind zulässig.

Syntax	UI- Code:	**drawPos** = **x0**, **y0**
	Schreiben:	**<obj>**.**drawPos** = **x0**, **y0**
	Lesen:	**<numVar>** = **<obj>**.**drawPos**(n)
		n= 0: x-Position lesen
		n= 1: y-Position lesen
	x0, y0	Koordinaten

borderColor

Die Instancevariable borderColor enthält Farbwerte um einen dünnen Rahmen um das Objekt zu zeichnen. Die Defaultwerte sind jeweils –1, das heißt per Default wird kein Rahmen gezeichnet.
Der Rahmen wird immer um das ganze Objekt gezeichnet, auch wenn die Grafik kleiner als Objekt ist und/oder nicht auf die Position (0; 0) gezeichnet wird.

Syntax	UI- Code:	**borderColor** = **ltCol**, **rbCol**
	Schreiben:	**<obj>**.**borderColor** = **ltCol**, **rbCol**
	Lesen:	**<numVar>** = **<obj>**.**borderColor**(n)
		n= 0: ltCol lesen
		n= 1: brCol lesen
	ltCol:	Farbwert für die Linien links und oben (left, top)
		-1: Links und oben keinen Rahmen zeichnen (Default)
	rbCol:	Farbwert für die Linien rechts und unten (right, bottom)
		-1: Rechts und unten keinen Rahmen zeichnen (Default)
	Für ltCol und rbCol sind nur Indexfarben zulässig.	

bgColor

Mit der Instancevariablen bgColor kann man eine Hintergrundfarbe für die Grafik festlegen. Das kann sinnvoll sein, wenn die Grafik transparente Anteile enthält. Der Defaultwert für den Parameter col ist –1, das heißt es wird die vom System vorgegeben Hintergrundfarbe ohne Füllmuster verwendet.

Syntax	UI- Code:	**bgColor** = **col**, **pattern**
	Schreiben:	**<obj>**.**bgColor** = **col**, **pattern**
	Lesen:	**<numVar>** = **<obj>**.**bgColor**(n)
		n= 0: Farbe col lesen
		n= 1: Füllmuster pattern lesen
	col:	Farbwert für den Hintergrund (nur Indexfarben erlaubt)
		-1: Systemhintergrund verwenden (Default)
	pattern:	Füllmuster (erlaubte Werte: siehe Tabelle unten)

BgColor ist für alle GenericClass Objekte definiert und erwartet einen Farbwert für das unselektierte und einen für das selektiert Objekt. Da dies bei Image-Objekten nicht sinnvoll ist wird der zweite Parameter als Füllmuster-Wert interpretiert. Wenn Sie kein Muster, sondern eine vollständig gefüllte Fläche wünschen, *müssen* Sie die Konstante DM_100 (Wert: 25) als zweiten Parameter verwenden.
Füllmuster werden im Programmierhandbuch, Kapitel 2.8.4 (Die Systemvariable "graphic": Mixmodes und mehr) beschrieben. Im Anhang C finden Sie verschiedene Beispiele für die von GEOS bereitgestellten Füllmuster.

Tabelle: Erlaubte Werte für Füllmuster

Wert	Konstante	Bedeutung
0 - 24	–	Von GEOS bereitgestellte Muster.
25	DM_100	"Normalzustand", 100% Deckung.
26 - 88	–	Unterschiedliche "Transparenzgrade". Größere Werte entsprechen höherer Transparenz.
89	DM_0	Null % Deckung, vollständig transparent.
128	DM_INVERSE	Wird zu einem der anderen Werte addiert. Das Muster wird invertiert.

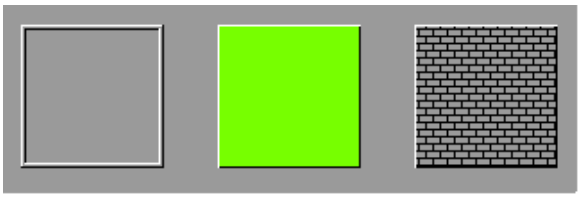
Beispiele:
Die folgenden Bilder setzen folgende Objektdeklaration voraus:

```
Image DemoImage
  imageFile = 0, "D:\\Bilder\\TEST3.RLE"
  fixedSize = 80, 80
End OBJECT
```

Von links nach rechts wurden folgende Zeilen hinzugefügt:

- keine (Standardansicht, Grafik auf Position (0; 0))
- drawPos = 8, 8
  borderColor= WHITE, BLACK
- drawPos = 8, 8
  borderColor= WHITE, BLACK
  bgColor = LIGHT_BLUE, DM_100

Die Umrandung und der Hintergrund (Instancevariablen borderColor und bgColor) werden auch dann gezeichnet, wenn dem Objekt keine Grafik zugewiesen ist (oder wenn die Grafik ungültig ist). Das Behandeln von Mausereignissen ist in allen diesen Fällen trotzdem möglich. Das kann man benutzen um sehr einfach eine farbige Fläche zu erzeugen. Bitte beachten Sie das Füllmuster im dritten Beispiel.



Von links nach rechts wurden folgende Zeilen hinzugefügt:

- keine (Standarddarstellung im Fehlerfall)
- borderColor= WHITE, BLACK
  bgColor = LIGHT_GREEN, DM_100
- borderColor= WHITE, BLACK
  bgColor = BLACK, 8 ' Versuchen Sie auch bgColor = RED, 8 + 128

autoSize

Die Instancevariable autoSize bestimmt, ob das Objekt seine Größe an die dargestellte Grafik anpassen soll. Der Defaultwert ist FALSE (Größe nicht automatisch berechnen).
Hat autoSize den Wert TRUE so berechnet das Objekt seine Größe automatisch neu, wenn das Programm startet oder wenn es eine neue Grafik darstellen soll.
Intern setzt das Objekt den Hint "fixedSize" um seine Größe festzulegen. Dabei kommen folgende Formeln zur Anwendung:

Breite = breite_der_grafik + 2 * drawPos_x
Höhe = höhe_der_grafik + 2 * drawPos_y

Durch dieses Vorgehen wird ein "Rahmen" um die eigentliche Grafik erzeugt, wenn Sie die Instancevariable "drawPos" verwenden.

Syntax	UI- Code:	**autoSize** = **TRUE** Der Defaultwert ist FALSE.
	Schreiben:	**<obj>.autoSize** = **TRUE** | **FALSE**
	Lesen:	**<numVar>** = **<obj>.autoSize**

Sie können die Größe des Imageobjekts bei der automatischen Berechnung begrenzen, indem Sie die Hints "minimumSize" und "maximumSize" verwenden. Der Berechnungsalgorithmus prüft, ob einer oder beide dieser Hints gesetzt sind und schränkt die Werte für Breite und Höhe des Objekts so ein, dass die minimalen und maximalen Werte nicht überschritten werden. Diese Werte werden dann an den Hint "fixedSize" übergeben.

Wichtige Hinweise:

- Bei der Berechnung der Größe wird der Skalierungsfaktor (Instancevariable "scale") nicht berücksichtigt.
- Wählen Sie aus einer Datei mit mehreren Bildern (z.B. einer ICO Datei) ein neues Bild aus, so berechnet das Imageobjekt seine Größe jedes Mal neu.
- Wenn Sie "autoSize" zu Laufzeit auf TRUE setzen berechnet das Objekt seine Größe umgehend neu und stellt sich neu dar. Das gilt auch, wenn sie bereits TRUE ist.
- Eine Neubelegung der Instancevariable "drawPos" bewirkt keine Neuberechnung der Größe des Objekts.

4.17.4 Animationen

Das Image-Objekt kann Animationen automatisch ohne weiteres Zutun des Programmierers abspielen. Die Berechnung und Darstellung der Bilder erfolgt dabei im Hintergrund, so dass Ihr BASIC Programm ganz normal weiterarbeiten kann.
Um eine Animation mit einem Image-Objekt darzustellen müssen Sie dem Objekt nur eine Datei zuweisen, die eine Animation enthält (Instancevariable ImageFile).
Mit den Methoden **AnimationStart**, **AnimationStop** und **AnimationNext** haben Sie volle Kontrolle über das Abspielen der Animation. Mit der Instancevariablen **autoStart** können Sie festlegen, dass die Animation automatisch startet. Informationen über den aktuellen Status der Animation bekommen Sie mit den Instancevariablen **currentFrame**, **numFrames**, **animationTics** (kann auch gesetzt werden um die Geschwindigkeit zu ändern) sowie den weiter oben beschriebenen Instancevariablen **imgInfo** und **imgState**.

AnimationStart

Die Methode AnimationStart startet eine Animation. Sie können AnimationStart auch zum Fortsetzen einer mit AnimationStop angehaltenen Animation verwenden. Wenn Sie im UI Code die Instancevariable autoStart auf TRUE gesetzt haben startet die Animation beim Laden des Programms automatisch, ohne den Aufruf von AnimationStart.

Syntax: **<obj>**.**AnimationStart**

AnimationStop

Die Methode AnimationStop hält eine laufende Animation an. Falls Sie eine Animation in einer Dialogbox haben sollten Sie nach dem Schließen der Dialogbox AnimationStop rufen, damit die Animation nicht unnötig im Hintergrund weiterläuft.

Syntax: **<obj>**.**AnimationStop**

AnimationNext

Die Methode AnimationNext wählt das nächste Bild einer angehaltenen Animation an. AnimationNext ist wirkungslos bei einer laufenden Animation.

Syntax: **<obj>**.**AnimationNext**

Beispiel. Die im Objekt DemoImage laufende Animation wird für 3 Sekunden gestoppt, dann wird das nächste Bild angezeigt und nach weiteren 3 Sekunden wir die Animation mit normaler Geschwindigkeit fortgesetzt.

```
DemoImage.AnimationStop
Pause 30
DemoImage.AnimationNext
Pause 30
DemoImage.AnimationStart
```

autoStart

Die Instancevariable autoStart bestimmt, ob ein vom Imageobjekt angezeigte Animation beim Starten des Programms oder bei der Zuweisung einer neuen Datei automatisch abgespielt werden soll oder nicht. Der Defaultwert ist FALSE (Animation nicht automatisch starten). Beachten Sie, dass eine gestartete Animation "im Hintergrund" weiterläuft auch wenn das Imageobjekt gerade nicht sichtbar ist. Setzen Sie "autoStart" nur dann auf TRUE, wenn es wirklich notwendig ist.
Wenn Sie "autoStart" zu Laufzeit auf TRUE setzen wird die Animation sofort gestartet.

Syntax	UI- Code:	**autoStart** = **TRUE** Der Defaultwert ist FALSE.
	Schreiben:	**<obj>.autoStart** = **TRUE** | **FALSE**
	Lesen:	**<numVar>** = **<obj>.autoStart**

currentFrame

Die Instancevariable currentFrame enthält die Nummer des aktuell angezeigten Bildes der Animation. Die Zählung beginnt bei Null. Der Wert kann jederzeit gelesen werden. Ein Setzen zur Laufzeit ist nur möglich, wenn die Animation gerade nicht läuft. Im UI-Code kann der Wert nicht gesetzt werden.

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

Hinweis: Das Aufrufen eines speziellen Frames kann manchmal etwas dauern, weil das Aussehen eines Bildes vom Aussehen der vorhergehenden Bilder abhängen kann. Deshalb muss das Image Objekt möglicherweise einen großen Teil der Animation erneut dekodieren um den gewünschten Frame anzuzeigen.

numFrames

Die Instancevariable numFrames enthält die Anzahl der Bilder in einer Animation. Im Fehlerfall enthält numFrames den Wert Null. Außerdem können Sie im Fehlerfall die globale Variable fileError abfragen, um das Problem einzugrenzen. Verwenden Sie die Routine ErrorText$() um den Fehlercode in fileError in einen verständlichen Text zu übersetzen.

Syntax	UI- Code:	—
	Schreiben:	—
	Lesen:	**<numVar>** = **<obj>**.**numFrames**

animationTics

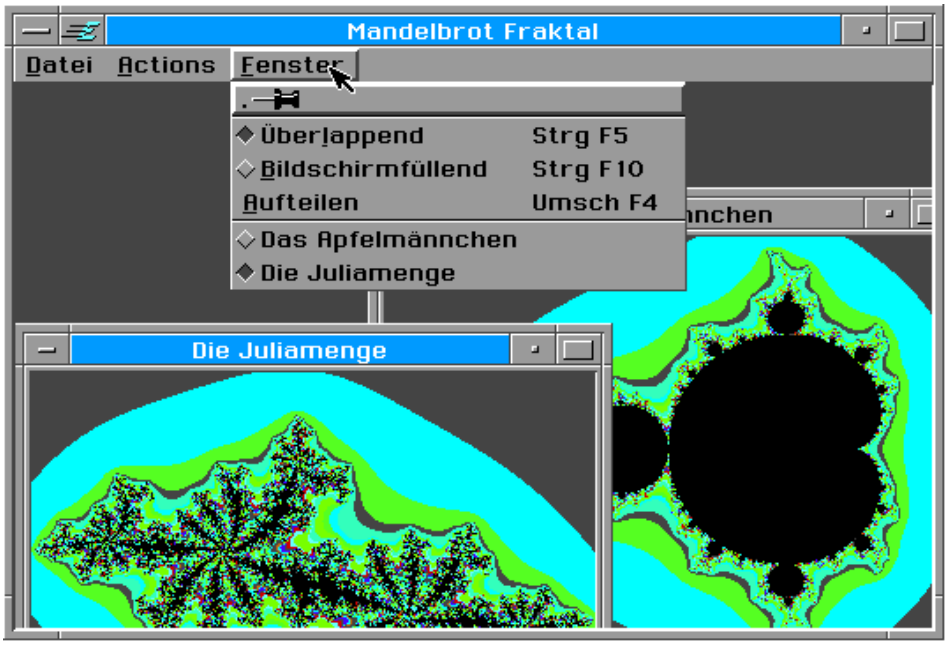
Die Instancevariable animationTics enthält die Zeit zwischen zwei benachbarten Frames einer Animation, gemessen in tics (1 tic = 1/60 s). Je kleiner der Wert ist, desto schneller läuft die Animation. Der Wert kann zur Laufzeit gelesen und geschrieben, aber nicht im UI Code gesetzt werden.
Wird der Wert bei laufender Animation gesetzt so ändert das Imageobjekt die Abspielgeschwindigkeit sofort.

Syntax	UI- Code:	—
	Schreiben:	**<obj>**.**animationTics** = **<Wert>**
	Lesen:	**<numVar>** = **<obj>**.**animationTics**

4.18 Display und zugehörige Objekte

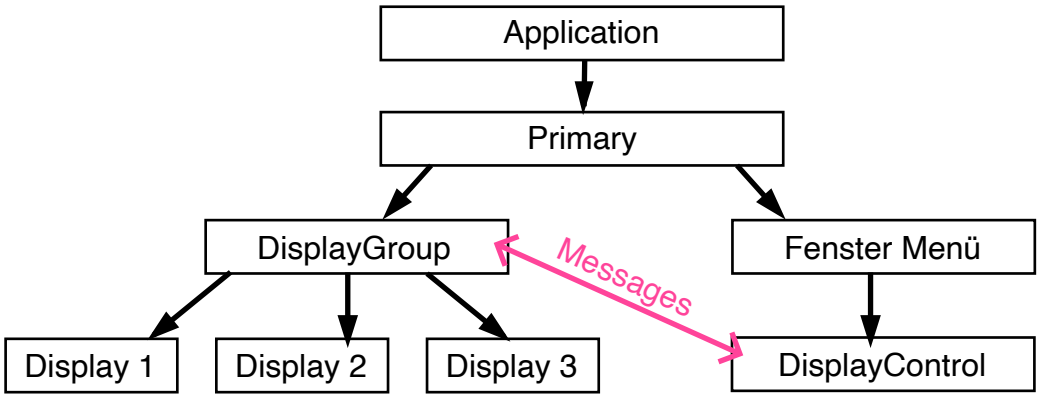
4.18.1 Überblick

Jede große Applikation (z.B. Write, R-BASIC) zeigt ihre Dokumente in eigenen Fenstern an. Diese Fenster gehören der Objektklasse **Display** an. Displayobjekte enthalten ähnlich wie Dialoge weitere UI-Objekte, z.B. Views, Memos, Buttons oder Listen. Der Bereich, in dem die Displays dargestellt werden, ist ein Objekt der Klasse **DisplayGroup**. Die DisplayGroup managet die Displays, organisiert z.B. dass sie überlappend dargestellt werden oder sich den Platz aufteilen. Dabei hat die DisplayGroup selbst keine UI, mit der der Nutzer interagieren kann. Das übernimmt ein Objekt der Klasse **DisplayControl**, das sich üblicher Weise im "Fenster" Menü befindet. Der Vorteil von dieser Trennung ist, dass man mehrere DisplayGroups haben kann, die alle über das Fenster-Menü gesteuert werden.



Eine typische Konfiguration: Eine DisplayGroup mit zwei Displays. Das DisplayControl befindet sich im "Fenster" Menü.

Displays, DisplayGroup und DisplayControl arbeiten im Hintergrund eng zusammen. Im Kern ist es so, dass die DisplayGroup weiß, welche Displays (Fenster) es gibt, da sie die Children der DisplayGroup sind. Sie sendet eine Message an das DisplayControl, das daraufhin eine Liste der vorhandenen Fenster aufbaut. Wählt der Nutzer aus dieser Liste ein Display aus so sendet das DisplayControl eine Message an die DisplayGroup. Diese wiederum wählt das geforderte Display aus. Genauso verhält es sich, wenn der Nutzer den Eintrag "Überlappend", "Bildschirmfüllend" oder "Aufteilen" auswählt. Die DisplayGroup bekommt den Befehl vom DisplayControl und organisiert die Anzeige entsprechend. Im Gegenzug bekommt das DisplayControl eine Message, wenn der Nutzer z.B. auf ein bestimmtes Display klickt und es so zum aktiven Fenster macht. All das passiert ohne weiteres Zutun des Programmierers.

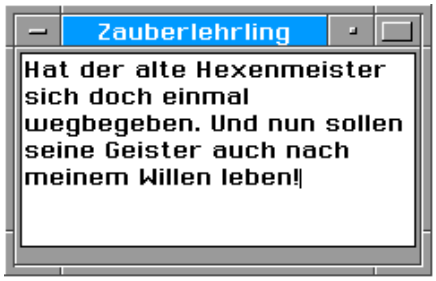


Prinzipielle Organisation von Display, DisplayGroup und DisplayControl

Das System organisiert diese Zusammenarbeit intern über die Target-Hierarchie. Das stellt auch sicher, dass die Zusammenarbeit mit mehreren DisplayGroups funktioniert. Hintergrundinformationen zur Target-Hierarchie finden Sie im Handbuch Spezielle Themen, Vol. 2, Kapitel 12. Für die Verwendung von Display, DisplayGroup und DisplayControl müssen Sie in diesem Zusammenhang folgendes wissen:

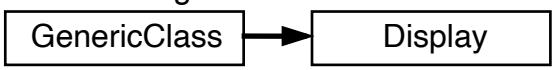
- Sie müssen der DisplayGroup (falls Sie mehrere haben: genau einer) den Hint **defaultTarget** geben. Andernfalls weiß das DisplayControl nicht, mit wem es zusammenarbeiten soll und baut die Fenster-Liste nicht automatisch auf.
- Falls Sie verhindern wollen, dass eine DisplayGroup auf die Messages des DisplayControl reagiert, so müssen Sie ihre Instancevariable **targetable** auf FALSE setzen.

4.18.2 Display



Objekte der Klasse Display sind die "Fenster" in denen alle großen Applikationen ihre Daten anzeigen. Displays müssen Children eines DisplayGroup Objekts sein. Die Displays selbst enthalten weitere UI-Objekte, die die eigentlichen Informationen darstellen. Im Bild links ist das ein Memo-Objekt.

Abstammung:



Das Displayobjekt erbt alle Eigenschaften und Fähigkeiten der GenericClass. Von besonderer Bedeutung sind dabei die Fähigkeiten zum Geometriemanagement, insbesondere die Window-bezogenen Hints (Kapitel 3.3.7: Spezielle Hints für Window-Objekte) sind hier von Bedeutung. Beachten Sie, dass insbesondere die Größe eines Displayobjekts im Zusammenspiel mit der DisplayGroup geändert

werden kann. Es ist deshalb leicht möglich, dass Sie widersprüchliche Hints setzen. So verhindert z.B. der Hint "NotMaximizable", dass sich das Display bildschirmfüllend darstellt.

Displays können in drei Modi dargestellt werden:

- Bildschirmfüllend (maximiert): Das Displayobjekt nimmt den gesamten verfügbaren Platz in der DisplayGroup ein. Wenn ein Displayobjekt bildschirmfüllend dargestellt wird so sind auch alle anderen Displays bildschirmfüllend.
- Überlappend: Jedes Displayobjekt hat seine eigene Größe. Der Nutzer kann die Größe ändern, Displayobjekte können sich gegenseitig überlappen. Der Zustand "aufgeteilt" ist ein Spezialfall von "überlappend", bei dem Größe und Anordnung der Displays automatisch so gewählt wird, dass sie alle möglichst gut zu sehen sind.
- Minimiert: Das Displayobjekt ist nicht mehr sichtbar, aber noch in der Liste des DisplayControl Objekts verfügbar. Von dort aus kann es wieder sichtbar gemacht werden.
  Der Zustand "minimiert" ist nicht identisch mit "unsichtbar" (visible = FALSE). Wird die Instancevariable "visible" auf FALSE gesetzt verschwindet das Displayobjekt auch aus der Liste des DisplayControl Objekts.

Die Instancevariablen **minimizedState** und **maximizedState** bestimmen gemeinsam mit den Hints **MinimizedOnStartup** und **MaximizedOnStartup** in welchem der drei Modi sich das Displayobjekt befindet. Mit den Hints **NotMinimizable**, **NotMaximizable**, **NotResizable** und **NotRestorable** kann man bei Bedarf die Fähigkeiten des Displayobjekts einschränken. Die Instancevariable **userDismissable** und die Methode **Close** wird nur benötigt, wenn man den Mechanismus "Schließen von Displays" (siehe unten) implementieren will.

Spezielle Instance-Variablen:

Variable	Syntax im UI-Code	Im BASIC-Code
minimizedState	minimizedState = TRUE | FALSE	lesen, schreiben
MinimizedOnStartup	MinimizedOnStartup	—
NotMinimizable	NotMinimizable	—
maximizedState	maximizedState = TRUE | FALSE	lesen, schreiben
MaximizedOnStartup	MaximizedOnStartup	—
NotMaximizable	NotMaximizable	—
NotResizable	NotResizable	—
NotRestorable	NotRestorable	—
userDismissable	userDismissable = TRUE | FALSE	lesen, schreiben
OnClose	OnClose = **<Handler>**	lesen, schreiben

Methoden:

Methode	Aufgabe
Close	Ruft den OnClose Handler des Objekts auf

Action-Handler-Typen:

Handler-Typ	Parameter
DialogAction	(sender as object, command as integer)

Ein einfaches Displayobjekt, dass ein Memo als Child enthält, sieht so aus:

```
Display Disp1
  Caption$ = "Erlkönig"' , 0
  Children = Memo1
End OBJECT

Memo Memo1
  text$ = "Wer reitet so spät durch Nacht und Wind?"
  ExpandWidth:ExpandHeight
  backColor = WHITE
End Object
```

Minimized und Maximized State

Die folgenden Instancevariablen bestimmen ob das Displayobjekt minimiert (versteckt), maximiert (bildschirmfüllend) oder überlappend dargestellt wird.
Hinweis: Da Primary-Objekte von Displays abstammen erben Sie die im Folgenden aufgelisteten Fähigkeiten.

minimizedState

MinimizedState enthält die Information ob das Displayobjekt "minimiert" ist oder nicht.

- Am Programmstart bestimmt das DisplayGroup-Objekt, wie die Displays dargestellt werden. Das Setzen des Wertes im UI-Code ist möglicherweise wirkungslos. Verwenden Sie in diesem Fall MinimizedOnStartup.

Syntax	UI- Code:	**minimizedState** = TRUE Der Defaultwert ist FALSE.
	Schreiben:	**<obj>.minimizedState** = **<Wert>**
	Lesen:	**<numVar>** = **<obj>.minimizedState**

MinimizedOnStartup

MinimizedOnStartup bewirkt, dass das Displayobjekt am Programmstart minimiert ist.

Syntax	UI- Code:	**MinimizedOnStartup**

NotMinimizable

NotMinimizable bewirkt, dass das Displayobjekt nicht minimiert werden kann. Der entsprechende Button in der Titelbar des Displayobjekts wird entfernt und das Setzen der Instancevariablen minimizedState bleibt wirkungslos.

Syntax UI- Code:	**NotMinimizable**

maximizedState

MaximizedState enthält die Information, ob das Displayobjekt "bildschirmfüllend" (maximizedState = TRUE) dargestellt wird oder nicht.

- Am Programmstart bestimmt das DisplayGroup-Objekt, wie die Displays dargestellt werden. Das Setzen des Wertes im UI-Code ist möglicherweise wirkungslos.
- Wenn Sie den Wert für ein Displayobjekt zur Laufzeit ändern, hat das Auswirkungen auf alle anderen Displays in der DisplayGroup.

Syntax UI- Code:	**maximizedState** = TRUE Der Defaultwert ist FALSE.
Schreiben:	**<obj>.maximizedState** = **<Wert>**
Lesen:	**<numVar>** = **<obj>.maximizedState**

MaximizedOnStartup

MaximizedOnStartup bewirkt, dass das Displayobjekt am Programstart maximiert dargestellt wird.

- Am Programmstart bestimmt das DisplayGroup-Objekt, wie die Displays dargestellt werden. Das Setzen des Wertes ist möglicherweise wirkungslos.

Syntax UI- Code:	**MaximizedOnStartup**

NotMaximizable

NotMaximizable bewirkt, dass der Nutzer das Displayobjekt nicht maximieren kann. Das Display bleibt im "überlappenden" Modus, auch wenn die anderen Displays bildschirmfüllend sind.

Syntax UI- Code:	**NotMaximizable**

Weitere Hints

NotResizable

NotResizable bewirkt, dass der Nutzer die Größe des Displayobjekts nicht ändern kann, wenn es im Modus "überlappend" dargestellt wird.

Syntax UI- Code:	**NotResizable**

NotRestorable

NotRestorable bewirkt, dass ein Wechsel in den "minimiert" Modus nicht zurückgenommen werden kann. Verwenden Sie diesen Hint mit Vorsicht.

Syntax UI- Code:	**NotRestorable**

Schließen von Displays

Große Applikationen wie GeoWrite oder R-BASIC stellen Ihre Dokumente in Fenstern dar, die Display-Objekte sind. Wird ein Dokument geschlossen so muss auch das zugehörige Display-Objekt vom Schirm genommen werden. Falls das Display beim Öffnen des Dokuments, also zur Laufzeit, mit der Routine CreateObject erzeugt wurde muss es dann auch wieder mit der Routine DestroyObject wieder vernichtet werden. Bitte lesen Sie die Dokumentation dieser Routinen sorgfältig.
Werden die Fenster (Display-Objekte) überlappend dargestellt so findet sich im Systemmenü des Displays der Eintrag "Schließen". Er ist per Default inaktiv. Um ihn zu aktivieren müssen Sie die Instancevariable userDismissable des Display-Objekts auf TRUE setzen. Klickt der Nutzer jetzt auf diesen Eintrag wird der OnClose Handler des Displayobjekts aufgerufen. Dieser Handler muss alle notwendigen Schritte auslösen um das Dokument zu schließen und das Display-Objekt vom Schirm zu nehmen. Ist kein OnClose Handler gesetzt so passiert nichts.

Hinweis: Primary-Objekte stammen von Displays ab. Sie implementieren jedoch ihr eigenes Handling zum Schließen eines Programms. Die folgenden Instancevariablen sind für Primaries daher nicht verfügbar.

userDismissable

UserDismissable = TRUE aktiviert den Eintrag "Schließen" im Systemmenü des Displayobjekts. Der Defaultwert ist FALSE. Das Systemmenü ist nur sichtbar, wenn die Displays überlappend dargestellt sind.
Wenn der Nutzer auf "Schließen" im Systemmenü des Displayobjekts klickt wird der OnClose Handler des Objekts aufgerufen.

Syntax	UI- Code:	**userDismissable** = TRUE Der Defaultwert ist FALSE.
	Schreiben:	**<obj>.userDismissable = <Wert>**
	Lesen:	**<numVar> = <obj>.userDismissable**

OnClose

Der OnClose Handler wird gerufen, wenn der Nutzer auf den Eintrag "Schließen" im Systemmenü des Displayobjekts klickt. Das Systemmenü ist nur sichtbar, wenn die Displays überlappend dargestellt sind.
Um den Eintrag "Schließen" im Systemmenü des Displayobjekts zu aktivieren müssen Sie die Instancevariable userDismissable des Displayobjekts auf TRUE setzen.
Der OnClose Handler muss als **DialogAction** deklariert sein.

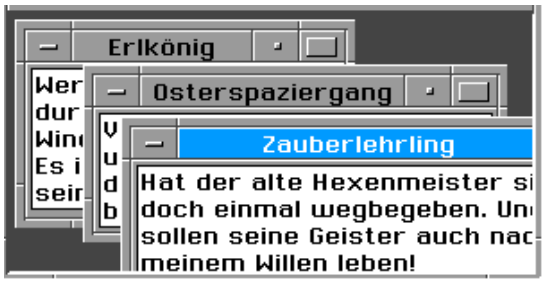
Syntax	UI- Code:	**OnClose = <Handler>**
	Schreiben:	**<obj>.OnClose = <Handler>**
	Lesen:	–

Close

Die Methode Close ruft den OnClose Handler des Objekts auf. Dieser Handler muss dann alle weiteren Schritte auslösen. Ist kein OnClose Handler definiert so passiert auch nichts.

Syntax	BASIC Code:	**<obj>.Close**

4.18.3 DisplayGroup

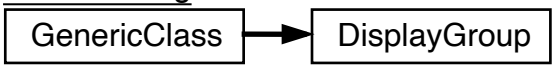


Eine DisplayGroup stellt den Bereich bereit, in dem die Displayobjekte dargestellt werden. Im Bild sind drei Displays in einer DisplayGroup zu sehen.
Das DisplayGroup Objekt interagiert mit den Displays um sie anzuordnen, ihre Größe festzulegen usw.

Außerdem arbeitet das DisplayGroup Objekt automatisch mit dem DisplayControl Objekt zusammen. Damit dies alles funktioniert müssen Sie Folgendes tun:

- Die Displays müssen Children des DisplayGroup Objekts sein.
- Das DisplayGroup Objekt muss den Hint **defaultTarget** gesetzt haben.

Abstammung



Per Default ist eine DisplayGroup so eingestellt, dass die Displays am Programmstart im bildschirmfüllenden Modus angezeigt werden. Wenn Sie das nicht möchten setzen Sie im UI-Code die Instancevariable fullSizeState auf FALSE. Um die Displays am Programmstart "aufgeteilt" darzustellen müssen Sie in Ihrem OnStartup Handler die Methode "TileDisplays" für das DisplayGroup Objekt aufrufen.

Spezielle Instance-Variablen:

Variable	Syntax im UI-Code	Im BASIC-Code
fullSizeState	fullSizeState = TRUE | FALSE	lesen, schreiben
activeDisplay	—	nur lesen
NoFullSizeMode	NoFullSizeMode	—
NoOverlappingMode	NoOverlappingMode	—
TileHorizontally	TileHorizontally	—
TileVertically	TileVertically	—
SizeIndependentlyOfDisplays	SizeIndependentlyOfDisplays	—

Methoden:

Methode	Aufgabe
TileDisplays	Ordnet die Displays "aufgeteilt" an
SelectDisplay (n)	Wählt ein Display als aktives Display aus

Eine typische Konfiguration einer DisplayGroup sieht so aus:

```
DisplayGroup DGroup
  children = Disp1, Disp2, Disp3
  initialSize = 800, 400
  defaultTarget
  fullSizeState = FALSE
  ExpandWidth
  ExpandHeight
  SizeIndependentlyOfDisplays
End OBJECT
```

FullSizeState

Die Instancevariable fullSizeState enthält die Information, ob die Displays in der DisplayGroup überlappend (fullSizeState = FALSE) oder bildschirmfüllend (fullSizeState = TRUE) dargestellt werden. Sie können den Wert zur Laufzeit ändern um den entsprechenden Zustand einzustellen.
Um die Displays gleichmäßig in der DisplayGroup aufzuteilen verwenden Sie die Methode "TileDisplays" (siehe unten).

Syntax	UI- Code:	**fullSizeState** = FALSE Der Defaultwert ist TRUE
	Schreiben:	**<obj>.fullSizeState** = TRUE I FALSE
	Lesen:	**<numVar>** = **<obj>.fullSizeState**

activeDisplay

Die Instancevariable activeDisplay enthält das aktuell aktive Displayobjekt. Ist kein Displayobjekt "aktiv" enthält activeDisplay das zuletzt aktive Displayobjekt. Sollte die DisplayGroup keine Displays enthalten so liefert activeDisplay ein Null-Objekt.

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

NoFullSizeMode

NoFullSizeMode verhindert, dass die Displays in der DisplayGroup bildschirmfüllend angezeigt werden.
Am Programmstart werden die Displays per Default trotzdem immer bildschirmfüllend angezeigt. Setzen Sie daher im UI-Code zusätzlich die Instancevariable fullSizeState auf FALSE.

Syntax	UI-Code:	**NoFullSizeMode**

NoOverlappingMode

NoOverlappingMode verhindert, dass die Displays in der DisplayGroup überlappend angezeigt werden.

Syntax UI-Code:	**NoOverlappingMode**

TileDisplays

Die Methode TileDisplays ordnet die Displays "aufgeteilt" an. Der Aufruf dieser Methode hat die gleiche Wirkung als ob der Nutzer im DisplayControl den Eintrag "Aufteilen" anklickt.

Syntax BASIC Code:	**<obj>.TileDisplays**

TileHorizontally

TileHorizontally bewirkt, dass die Displays in der DisplayGroup nebeneinander angeordnet werden, wenn sie "aufgeteilt" werden.
Um die Displays aufzuteilen kann der Nutzer im DisplayControl den entsprechenden Eintrag anklicken oder man ruft die Methode "TileDisplays" auf.

Syntax UI-Code:	**TileHorizontally**

TileVertically

TileVertically bewirkt, dass die Displays in der DisplayGroup übereinander angeordnet werden, wenn sie "aufgeteilt" werden.
Um die Displays aufzuteilen kann der Nutzer im DisplayControl den entsprechenden Eintrag anklicken oder man ruft die Methode "TileDisplays" auf.

Syntax UI-Code:	**TileVertically**

SizeIndependentlyOfDisplays

Sowohl die Anordnung der Displays als auch die Größe der DisplayGroup werden zwischen Displays und DisplayGroup automatisch ausgehandelt. In einigen Situationen kann das dazu führen, dass die DisplayGroup nicht so aussieht, wie Sie sich das wünschen, z.B. dass sie zu klein ist, oder dass die Geometrie der Displays nicht stimmt. Verwenden Sie in diesen Fällen den Hint

SizeIndependentlyOfDisplays um die Geometrie der DisplayGroup und die Geometrie der Displays voneinander zu entkoppeln.
Tipp: Setzen Sie diesen Hint immer. Nur wenn Sie den Eindruck haben, dass die Geometrie nicht stimmt versuchen Sie es ohne ihn.

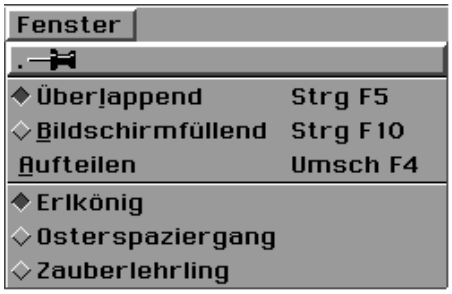
Syntax UI-Code:	**SizeIndependentlyOfDisplays**

SelectDisplay

SelectDisplay(n) wählt ein Display als aktives Display aus. Die Zählung beginnt dabei bei Null.
Tipp: Um herauszubekommen wie viele Displays zu einem DisplayGroup Objekt gehören fragen Sie die Instancevariable numChildren der DisplayGroup ab.

Syntax BASIC Code:	**<obj>.SelectDisplay** (**<Wert>**)
<Wert>	Nummer des auszuwählenden Displays Die Zählung beginnt bei Null.

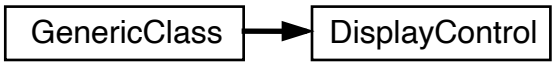
4.18.4 DisplayControl



Das DisplayControl stellt die UI bereit, mit der der Nutzer die Displays in der DisplayGroup anordnen kann. Außerdem enthält es eine Liste mit den Namen (Caption$) der Displays in der DisplayGroup. Wie im Bild zu sehen ist das DisplayControl üblicher Weise ein Child des "Fenster" Menüs.

Das DisplayControl arbeitet automatisch mit den Displays und der DisplayGroup zusammen. Das funktioniert sogar, wenn Sie mehrere DisplayGroup Objekte haben.
Der Programmierer muss dazu nichts weiter tun als die Objekte in seinen Tree einbinden. Außerdem muss das DisplayGroup Objekt (falls Sie mehrere haben: genau eines) den Hint **defaultTarget** gesetzt haben.

Abstammung



Spezielle Instance-Variablen:

Variable	Syntax im UI-Code	Im BASIC-Code
dcFeatures	dcFeatures = **<Wert>**	lesen, schreiben
nameOnPrimaryIfMaximized nameOnPrimaryIfMaximized = TRUE		lesen, schreiben

dcFeatures

DcFeatures bestimmt, welche Elemente der Controller-UI angezeigt werden. Per Default werden alle Elemente angezeigt.

Syntax	UI- Code:	**dcFeatures** = **<Wert>**
	Schreiben:	**<obj>.dcFeatures** = **<Wert>**
	Lesen:	**<numVar>** = **<obj>.dcFeatures**

Folgende Werte sind für dcFeatures verfügbar:

Konstante	Wert	Bedeutung
DCF_OVERLAPP_FULL	4	Auswahl "Überlappend" / "Bildschirm-füllend" anzeigen
DCF_TILE	2	Schalter "Aufteilen" anzeigen
DCF_DISPLAY_LIST	1	Liste aller Displays anzeigen

NameOnPrimaryIfMaximized

Diese Instancevariable bestimmt, ob der Name des aktuell aktiven Displays in der Titelzeile des Primary-Objekts angezeigt werden soll.

- Üblicher Weise wird die Instancevariable im UI-Code belegt.
- Ändern Sie den Wert zur Laufzeit von TRUE auf FALSE während die Displays maximiert sind, so updated das DisplayControl die Titelzeile im Primary nicht mehr. Die Anzeige ist dann möglicherweise fehlerhaft oder veraltet.
- Intern wird die Instancevariable Caption2$ des Primaryobjekts verwendet, um diese Funktion zu realisieren.

Syntax	UI- Code:	**nameOnPrimaryIfMaximized** = TRUE Der Defaultwert ist FALSE.
	Schreiben:	**<obj>.nameOnPrimaryIfMaximized** = **<Wert>**
	Lesen:	**<numVar>** = **<obj>.nameOnPrimaryIfMaximized**

Beispiel. Beachten Sie, dass das DisplayControl keinen Verweis auf die Display-Group enthält. Das wird intern über die Target Hierarchie geregelt.

```
Menu WindowMenu
  Caption$ = "Fenster" , 0
  Children = DControl
End OBJECT

DisplayControl DControl
  nameOnPrimaryIfMaximized = TRUE
End OBJECT
```

(Leerseite)