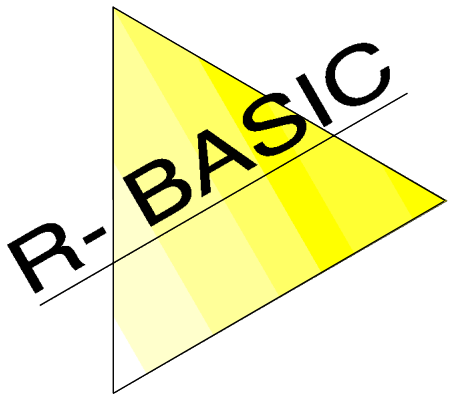
R-BASIC

Einfach unter PC/GEOS programmieren



Objekt-Handbuch

Volume 7
ColorSelector, DocumentGuardian, PrintControl,
PageSizeControl

Version 1.0

(Leerseite)

Inhaltsverzeichnis

(Leerseite)

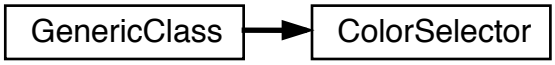
4.12 ColorSelector

Ein ColorSelector stellt die UI bereit, die notwendig ist, um eine Farbe, ein Füllmuster oder ein Füllraster auszuwählen.

Welche der Eigenschaften (Farbe, Füllmuster, Raster) dem Nutzer zur Auswahl angeboten werden kann eingestellt werden. Per Default sind es nur die Indexfarben.

Sie können einen Actionhandler vereinbaren, der gerufen werden soll, wenn der Nutzer eine der Eigenschaften ändert oder Sie können die eingestellten Eigenschaften manuell abfragen.

Abstammung:



Der ColorSelector erbt alle Eigenschaften und Fähigkeiten der GenericClass.

Spezielle Instance-Variablen:

Instancevariable	Syntax im UI-Code	Im BASIC-Code
ColorChangedHandler	ColorChangedHandler = **<Handler>**	nur schreiben
csFeatures	csFeatures = **numVal**	lesen, schreiben
csColor	csColor = **numVal**	lesen, schreiben
csIndexColor	—	nur lesen
csDrawMask	csDrawMask = **numVal**	lesen, schreiben
csDrawMaskValue	csDrawMaskValue = **numVal**	lesen, schreiben
csFillPattern	csFillPattern = **numVal**	lesen, schreiben

Action-Handler-Typen:

Handler-Typ	Parameter
ColorAction	(sender as object, csColor as DWord, csDrawMask as Byte, csFillPattern as Byte)

csFeatures

Die Instancevariable csFeatures bestimmt, welche UI der ColorSelector dem Nutzer anbietet. Per Default ist nur das Bit CSI_INDEX gesetzt, das heißt der ColorSelector bietet nur die Liste mit den ersten 16 Indexfarben an. Die folgenden Werte stehen für csFeatures zur Verfügung:

Konstante	Wert	hex.	Bereitgestellte Eigenschaft
CSF_FILL_PATTERN	1	&h1	Auswahl an Füllmustern
CSF_DRAW_MASK	2	&h2	Füll-Raster (0 bis 100%)
CSF_RGB	4	&h4	Farbauswahl über RGB-Werte
CSF_INDEX	8	&h8	Farbauswahl über Indexfarben
–	16	&h10	Gefüllt / Ungefüllt
CSF_MORE_COLORS	32	&h20	Dialogbox "Weitere Farben"

Beachten Sie, dass csFeatures Bitflags sind. Eine Verknüpfung der einzelnen Werte ist mit dem Operator '+' oder mit einem logischen OR möglich.

Beispiel: Ein ColorSelector der die Indexfarben und das Einstellen von RGB-Werten unterstützt.

```
ColorSelector  DemoColors
  csFeatures = CFS_INDEX OR CSF_RGB
End Object
```

Hinweise:

- Der Selector für "gefüllt / ungefüllt" (csFeatures Wert 16) kann zwar aktiviert werden, unter R-BASIC ist jedoch kein Zugriff darauf möglich.
- Die ColorSelector Tools aus den Menüleisten sind unter R-BASIC nicht verfügbar. Sie können Sie als RadioButtonGroups nachbilden.

csColor, csIndexColor

Die Instancevariable csColor enthält den aktuell vom ColorSelector ausgewählten Farbwert. Es hängt von den Umständen ab, ob csColor beim Auslesen einen RGB-Farbwert oder einen Indexwert liefert. Selbst wenn eine Indexfarbe gewählt ist kann csColor einen (natürlich den zugehörigen) RGB-Farbwert liefern. Wenn Sie sicher sein wollen, dass Sie eine Indexfarbwert erhalten müssen Sie die Instancevariable csIndexColor abfragen. Sollte eine Farbe ausgewählt sein, der kein Farbindex entspricht wird derjenige Index zurückgegeben, der der ausgewählten Farbe am besten entspricht.
Für die Zuweisung an csColor sind sowohl RGB- als auch Indexfarben zulässig, csIndexColor kann nur gelesen werden.

Syntax Lesen: **<numVar>** = **<obj>.csColor**
<numVar> = **<obj>.csIndexColor**
Schreiben: **<obj>.csColor** = **farbwert**
UI-Code: **csColor** = **farbwert**
farbwert: RGB-Farbwert oder Indexfarbe

Die numerischen Werte der Indexfarben liegen im Bereich von Null bis 255. RGB-Farbwerte berechnen sich nach der Formel "rot + 256*grün + 65536*blau + 16777216" oder gleichwertig hexadezimal "rot + &h100*grün + &h10000*blau + &h1000000". Mehr zum Thema Farben finden Sie im Kapitel 2.8.2 "Beschreibung von Farben" des R-BASIC Programmierhandbuchs.

csDrawMask, csDrawMaskValue

Der ColorSelector bietet unter der Bezeichnung "Raster" eine Auswahl von Füllmustern an, die als "Transparenzgrad" angesehen werden können. Die numerischen Werte dieser Füllmuster liegen im Bereich von 25 (vollständig deckend, Raster 100%) bis 89 (vollständig transparent, Raster 0%). Das sind 64 Stufen für den Transparenzgrad. Die Instancevariable csDrawMask enthält diese numerischen Werte. Sie kann gelesen und geschrieben werden.
Die Instancevariable csDrawMaskValue enthält die aus csDrawMask berechneten Prozentwerte (0 bis 100). Dabei wird intern die folgende Formel benutzt:

csDrawMaskValue = (89 - csDrawMask) * 100 / 64

Beim Schreiben von csDrawMaskValue wird die entsprechende Umkehrformel verwendet. Beachten Sie, dass es durch die Stufung in 64 Schritte zu leichten Abweichungen der Werte kommen kann.

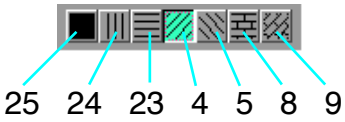
Syntax Lesen: **<numVar>** = **<obj>.csDrawMask**
Schreiben: **<obj>.csDrawMask** = **raster**
UI-Code: **csDrawMask** = **raster**
raster: Numerischer Wert des Rasters, Werte 25 bis 89

Syntax Lesen: **<numVar>** = **<obj>.csDrawMaskValue**
Schreiben: **<obj>.csDrawMaskValue** = **raster**
UI-Code: **csDrawMaskValue** = **raster**
raster: Numerischer Wert des Rasters, in Prozent (0 bis 100)

Die Werte für csDrawMask können direkt mit den Feldern lineDrawMask, areaDrawMask und textDrawMask der Systemvariablen "graphic" verwendet werden um das Füllmuster von Linien, Flächen und Texten einzustellen. Alternativ können Sie den Wert von csFillPattern oder jeden anderen Wert von 0 bis 255 für die Felder lineDrawMask, areaDrawMask und textDrawMask der Systemvariablen "graphic" benutzen. Mehr zum Thema Füllmuster finden Sie im Programmierhandbuch, Kapitel 2.8.4 (Die Systemvariable "graphic": Mixmodes und mehr) sowie im Anhang, Abschnitt C. Dort sind weitere Beispiele für Füllmuster angegeben.

csFillPattern

Der ColorSelector bietet eine kleine Auswahl von grafischen Füllmustern an. Die Instancevariable csFillPattern enthält diese Auswahl. Im Bild sind die zugehörigen numerischen Werte dargestellt.



Syntax	Lesen:	**<numVar>** = **<obj>.csFillPattern**
	Schreiben:	**<obj>.csFillPattern** = **muster**
	UI-Code:	**csFillPattern** = **muster**
	muster:	Numerischer Wert des Füllmusters

Diese Füllmuster können mit den Feldern lineDrawMask, areaDrawMask und textDrawMask der Systemvariablen "graphic" verwendet werden um das Füllmuster von Linien, Flächen und Texten einzustellen. Alternativ können Sie den Wert von csDrawMask oder jeden anderen Wert von 0 bis 255 für die Felder lineDrawMask, areaDrawMask und textDrawMask der Systemvariablen "graphic" benutzen. Mehr zum Thema Füllmuster finden Sie im R-BASIC Programmierhandbuch, Kapitel 2.8.4 (Die Systemvariable "graphic": Mixmodes und mehr) sowie im Anhang, Abschnitt C. Dort sind weitere Beispiele für Füllmuster angegeben.

ColorChangedHandler

Die Instancevariable ColorChangedHandler enthält den Namen des Action-Handlers der aufgerufen werden soll, wenn der Nutzer eine der vom ColorSelector angebotenen Eigenschaften (also auch Raster und Füllmuster) ändert.

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

ColorChangedHandler müssen als ColorAction deklariert sein. Diesem Handler werden die folgenden Parameter übergeben:

sender:	Der ColorSelector, der den Handler aufgerufen hat.
csColor:	Der aktuell ausgewählte Farbwert. Das kann eine RGB-Farbe oder eine Indexfarbe sein. Selbst wenn eine Indexfarbe gewählt ist kann csColor einen (natürlich den zugehörigen) RGB-Farbwert enthalten.
csDrawMask:	Der aktuell ausgewählte Wert für das Füllraster. Mögliche Werte liegen im Bereich von 25 (Raster = 100%) bis 89 (Raster = 0%).
csFillPattern:	Der aktuell ausgewählte Wert für das Füllmuster. Mögliche Werte sind 25, 24, 23, 4, 5, 8 und 9.

Beispiel: Der Actionhandler soll eine farbige Figur auf dem Bildschirm zeichnen. Farbe und Füllmuster sollen durch den ColorSelector, der den ActionHandler gerufen hat, bestimmt werden. Da in R-BASIC Raster und Füllmuster nicht gleichzeitig angewendet werden können wird der Raster-Wert dann verwendet, wenn das Füllmuster auf "vollständig ausgefüllt" gestellt ist. Dem entspricht die in R-BASIC definierte Konstante DM_100 mit dem Zahlenwert 25.

```
ColorAction ColorChangedNotification
DIM mask

  IF csFillPattern = DM_100 THEN
    mask = csDrawMask
  ELSE
    mask = csFillPattern
  END IF

  ' Belegen der Systemvariablen graphic
  graphic.areaDrawMask = mask
  graphic.areaColor = csColor

  FillEllipse 10, 10, 300, 150

END ACTION
```

Der dazugehörige ColorSelector ist folgendermaßen definiert:

```
ColorSelector DemoColorSel
  caption$ = "Farbauswahl"
  justifyCaption = J_CENTER
  DrawInBox
  csFeatures = CSF_INDEX + CSF_MORE_COLORS + CSF_RGB + \
                 CSF_FILL_PATTERN + CSF_DRAW_MASK
  csDrawMask = 58
  csFillPattern = 8
  ColorChangedHandler = ColorChangedNotification
End OBJECT
```

Dieser ColorSelector ruft den Handler ColorChangedNotification jedes Mal, wenn der Nutzer die Farbe, das Füllmuster oder das Raster ändert. Ändert er alle drei Eigenschaften wird der Handler auch drei Mal gerufen. Um das zu umgehen definieren Sie am einfachsten keinen ColorChangedHandler und fragen die Werte für csColor, csDrawMask und csFillPattern manuell ab. Das kann z.B. in einem Button-Handler (Button "Anwenden") geschehen.

Die hier beschriebenen Codefragmente finden Sie komplett als Beispiel unter "R-BASIC\Beispiel\Objekte\Grafik\ColorSelector Demo".

Erfahrene Programmierer finden vielleicht die folgenden Informationen hilfreich:

Wenn Sie im UI Code des ColorSelectors den Hint MakeDelayedApply setzen oder den ColorSelector in einem Dialog verwenden, der den Dialogtyp dialogType = DT_DELAYED_APPLY hat, arbeitet der ColorSelector im sogenannten "Delayed Mode", der im Kapitel 3.4.2 beschrieben ist. In diesem Modus ruft der ColorSelector seinen ColorChangedHandler erst auf, wenn er durch Aufruf der Methode "Apply" dazu aufgefordert wird.
Innerhalb eines Dialogs mit dialogType = DT_DELAYED_APPLY wird der benötigte "Anwenden" Button automatisch erzeugt und auch die Methode "Apply" wird automatisch gerufen. Das System nimmt Ihnen also sehr viel Arbeit ab. Ansonsten müssen Sie den Button selbst definieren. Der ActionHandler sieht dann so aus:

```
BUTTONACTION SendApply
  DemoColorSel.Apply
END ACTION
```

Beachten Sie, dass Sie keinen Zugriff auf die Status-Messages des ColorSelectors haben. Diese werden auf Systemebene intern verwendet.

4.13 DocumentGuardian

Viele Programme arbeiten mit Dokumenten, die der Nutzer anlegen, bearbeiten, speichern und weitergeben kann. Objekte der Klasse DocumentGuardian erleichtern Ihnen den Umgang mit solchen Dokumenten, indem Sie allgemeine Informationen, die bei der Arbeit mit Dokumenten anfallen, verwalten. Dazu zählen z.B. der Name und der Pfad zur Dokumentendatei sowie das FileHandle der offenen Datei. Außerdem können Objekte der Klasse DocumentGuardian vorhandenen Dokumente öffnen, neue Dokumente anlegen und offene Dokumente schließen. Dabei berücksichtigen sie z.B. den Dateityp und das Token, behandeln schreibgeschützte Dateien korrekt und vieles mehr. Auf diese Weise entlasten diese Objekte den BASIC-Programmierer von einer Vielzahl von Standardaufgaben.

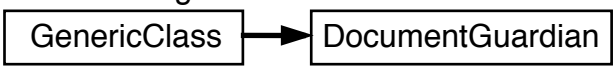
DocumentGuardian-Objekte sind dafür ausgelegt mit der Library "**DocumentTools**" zusammenzuarbeiten. Diese Library bietet z.B. die zur Dateiarbeit nötigen Dialogboxen wie "Öffnen" und "Speichern unter" an. Im Handbuch "Spezielle Themen", Kapitel 15, ("Implementieren eines Dokument-Interfaces") finden Sie eine ausführliche Beschreibung wie man die Arbeit mit Dokumenten unter Verwendung der DocumentTools Library und eines DocumentGuardian-Objekts organisiert. Bitte lesen Sie dort nach, wenn Sie ausführliche Beispiele oder weitergehende Erklärungen benötigen.
Zur Vereinfachung kann der gesamte Dokument-Interface Code über ein R-BASIC Menu in ihr Programm eingebaut werden. Dazu verwenden Sie das Menü "Extras" -> "Code Bausteine" -> "Dokument-Interface". Dort können Sie auch festlegen, welche Teile des Dokumentinterfaces Sie unterstützen wollen, z.B. ob Sie Musterdokumente unterstützen wollen oder nicht.

Im Prinzip läuft die Arbeit mit DocumentGuardian-Objekten so ab:

- Vereinbaren Sie ein DocumentGuardian-Objekt im UI-Code oder erzeugen Sie sein oder mehrere DocumentGuardian-Objekte zur Laufzeit (mit CreateObject).
- Initialisieren Sie das Objekt durch Belegen der Instancevariablen **configData** und **ButtonHandler**.
- Öffnen Sie ein vorhandenes Dokument mit der Methode **OpenDocument** oder erzeugen Sie ein neues Dokument mit **CreateNewDocument**.
- Um Daten aus der Dokument-Datei zu lesen oder in die Datei zu schreiben benötigen Sie das FILE Handle der Datei. Das bekommen Sie über die Instancevariable **documentHandle**.
  Wichtig: Das FILE Handle der Dokumentendatei kann sich im Laufe der Zeit ändern, z.B. wenn die Datei umbenannt (Menüpunkte "Speichern unter" oder "Umbenennen") oder verschoben wird (Menüpunkt "Verschieben nach"). Deswegen sollten Sie das FILE Handle nicht irgendwo zwischenspeichern.
- Wenn Sie Informationen über den Zustand der Datei wissen oder ändern wollen, können Sie die Instancevariable **documentState** und die Methode **SetDocumentState** verwenden. Insbesondere sollten Sie SetDocumentState rufen, wenn Sie die Datei als geändert markieren wollen.

- Weitere wichtige Informationen über die offene Datei speichern die Instance-variablen **documentName$** und **documentPath$**. **DocumentUserData** können Sie verwenden, um beliebige weitere Informationen über die Datei abzulegen.
- Um ein Dokument zu schließen verwenden Sie die Methode **CloseDocument**.
- Wenn GEOS herunterfährt bzw. wieder neu startet übernehmen die Methoden **HandleShutdown** und **HandleRestart** alle notwendigen Schritte um die Datei zu schließen und beim Neustart automatisch wieder zu öffnen.

Abstammung:



Objekte der Klasse DocumentGuardian sind per Default nicht sichtbar (visible = FALSE). Sie können in den generic Tree eingebunden werden. Werden sie auf visible = TRUE gesetzt verhalten sie sich wie eine Group. Die unterstützen das Geometriemanagement und können Children haben.

Spezielle Instance-Variablen:

Variable	Syntax im UI-Code	Im BASIC-Code
ButtonHandler	ButtonHandler = **<Handler>**	nur schreiben
configData	—	lesen, schreiben
documentState	—	nur lesen
documentHandle	—	nur lesen
documentName$	—	nur lesen
documentPath$	—	nur lesen
documentUserData	—	lesen, schreiben

Methoden:

Methode	Aufgabe
SetDocumentState	Informationen über das Dokument ändern
CreateNewDocument	Neues Dokument anlegen
OpenDocument	Vorhandenes Dokument öffnen
CloseDocument	Aktuelles Dokument schließen
HandleShutdown	System-Shutdown behandeln
HandleRestart	System Neustart behandeln

4.13.1 Konfiguration

ConfigData

Damit das DocumentGuardian seine Aufgaben, z.B. eine Datei anzulegen oder zu öffnen, erfüllen kann, benötigt es diverse Informationen. Dazu gehören der Dateityp (DOS-, GEOS-Daten- oder VM-Datei), für GEOS-Daten- und VM-Dateien das Token und einiges mehr. Außerdem arbeitet das DocumentGuardian-Objekt eng mit der "DocumentTools"-Library zusammen. Diese Library entlastet den Programmierer, indem Sie zum Beispiel die typischen Dialogboxen zum Öffnen oder "Speichern unter" von Dateien bereitstellt. Die für die Arbeit dieser Library nötigen Informationen werden ebenfalls vom DocumentGuardian-Objekt bereitgestellt.

Die Instancevariable **configData** enthält eine Struktur, die alle für die oben genannten Zwecke erforderlichen Informationen enthält. Die Struktur ist folgendermaßen definiert:

```
STRUCT DocumentConfigStruct
  noDocumentString$      As String(32)
  templateFolder$        As String(32)
  nameForNew$            As String(32)
  fileType               As word
  token                  As GeodeToken
  creatorToken           As GeodeToken
  matchMask$             As String(32)
  matchFlags             As Word
  reserved(4)            As Word
End Struct
```

Bedeutung der einzelnen Felder:

noDocumentString$
NoDocumentString$ enthält den Text, den die Methode DocumentName$ zurückgeben soll, wenn kein Dokument geöffnet ist. Ein typischer Wert wäre "kein Dokument".

templateFolder$
TemplateFolder$ enthält den Unterordner im SP_TEMPLATE-Verzeichnis, in dem das Programm Templates ("Muster-Dateien") speichert. Das Document-Guardian-Objekt stellt sicher, dass der Ordner existiert.

nameForNew$
NameForNew$ enthält den Kerntext, aus dem das DocumentGuardian-Objekt durch Hinzufügen einer Zahl von 1 bis 99 einen Namen für eine neue Datei bilden soll.
NameForNew$ hat das Format "core*.ext" oder "core". Beispiele:
Aus "LEER*.RBF" wird "LEER1.RBF", "LEER2.RBF" bis "LEER99.RBF"
Aus "NEU*" wird "NEU2", "NEU2" bis "NEU99"
Aus "namenlos " wird "namenlos 1", "namenlos 2" bis "namenlos 99"

Wichtig: Sie sind selbst dafür zuständig, dass der nach dem oben genannten Schema gebildete Name zum Dateityp passt, für DOS-Dateien also das Format 8+3 eingehalten wird.

fileType

FileType enthält den Typ der Dokumentendatei. Gültige Werte sind GFT_VM, GFT_DATA und GFT_NOT_GEOS_FILE. Erfahrene Programmierer sollten VM-Dateien (GFT_VM) verwenden, für alle anderen empfiehlt sich GFT_DATA (GEOS-Daten-Dateien). Diese unterstützen wie die VM-Dateien die Geos-Attribute wie Token und CreatorToken. DOS-Dateien (GFT_NOT_GEOS_FILE) sollten Sie nur dann verwenden, wenn Sie einen wirklich guten Grund dafür haben.

token, creatorToken

Token und CreatorToken enthalten das Token und das CreatorToken der Datei für GEOS-Daten- und VM-Dateien. Beim Anlegen einer neuen Datei setzt das DocumentGuardian-Objekt Token und CreatorToken automatisch.

matchMask$, matchFlags

MatchMask$ und matchFlags werden vom DocumentGuardian-Objekt nicht selbst verwendet, sondern sind zur Arbeit der DocumentTools-Library notwendig. MatchFlags enthält die Information, welche Eigenschaften der Datei der im "Öffnen"-Dialog angezeigte FileSelector zum Filtern der Dateien verwenden soll. MatchMask$ enthält die Dateimaske, die verwendet wird, falls das entsprechende Bit in MatchFlags gesetzt ist.

reserved

Diese 10 Bytes sind für eventuelle spätere Erweiterungen vorgesehen. Sie dürfen diese Werte nicht verwenden.

Zur Verwendung mit matchFlags sind die folgenden Konstanten definiert:

Konstante	Wert	Bedeutung
DOC_MATCH_TOKEN	1	Feld "token" verwenden
DOC_MATCH_CREATOR	2	Feld "creatorToken" verwenden
DOC_MATCH_MASK	4	Feld "matchMask$" verwenden
DOC_MATCH_TYPE	8	Feld "fileType" verwenden

Für DOS-Dateien wird oft die Kombination "DOC_MATCH_TYPE + DOC_-MATCH_MASK" verwendet, währen für GEOS-Daten- und VM-Dateien oft "DOC_MATCH_TOKEN" oder "DOC_MATCH_TOKEN + DOC_MATCH_-CREATOR" verwendet wird.

Beispiel:

```
SUB DoInitDocumentGuardian(guardian as object)
DIM dc as DocumentConfigStruct

  ' ButtonHandler kann hier oder im UI-Code gesetzt werden
  guardian.ButtonHandler = DocumentAndToolButtonHandler

  dc.noDocumentString$ = "kein Dokument"
  dc.templateFolder$ = "DemoTemplates"
  dc.nameForNew$ = "Unbenanntes Dokument "

  dc.fileType = GFT_DATA
  dc.creatorToken.tokenChars = "PHON"
  dc.creatorToken.manufid = 5
  dc.token.manufid = 5
  dc.token.tokenChars = "PHO2"

  'dc.matchMask$ = "*.TXX" # nicht benötigt für GEOS-Dateien
  dc.matchFlags = DOC_MATCH_TOKEN

  guardian.configData = dc
END SUB
```

ButtonHandler

Die Instancevariable ButtonHandler enthält einen Handler, der von der DocumentTools-Library benötigt wird. Er muss als ButtonAction deklariert sein und wird gerufen, wenn der Nutzer einen Button im "Neu/Öffnen" Dialog anklickt. Dieser Dialog wird von der Routine **DTNewOpenDialog** aus der DocumentTools Library erzeugt. Weitere Informationen dazu finden Sie im Handbuch der Library.

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

Beispiel:

```
DocumentGuardian MyDocObj
  ButtonHandler = DocumentAndToolButtonHandler
End Object
```

4.13.2 Verwalten von Dokumenten

CreateNewDocument

Mit der Methode CreateNewDocument legt das DocumentGuardian Objekt ein neues Dokument entsprechend den in der Instancevariablen configData gespeicherten Werten an.

- Der Name der neuen Datei wird aus configData.nameForNew und einer Zahl gebildet.
- Der Dateityp (DOS-, GEOS-Daten- oder GEOS-VM-Datei) wird von configData.fileType bestimmt.
- Für GEOS-Daten- und VM-Dateien wird Token (configData.token) und CreatorToken (configData.creatorToken) gesetzt.
- VM-Dateien werden so initialisiert, dass sie mit der "VMFiles" Library verwendet werden können. Details zur Arbeit mit VM-Dateien finden Sie Im Handbuch der "VMFiles" Library
- **Achtung!** Sollte das DocumentGuardian-Objekt noch eine Dokument-Datei offen haben wird sie ohne Nachfragen geschlossen.

Die neue Dokument-Datei wird im aktuellen Ordner angelegt. Die globale Variable fileError wird gesetzt (Null oder Fehlercode).

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

Beispiel:

```
MyDocObj.CreateNewDocument
MsgBox "Die neue Datei hat den Namen: " + MyDocObj.documentName$
```

OpenDocument

Mit der Methode OpenDocument öffnet das DocumentGuardian Objekt ein vorhandenes Dokument. Sollte das DocumentGuardian-Objekt noch eine Dokument-Datei offen haben wird sie ohne Nachfragen geschlossen.
Die globale Variable fileError wird gesetzt (Null oder Fehlercode).

Syntax BASIC-Code: **<obj>.OpenDocument "name"** [, **forceRO**]
"name": Dateiname der zu öffnenden Datei
forceRO: FALSE: normal öffnen (Default)
TRUE: schreibgeschützt öffnen

Beispiel:

```
MyDocObj.OpenDocument "MyDoc"
```

CloseDocument

Mit der Methode CloseDocument schließt das DocumentGuardian Objekt das aktuell von ihm geöffnete Dokument. Ist das Dokument noch unbenannt (das Bit DOCS_UNTITLED ist gesetzt in der Instancevariablen documentState) so wird die Datei nach dem Schließen gelöscht. Für den seltenen Fall, dass Sie nicht möchten, dass unbenannte Dokumente nach dem Schließen gelöscht werden, müssen Sie FALSE als Parameter angeben. Die globale Variable fileError wird in beiden Fällen gesetzt (Null oder Fehlercode).

Syntax BASIC-Code: **<obj>.CloseDocument**
<obj>.CloseDocument FALSE

Beispiele:

```
MyDocObj.CloseDocument
MyDocObj.CloseDocument FALSE
```

DocumentHandle

Diese Instancevariable enthält das File-Handle der aktuell vom Document-Guardian geöffneten Datei. Ist kein Dokument geöffnet enthält documentHandle ein NullHandle.

Syntax Lesen: **<fh> = <obj>.documentHandle**
<fh>: Variable vom Typ FILE

Beispiel:

```
DIM fh as FILE
DIM text$
  fh = MyDocObj.documentHandle
  IF fh <> NullFile() THEN
    text$ = FileReadLine$ ( fh )
  End IF
```

DocumentName$

DocumentName$ enthält den Namen der aktuell vom DocumentGuardian geöffneten Datei. Ist kein Dokument geöffnet enthält documentName$ den Text, der im Feld "noDocumentString" der Instancevariablen configData gespeichert ist.

Syntax Lesen: **<stringVar> = <obj>.documentName$**
<stringVar>: Variable vom Typ String

Beispiel:

```
MyPrimary.Caption2$ = MyDocObj.documentName$
```

DocumentPath$

DocumentPath$ enthält den Pfad zur aktuell vom DocumentGuardian geöffneten Datei. Ist kein Dokument geöffnet documentPath$ den Pfad zur zuletzt geöffneten Datei.

Syntax Lesen: **<stringVar>** = **<obj>**.**documentPath$**
<stringVar>: String-Variable, die 200 Zeichen aufnehmen kann.

Normale Stringvariablen können bis zu 128 Zeichen aufnehmen. Das ist in den meisten Fällen ausreichend, kann jedoch zu wenig sein, weil Pfade bis zu 198 Zeichen lang sein können.

Beispiel:

```
DIM path$(200)
  path$ = MyDocObj.documentPath$
  MsgBox "Das Dokument liegt im Ordner '" + path$ + "'."
```

documentState, SetDocumentState

Die Instancevariable documentState enthält Informationen (Flagbits) über den aktuellen Zustand des vom DocumentGuardian-Objekt verwalteten Dokuments. DocumentState enthält den Wert Null wenn kein Dokument offen ist. Folgende Bits sind definiert:

Konstante	Wert	Bedeutung
DOCS_OPEN	1	Es ist ein Dokument offen
DOCS_MODIFIED	2	Das Dokument wurde geändert. (*)
DOCS_UNTITLED	4	Das Dokument ist neu und hat noch keinen vom Nutzer vergebenen Namen.
DOCS_READ_ONLY	8	Das Dokument ist schreibgeschützt
DOCS_EDIT_TEMPLATE	16	Es ist ein Muster in Bearbeitung (**)

(*) Das Bit DOCS_MODIFIED wird vom DocumentGuardian-Objekt weder automatisch gesetzt noch zurückgesetzt. Es wird von der "DocumentTools" Library benötigt. Deswegen **muss** der Programmierer es auf dem aktuellen Stand halten.

(**) Das Bit DOCS_EDIT_TEMPLATE wird vom DocumentGuardian-Objekt nicht automatisch gesetzt. Es wird benutzt um sicherzustellen, dass der nächste "Öffnen"-Dialog den Dokument-Ordner anzeigt, und nicht dem der aktuell offenen Datei (den Template-Ordner). Der Programmierer **sollte** es setzen, wenn er ein Muster zum Bearbeiten öffnet. Es wird beim Schließen dieser Datei automatisch zurückgesetzt.

Die Instancevariable documentState kann nur gelesen werden. Um sie zu ändern verwenden Sie die Methode SetDocumentState. Sie erwartet zwei Parameter: die Bits, die zu setzen sind und die Bits, die zu löschen sind. Das vereinfacht die Verwaltung des Dokumentstatus wesentlich. SetDocumentState kann nur die Bits DOCS_MODIFIED, DOCS_UNTITLED und DOCS_EDIT_TEMPLATE ändern. Wenn kein Dokument offen ist können diese Bits nicht gesetzt werden.

Syntax BASIC-Code: **<numVar>** = **<obj>**.**documentState**

Syntax BASIC-Code: **<obj>**.**SetDocumentState** **bitsToSet**, **bitsToClear**

Beispiele:

```
DIM state
  state = MyDocumentGuardian.documentState
  IF state = 0 THEN MsgBox "Es ist kein Dokument offen."
  IF state AND DOCS_UNTITLED THEN MsgBox "Das Dokument ist neu."
```

```
  ' Dokumentstatus auf "geändert" setzen
  MyDocumentGuardian.SetDocumentState DOCS_MODIFIED, 0

  ' Dokumentstatus auf "ungeändert" setzen
  MyDocumentGuardian.SetDocumentState 0, DOCS_MODIFIED
```

4.13.3 Services

HandleShutdown, HandleRestart

Wenn GEOS herunterfährt muss das DocumentGuardian-Objekt das aktuelle Dokument schließen und beim Wiederhochfahren muss das gleiche Dokument wieder geöffnet werden. Darum kümmern sich die Methoden HandleShutdown und HandleRestart. Diese Methoden sollten im OnExit bzw. im OnStartup-Handler des Programms gerufen werden. Im Handbuch "Spezielle Themen", Kapitel 15, ("Implementieren eines Dokument-Interfaces") finden Sie ein ausführlicheres und kommentiertes Beispiel dazu.

Syntax BASIC-Code: **<obj>.HandleShutdown**
<obj>.HandleRestart

Beispiel:

```
SYSTEMACTION DocExitHandler
   IF flags AND AF_SHUTDOWN THEN
        DocumentObj.HandleShutdown
   Else
        ! Hier Dokument ggf. Speichern und normal schließen
   End IF
END ACTION

SYSTEMACTION DocStartupHandler
  IF flags AND AF_RESTORE THEN
        DocumentObj.HandleRestart
  End IF
END ACTION
```

documentUserData

Manchmal ist es wünschenswert, Dokumentdaten an einem sicheren Platz abzuspeichern, ohne sie in die Dokumentendatei zu schreiben. Zum Beispiel "überleben" globale Variablen eine Systemrestart nicht. Die Instancevariable documentUserData kann eine einzelne Strukturvariable (d.h. bis zu 3500 Byte) aufnehmen. Wie bei jeder anderen Instancevariablen von R-BASIC Objekten stehen die Daten nach einem Systemrestart automatisch wieder zur Verfügung.
Eine andere Anwendung wäre, wenn Sie ein Programm schreiben, dass mit mehreren offenen Dokumenten gleichzeitig umgehen kann, wobei jedes offene Dokument sein eigenes DocumentGuardian-Objekt hat. DocumentUserData könnte dann die Dokument-spezifischen Daten enthalten.

DocumentUserData ist zuweisungskompatibel mit jeder Art von Strukturvariablen. Beim Schreiben müssen Sie allerdings die Größe der Struktur extra angeben. Beim Lesen müssen Sie selbst darauf achten, dass die Strukturen kompatibel sind, R-BASIC führt weder eine Typ- noch eine Größenprüfung aus. Aber es ist

sichergestellt, dass niemals mehr Bytes kopiert werden, als die Variable auf der linken Seite der Zuweisung aufnehmen kann.

Es ist zulässig mehrfach hintereinander Strukturen verschiedenen Typs und verschiedener Größe in die Instancevariable zu schreiben. R-BASIC optimiert jedes Mal den verwendeten Speicher, so dass kein Platz verschwendet wird.

Syntax Schreiben: **<obj>.documentUserData** = **<struct>**, **size**
<struct>: Strukturausdruck beliebigen Typs
size Größe der Struktur
Lesen: **<sturctVar>** = **<obj>.documentUserData**
<structVar>: Strukturvariable des Typs, der beim Schreiben verwendet wurde.

Beispiel:
Wir definieren eine Struktur, die Referenzen auf zwei Objekte enthält (ein View-Objekt und ein Display-Objekt), die zur Darstellung der Dokumentdaten Verwendung finden könnten. Dann definieren wir eine Strukturvariable, belegen diese und speichern die Informationen im DocumentGuardian Objekt.

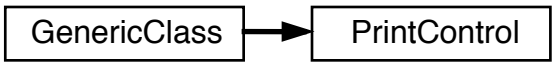
```
STRUCT MyObjects
  docView as OBJECT
  docDiaplay as OBJECT
End Struct
```

```
DIM ob, ob2 as MyObjects

  ob.docView = MyView
  ob.docDisplay = MyDisplay

  guardian.documentUserData = ob, sizeof(ob)
....
  ob2 = guardian.documentUserData
....
```

4.14 PrintControl

Wenn Sie aus einem R-BASIC-Programm heraus drucken wollen müssen Sie ein PrintControl-Objekt benutzen. Das PrintControl-Objekt stellt den "Drucken"-Dialog bereit und steuert alle für das Drucken notwendigen Vorgänge. Wenn der Nutzer die Möglichkeit haben soll, die Seitengröße des Dokuments einzustellen, sollten Sie zusätzlich ein PageSizeControl Objekt verwenden, dass alle für die Einstellung der Seitengröße notwendige UI bereitstellt.

Abstammung:



Objekte der Klasse PrintControl selbst sind unsichtbar. Sie müssen aber in den generic Tree eingebunden werden, damit der "Drucken"-Dialog angezeigt werden kann.

Spezielle Instance-Variablen:

Variable	Syntax im UI-Code	Im BASIC-Code
pcAttrs	pcAttrs = **attrWert**	lesen, schreiben
pcAppUI	pcAppUI = **<Obj>**	—
totalPageRange	totalPageRange = **start**, **end**	lesen, schreiben
userPageRange	userPageRange = **start**, **end**	lesen, schreiben
pcDocSize	pcDocSize = **width**, **height**	lesen, schreiben
pcDocMargins	pcDocMargins = **left**, **top**, **right**, **bottom**	lesen, schreiben
pcLayout	pcLayout = **layoutWert**	lesen, schreiben
pcPaperSizeInfo	—	nur Lesen
printJobName$	printJobName$ = **"Name"**	nur Schreiben
OnPrint	OnPrint = **<Handler>**	lesen, schreiben
OnVerifyPrint	OnVerifyPrint = **<Handler>**	lesen, schreiben
printMode	—	nur Lesen

Methoden:

Methode	Aufgabe
InitiatePrint	"Drucken" Dialogbox anzeigen und drucken
PrintingVerified	Überprüfen der Nutzereingaben beendet
ReportProgress	Druckfortschritt für Nutzer anzeigen
ReportProgressText	Druckfortschritt für Nutzer anzeigen
PrintNewPage	Neue Seite beginnen
PrintingCompleted	Grafik- und Textausgabe fertig Daten an Drucker senden
PrintingCancelled	Drucken abbrechen

Action-Handler-Typen:

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

Spezielle Routinen:

Routine	Deklaration
PrintObj	SUB PrintObj (obj as object, x, y as word)

4.14.1 Überblick über das Drucken

Um unter R-BASIC zu drucken benötigen Sie die folgenden Objekte
- ein "Drucken" Button, der sich üblicher Weise im Datei-Menü befindet
- ein PrintControl-Objekt
- gegebenenfalls ein PageSizeControl Objekt.

Das PrintControl-Objekt arbeitet intern mit der "Spool"-Library zusammen. Diese Library sorgt für das Drucken im Hintergrund und arbeitet mit den Druckertreibern zusammen.

Im Einzelnen passiert folgendes:

1. Der Nutzer aktiviert den "Drucken" Button.
   Daraufhin wird der ActionHandler dieses Buttons aufgerufen.
2. Im ActionHandler des Drucken-Buttons müssen Sie das PrintControl-Objekt initialisieren, soweit das noch nicht im UI-Code passiert ist.
   Sie müssen an dieser Stelle sicherstellen, dass folgende Instancevariablen korrekt belegt sind:
   - pcDocSize, pcDocMargins, pcLayout (Größe und Layout des Dokuments)
     Ausnahme: Wenn Sie die Größe des Papiers im Drucker als Dokumentgröße verwenden wollen, sollten Sie diese Instancevariablen erst im OnPrintHandler setzen.
   - bei mehrseitigen Druckaufträgen: totalPageRange, userPageRange (zu druckender Seitenbereich)
   - printJobName$ (Name des Printjobs)

   Die letzte Anweisung dieses Handlers muss der Aufruf der InitiatePrint Methode des PrintControl-Objekts sein.
3. Das PrintControl-Objekt präsentiert den "Drucken" Dialog. Der Nutzer hat hier die Möglichkeit den Drucker, die Papiersorte und vieles mehr auszuwählen. Mit dem aktivieren des "Drucken" Buttons in diesem Dialog beginnt der eigentliche Druckvorgang.
4. Wenn das PrintControl-Objekt einen OnVerifyPrint-Handler hat wird dieser jetzt gerufen. Das Programm kann hier prüfen, ob die Eingaben des Nutzers im Drucken-Dialog konsistent sind und ihn gegebenenfalls zur Korrektur seiner Eingaben auffordern.
   Besitzt das PrintControl-Objekt keinen OnVerifyPrint-Handler - was sehr häufig der Fall ist - wird dieser Schritt übersprungen.
5. Die Spool-Library erzeugt einen GString, in dem alle Druckausgaben zwischengespeichert werden, und übergibt ihn an das PrintControl-Objekt.
6. Das PrintControl-Objekt ruft seinen OnPrint Handler. Dabei setzt es den GString von der Spool-Library als Screen. Alle Grafik- und Textausgaben gehen jetzt in dieses GString.
7. Der OnPrint-Handler muss die folgenden Schritte abarbeiten:
   a) Falls noch nicht im Actionhandler des Printbuttons geschehen: Belegen der Instancevariablen pcDocSize, pcDocMargins und pcLayout des PrintControl-Objekts
   b) Sehr häufig:
      • Setzen des Fonts für den Ausdruck (Anweisung: FontSetGeos). Der Standardfont sieht beim Ausdruck nicht so toll aus.

- Setzen des Koordinatenursprungs so, dass der linke und obere Rand standardmäßig nicht bedruckt wird (Anweisung: ScreenSetTranslation, Beispiele siehe unten).

c) Ausgabe von Grafik oder Text auf den Drucker. Es sind alle Grafik- und Textbefehle zulässig. Sie können als Programmierer so tun, als ob das zu bedruckende Blatt Papier der Bildschirm ist.
Um den Nutzer über den Druckfortschritt zu informieren können Sie in regelmäßigen Abständen, z.B. am Beginn jeder neuen Seite, eine der Methoden ReportProgress oder ReportProgressText aufrufen.
Um eine neue Seite zu beginnen müssen die die Methode PrintNewPage des PrintControl-Objekts aufrufen.

d) Wenn Sie alles gedruckt haben müssen Sie die Methode Printing-Completed aufrufen. Sie wirft automatisch die letzte gedruckte Seite aus.
Falls Sie den Druckvorgang vorzeitig abbrechen wollen rufen Sie satt PrintingCompleted die Methode PrintingCancelled auf. Der Spooler bricht dann den Druckjob ab.

8. Das PrintControl-Objekt setzt den Screen zurück und informiert den Spooler, dass der Druck beendet ist.
9. Der Spooler kommuniziert mit dem Druckertreiber und prüft, ob die auszudruckende Seite aufs Papier passt. Falls nicht wird der Nutzer gefragt, ob er die Seite herunterskalieren oder auf mehrere Blätter verteilen will.
10. Der Druckertreiber übersetzt die GEOS - Grafik- und Textkommandos in die Sprache des Druckers. Die Seiten werden jetzt endlich gedruckt.

Beispiele, in denen typische Fälle behandelt werden, finden Sie im Ordner R-BASIC\Beispiel\Objekte\Drucken.

4.14.2 Korrektes Setzen der Instancevariablen

Wichtig für das problemlose Drucken ist das korrekte Belegen der Instancevariablen des PrintControl-Objekts. Prinzipiell gibt es drei Stellen, an denen die Instancevariablen belegt werden können: im UI-Code, im Actionhandler des Drucken-Buttons und im OnPrint-Handler des PrintControl-Objekts.

Die folgende Tabelle enthält Empfehlungen, wo die entsprechenden Instancevariablen sinnvoller Weise gesetzt werden sollten. Für einfache Fälle reicht es aus, die Werte fest im UI-Code zu setzen. In den meisten Fällen ist der Actionhandler des Drucken-Buttons der richtige Platz zum Belegen der entsprechenden Instancevariablen. Nur wenn dort die gewünschte Funktionalität nicht erreicht werden kann sollten Sie auf den OnPrint-Handler ausweichen. In diesem Fall wird empfohlen, die Initialisierungen vor der ersten Text- oder Grafikausgabe zu machen.

Spezielle Instance-Variablen:

Variable	Defaultwert belassen	UI-Code	Actionhandler des Drucken Buttons	OnPrint Handler
pcAttrs	X	X		
totalPageRange	X	X	X	
userPageRange	X	X	X	Lesen
pcDocSize		X	X	X
pcDocMargins		X	X	X
pcLayout	X	X	X	X
pcPaperSizeInfo				Lesen
printJobName$	(X)	(X)	X	X
printMode				Lesen

Beachten Sie, dass es nicht zulässig ist, für pcDocSize und pcDocMargins die Defaultwerte zu belassen (da diese jeweils Null sind).

4.14.3 Konfigurieren des PrintControl-Objekts

Die Basiskonfiguration des PrintControl-Objekts findet im UI-Code statt.

pcAttrs

Die Instancevariable pcAttrs beschreibt die Eigenschaften und Fähigkeiten, die durch das PrintControl-Objekt bereitgestellt werden sollen. pcAttrs enthält Bitflags, d.h. jedes Bit hat eine eigene Bedeutung.

Syntax:	UI-Code	**pcAttrs** = **attrWert**
	Schreiben	**<obj>.pcAttrs** = **attrWert**
	Lesen:	**<numVar>** = **<obj>.pcAttrs**
	attrWert:	Numerischer Wert Kombination der PCA_-Konstanten, siehe Tabelle

Für pcAttrs sind folgende Werte definiert. Hier nicht aufgeführte Bits müssen Null bleiben!
Die meisten der in der Tabelle angegeben Bit erfordern keine weiter Unterstützung durch das Programm. Ausnahmen sind:

PCA_VERIFY_PRINT
PCA_PROGRESS_PERCENT / PCA_PROGRESS_PAGE,
PCA_PAGE_CONTROLS

Konstante	Wert	hex.	Default gesetzt
PCA_NO_PRINTER_CONTROLS	32868	&h8000	
PCA_SEE_IF_DOC_WILL_FIT	16384	&h4000	X
PCA_MARK_APP_BUSY	8192	&h2000	
PCA_VERIFY_PRINT	4096	&h1000	
PCA_SHOW_PROGRESS	2048	&h800	X
PCA_PROGRESS_PERCENT	1024	&h400	
PCA_PROGRESS_PAGE	512	&h200	
PCA_FORCE_ROTATION	256	&h100	
PCA_COPY_CONTROLS	128	&h80	X
PCA_PAGE_CONTROLS	64	&h40	X
PCA_QUALITY_CONTROLS	32	&h20	X
PCA_USES_DIALOG_BOX	16	&h10	X
PCA_GRAPHICS_MODE	8	&h8	X
PCA_TEXT_MODE	4	&h4	X
PC_DEFAULT_ATTRS	18684	&h48FC	X

Bedeutung der einzelnen Bits:

PCA_NO_PRINTER_CONTROLS legt fest, dass der Nutzer den Drucker nicht auswählen kann.

PCA_SEE_IF_DOC_WILL_FIT bewirkt, dass der Spooler prüft, ob eine Dokumentseite auf eine Seite im Drucker passt und gegebenenfalls nachfragt, wie weiter verfahren werden soll.

PCA_MARK_APP_BUSY ändert während des Druckens den Mauszeiger in eine Sanduhr. Das ist nützlich für längere Printjobs.

PCA_VERIFY_PRINT bewirkt, dass der OnVerifyPrint-Handler zur Überprüfung der Nutzereingaben im Drucken-Dialog verwendet wird.

PCA_SHOW_PROGRESS bewirkt, dass während des Druckens eine Fortschrittsbox angezeigt wird. Sie können bei Bedarf die Methode ReportProgressText aufrufen um die Anzeige zu aktualisieren.

PCA_PROGRESS_PERCENT, PCA_PROGRESS_PAGE. Diese Bits bewirken, der Fortschritt in der Fortschrittsbox als Prozentwert oder in der Form "Seite (von Seiten)" (oder beides) angezeigt wird. Der genaue Text der Anzeige hängt von der konkreten GEOS-Version ab.
Wenn Sie diese Bits verwenden müssen Sie in regelmäßigen Abständen die Methode ReportProgress aufrufen um die Anzeige zu aktualisieren.

PCA_FORCE_ROTATION bewirkt, dass die Ausgabe auf jeden Fall im Querformat erfolgt. Normalerweise wird dieses Bit nicht gesetzt, so das der Spooler entscheiden kann, wie er zu große Dokumente optimal verteilt. Die "GeoBanner" Applikation nutzt dieses Bit um das Querformat zu erzwingen.

PCA_COPY_CONTROLS legt fest, ob der Nutzer die Anzahl der zu druckenden Kopien einstellen kann.

PCA_PAGE_CONTROLS legt fest, ob der Nutzer den zu druckenden Seitenbereich einstellen kann.

PCA_QUALITY_CONTROLS legt fest ob der Nutzer die Druckqualität einstellen kann oder nicht.

PCA_USES_DIALOG_BOX legt fest, dass das PrintControl Objekt den "Drucken"-Dialog anzeigen soll.

PCA_GRAPHICS_MODE, PCA_TEXT_MODE
Diese beiden Bits legen fest, ob der Controller die Optionen "Text-Modus" und / oder "Grafik-Modus" anbieten soll. Sie sollten diese Bits gesetzt lassen. Bei Druckern, die nicht über die entsprechenden Fähigkeiten verfügen bietet das PrintControl-Objekt die zugehörige Option gar nicht an.

PC_DEFAULT_ATTRS enthält die Summe (logische ODER-Kombination) aller per Default gesetzten Attribute.

Beispiele:

```
' Ein PrintControl-Objekt ohne Auswahl der zu druckenden Seiten
' z.B. weil es immer genau eine Seite gibt wie in GeoDraw
PrintControl DemoPrintControl
  pcAttrs = PC_DEFAULT_ATTRS AND (NOT PCA_PAGE_CONTROLS)
  < ... >
End OBJECT

' Ein PrintControl-Objekt wobei die Fortschrittsbox für die
  Verwendung der Methode ReportProgressText vorbereitet ist
PrintControl DemoPrintControl
  pcAttrs = PC_DEFAULT_ATTRS OR PCA_PROGRESS_PERCENT
  < ... >
End OBJECT
```

pcAppUI

In vielen Fällen ist es erforderlich, dass programmspezifische Einstellungen vor dem Drucken vorgenommen werden müssen. Zum Beispiel kann eine Textverarbeitung eine Serienbrief-Option anbieten oder eine Tabellenkalkulation bietet die Möglichkeit an, Zeilen- und Spaltennamen zu drucken. Für diesen Zweck kann man über die Instancevariable pcAppUI ein UI-Objekt angeben, dass dann in die Drucken-Dialogbox eingebaut wird. Sehr häufig wird hier eine Group verwendet, die dann die weiteren UI-Objekte enthält.
Das Objekt darf noch nicht an anderer Stelle in den generic Tree eingebunden sein. Und es sollte auf visible = FALSE gesetzt sein.
Obwohl das Objekt noch nicht im generic Tree ist können Sie mit ihm und seinen Children ohne Einschränkung kommunizieren. Sie dürfen z.B. Instancevariablen verändern (etwa die gesetzten Optionen einer OptionGroup verändern) und auch Objekt-Methoden aufrufen.

Syntax: UI-Code	**pcAppUI** = **<obj>**
<obj>	Objekt, das zur Drucken-Dialogbox hinzugefügt werden soll. Das Objekt darf nicht in den generic Tree eingebunden sein.

Beispiel:

```
PrintControl DemoPrintControl
  pcAttrs = PC_DEFAULT_ATTRS AND \
        (NOT (PCA_PAGE_CONTROLS or PCA_QUALITY_CONTROLS))
  pcAppUI = ExtraPrintGroup
  < ... >
End OBJECT
Group ExtraPrintGroup
  Caption$ = "Sonstige Optionen"
  Children = ExtraPrintOptionGroup
  visible = FALSE
  DrawInBox : ExpandWidth
  End Object
OptionGroup ExtraPrintOptionGroup
  Children =  ExtraPrintOption
  selection = 0
End OBJECT
Option ExtraPrintOption
  Caption$ = "Hintergrund drucken"
  identifier = 1
End OBJECT
```

Die Abfrage der Option kann im OnPrint Handler folgendermaßen erfolgen:

```
  ' Hintergrund drucken?
  IF ExtraPrintOptionGroup.selection AND 1 THEN
    < ... >
    END IF
```

4.14.4 Vorbereiten für den Druck

4.14.4.1 Dokumentgröße und Dokumenträndер

GEOS unterscheidet grundsätzlich zwischen dem Format des Papiers im Drucker und dem Format (Seitengröße) des zu druckenden Dokuments. Stimmen die Formate nicht überein so kümmert sich Spooler - gegebenenfalls mit Nachfrage beim Nutzer - darum, die Seiten vernünftig aufs Papier zu bringen. Typische Fälle, zum Beispiel, dass ein Dokument im Querformat erstellt wurde und deshalb gedreht werden muss, handelt der Spooler selbständig und intelligent.

Damit dieses Zusammenspiel reibungslos klappt müssen Sie dem PrintControl-Objekt die Größe und die Ränder des auszudruckenden Dokuments mitteilen. Dazu dienen die Instancevariablen **pcDocSize** und **pcDocMargins**.
Über die Instancevariable **pcLayout** teilen Sie dem PrintControl-Objekt z.B. mit, ob Ihr Dokument im Hoch- oder Querformat vorliegt.

Bei Anwendungen mit fester Dokumentgröße können Sie Größe und Ränder im UI-Code setzen. Ansonsten ist der übliche Platz der ActionHandler des "Drucken" Buttons. Nur wenn Sie die Dokumentgröße an die Größe des Papiers im Drucker anpassen wollen setzen Sie diese Werte im OnPrint Handler.

pcDocSize

Die Instancevariable pcDocSize enthält die Größe des Dokuments, einschließlich der Ränder. Die Einheit ist ein typografischer Point (= 1/72 Zoll bzw. 0,35378 cm).
Der größte erlaubte Wert für Breite und Höhe sind jeweils 32767 Point. Das entspricht etwa 1,15 m.
Die Default-Werte für pcDocSize sind Null, pcDocSize muss gesetzt werden, bevor ein Dokument gedruckt werden kann. Das kann im UI-Code oder im Actionhandler des Drucken-Buttons erfolgen.

Syntax:	UI-Code	**pcDocSize** = **width**, **heigth**
	Schreiben	**<obj>.pcDocSize** = **width**, **heigth**
	Lesen:	**<numVar>** = **<obj>.pcDocSize** (**n**)
	width:	Breite des Dokuments, in Point
	height:	Höhe des Dokuments, in Point
	n:	Information, welcher Wert gelesen werden soll, siehe Tabelle (PS steht für Page-Size)

Tabelle: Informationen, die von pcDocSize gelesen werden sollen

Konstante	Wert	Information
PS_WIDTH	4	Breite des Dokuments
PS_HEIGHT	5	Höhe des Dokuments

Wichtig: Wenn Ihr Dokument breiter ist als hoch sollten Sie auch pcLayout auf PL_PAPER_LANDSCAPE setzen.

pcDocMargins

Die Instancevariable pcDocMargins enthält die Ränder des zu druckenden Dokuments. Die Einheit ist ein typografischer Point, das ist 1/72 Zoll (0,35378 cm).
Die Default-Werte für pcDocMargins sind Null, pcDocMargins muss gesetzt werden, bevor das Dokument gedruckt wird.

Syntax:	UI-Code	**pcDocMargins** = **left, top, right, bottom**
	Schreiben	**<obj>.pcDocMargins** = **left, top, right, bottom**
	Lesen:	**<numVar>** = **<obj>.pcDocMargins** (**n**)
	top:	Oberer Rand, in Point
	left:	Linker Rand, in Point
	right:	Rechter Rand, in Point
	bottom:	Unterer Rand, in Point
	n:	Information, welcher Wert gelesen werden soll, siehe Tabelle (PS steht für Page-Size)

Konstante	Wert	Information
PS_LEFT_MARGIN	0	Linker Rand
PS_TOP_MARGIN	1	Oberer Rand
PS_RIGHT_MARGIN	2	Rechter Rand
PS_BOTTOM_MARGIN	3	Unterer Rand

Beispiel: Berechnung der Breite zu bedruckenden Bereichs im Dokument

```
DIM width
  width = MyPrintControl.pcDocSize(PS_WIDTH) \
        - MyPrintControl.pcDocMargins(PS_LEFT) \
        - MyPrintControl.pcDocMargins(PS_RIGHT)
```

pcLayout

Die Instancevariable pcLayout muss je nach Breite und Höhe des Dokuments entweder auf den Wert PL_PAPER (Defaultwert, normales Papier, Hochformat) oder PL_PAPER_LANDSCAPE (normales Papier, Querformat) gesetzt werden. Prinzipiell unterstützt GEOS weitere Layouttypen. Diese sind weiter unten, im Kapitel 4.14.5.4 ohne Gewähr aufgelistet.

Syntax:	UI-Code	**pcLayout** = **layoutWert**
	Schreiben	**<obj>.pcLayout** = **layoutWert**
	Lesen:	**<numVar>** = **<obj>.pcLayout**

Konstante	Wert	Bedeutung
PL_PAPER	0	Normales Papier, Hochformat
PL_PAPER_LANDSCAPE	8	Normales Papier, Querformat

4.14.4.2 Mehrseitiger Druck

Wenn Sie Dokumente mit mehreren Seiten drucken wollen müssen Sie im Actionhandler des "Drucken" Buttons den zu druckenden Seitenbereich festlegen. Ein setzen der Werte im UI-Code ist möglich, aber nur sinnvoll, wenn die Anzahl der zu druckenden Seiten nicht variabel ist.

totalPageRange

Die Instancevariable totalPageRange enthält die Seitennummern des Dokuments. Der erste Wert (erste zu druckende Seite) ist meist 1, es sind jedoch auch die Null oder sogar negative Werte erlaubt. Der Maximalwert beträgt jeweils 32767.
Wenn Ihr Programm nur eine einzelne Seite drucken kann (wie z.B. GeoDraw) brauchen Sie keinen Wert für totalPageRange zu setzen, da die Defaultwerte für Startseite und Endseite jeweils 1 sind.

Syntax:	UI-Code	**totalPageRange** = **start**, **end**
	Schreiben	**<obj>**.**totalPageRange** = **start**, **end**
	Lesen:	**<numVar>** = **<obj>**.**totalPageRange** (**n**)
	start:	Nummer der ersten Seite
	end:	Nummer der letzten Seite
	n:	Information, welcher Wert gelesen werden soll
		n = 0: erste druckbare Seite lesen
		n = 1: letzte druckbare Seite lesen

userPageRange

Die Instancevariable userPageRange enthält den vom Nutzer zum Drucken ausgewählten Seitenbereich. Damit der Nutzer die Möglichkeit hat, den zu druckenden Seitenbereich im Drucken-Dialog auszuwählen, muss das Bit PCA_PAGE_CONTROLS in der Instancevariable pcAttrs gesetzt sein. Per Default ist dieses Bit gesetzt.
Bevor das PrintControl-Objekt den Drucken-Dialog anzeigt korrigiert es eventuell widersprüchliche Werte von totalPageRange und userPageRange. Wurde zum Beispiel im UI-Code userPageRange auf einen Bereich gesetzt, der von totalPageRange nicht abgedeckt wird, so korrigiert das PrintControl-Objekt die Werte für userPageRange, bevor es den Dialog anzeigt.

Wenn Sie dem Nutzer die Möglichkeit geben, die zu druckenden Seiten auswählen, müssen Sie auch selbst dafür sorgen, dass genau diese Seiten gedruckt werden. Dazu müssen Sie userPageRange im OnPrint-Handler auslesen und entsprechend berücksichtigen. Das PrintControl-Objekt stellt nur die passenden UI-Objekte bereit, es benutzt die vom Nutzer ausgewählten Werte nicht selbst.

Syntax:	UI-Code	**userPageRange** = **start**, **end**
	Schreiben	**<obj>**.**userPageRange** = **start**, **end**
	Lesen:	**<numVar>** = **<obj>**.**userPageRange** (**n**)
	start:	Nummer der ersten Seite
	end:	Nummer der letzten Seite
	n:	Information, welcher Wert gelesen werden soll n = 0: erste ausgewählte Seite lesen n = 1: letzte ausgewählte Seite lesen

Achtung: Die Werte für userPageRange sind nur gültig, wenn folgendes gilt:

- Die Drucken-Dialogbox ist oder war schon auf dem Schirm.
- Das Bit PCA_PAGE_CONTROLS in der Instancevariable pcAttrs ist gesetzt. Per Default ist dieses Bit gesetzt.

4.14.4.3 Nutzereigaben verifizieren

Das PrintControl-Objekt sorgt im Normalfall selbst dafür, dass die Nutzereingaben im Drucken-Dialog konsistent sind. Zum Beispiel korrigiert es die zu druckenden Seitennummer automatisch, wenn der Nutzer für die letzte zu druckende Seite eine kleinere Seitennummer angegeben hat, als für die erste zu druckende Seite.
Falls Sie aber eigene UI-Objekte in der Drucken-Dialogbox haben (Instancevariable pcAppUI), bei denen der Nutzer inkonsistente Angaben machen könnte, müssen Sie diese Überprüfung selbst durchführen. Zu diesem Zweck gibt es die Instancevariable OnVerifyPrint und die Methode PrintingVerified. Damit der OnVerifyPrint-Handler gerufen wird, müssen Sie zusätzlich das Bit PCA_VERIFY-_PRINT in der Instancevariablen pcAttrs setzen.

OnVerifyPrint

Die Instancevariable OnVerifyPrint enthält einen Actionhandler. Er muss als ButtonAction deklariert sein wird aufgerufen, wenn der Nutzer den "Drucken" Button im Drucken-Dialog aktiviert hat, aber noch bevor der Drucken-Dialog geschlossen wird. Sie sollten hier die Konsistenz der Nutzereingaben prüfen und den Nutzer gegebenenfalls auf seinen Fehler aufmerksam machen. Alternativ können Sie auch die Nutzereingaben korrigieren, um sie konsistent zu machen.
Die letzte Anweisung in diesem Handler muss der Aufruf der Methode PrintingVerified sein.

Syntax:	UI-Code	**OnVerifyPrint** = **<Handler>**
	Schreiben	**<obj>**.**OnVerifyPrint** = **<Handler>**
	Lesen:	—
	<Handler>:	Name des Actionhandlers Der Handler muss als ButtonAction deklariert sein

Achtung! Der OnVerifyPrint-Handler wird nur gerufen, wenn das Bit PCA_-VERIFY_PRINT in der Instancevariablen pcAttrs gesetzt ist.

PrintingVerified

Die Methode PrintingVerified muss innerhalb des OnVerifyPrint Handlers aufgerufen werden. Sie teilt dem PrintControl-Objekt mit, dass das Überprüfen der Nutzereingaben abgeschlossen ist.
Wird als Parameter TRUE übergeben war die Überprüfung erfolgreich. Die Drucken-Dialogbox wird geschlossen und das Drucken wird fortgesetzt.
Wird als Parameter FALSE übergeben war die Überprüfung nicht erfolgreich. Die Drucken-Dialogbox wird nicht geschlossen und der Nutzer kann seine Eingaben korrigieren.

Syntax:	BASIC-Code	**<obj>**.**PrintingVerified** TRUE I FALSE

4.14.4.4 Starten des Ausdrucks

Als letzte Operation im Actionhandler des "Drucken" Buttons müssen Sie den Ausdruck starten, indem Sie die InitiatePrint-Methode aufrufen. Üblicher Weise vor vorher noch ein Name für den Printjob festgelegt.

printJobName$

Der Spooler benötigt zum Drucken einen Namen, über den der Nutzer den Druckjob identifizieren kann. Dieser Name wird in der Dialogbox "Drucker-steuerung" (aus dem Express-Menü) angezeigt. Durch Belegen der Instance-variablen printJobName$ legen Sie diesen Namen fest. Üblicher Weise wird hier der Name des auszudruckenden Dokuments verwendet. Wenn Ihr Programm nicht mit Dokumenten arbeitet sollten Sie einen anderen, aussagekräftigen Namen (max. 32 Zeichen) verwenden. Die können aber auch den Defaultwert ("R-BASIC Print Job") lassen.
Sie sollten printJobName$ immer im Actionhandler des Drucken-Buttons setzen. Wenn Sie ihn nur im UI-Code setzen wird er vom R-BASIC Translator nicht erkannt und kann daher nicht übersetzt werden.

Syntax:	UI-Code	**printJobName$** = **"name"**
	Lesen:	—
	Schreiben:	**<obj>**.**printJobName$** = **"name"**
	"name":	Bezeichnung des Druckjobs, max. 32 Zeichen

Tipp: Wenn Sie mit dem PostScript Druckertreiber in eine Datei Drucken und dann das Windows-Programm PDFCreator zum Umwandeln der entstandenen Dateien in eine PDF-Datei verwenden, wird printJobName$ auch als Vorschlag für den Namen der PDF-Datei verwendet.

InitiatePrint

Üblicher Weise wird diese Methode am Ende des Actionhandlers des "Drucken" Buttons gerufen. Vorher sollten Sie das PrintControl-Objekt initialisiert haben. Nach dem Aufruf der Methode InitiatePrint zeigt das PrintControl-Objekt die "Drucken"-Dialogbox an und der eigentliche Druckvorgang beginnt.

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

4.14.4.5 Ein Beispiel

Das folgende Codefragment zeigt die prinzipiellen Schritte für einen Actionhandler des "Drucken"-Buttons. Alle Zahlen geben Werte in typografischen Punkt (pt) an. Sie sind nur als Beispiele zu verstehen.

```
BUTTONACTION PrintButtonHandler

  ' Dokumentgröße: ca. 17,6 cm x 28,2 cm
  DemoPrintControl.pcDocSize = 500, 800

  ' Ränder: links 5,3 cm, sonst 1,7 cm
  DemoPrintControl.pcDocMargins = 150, 50, 50, 50

  ' Layout: Papier, Hochformt
  DemoPrintControl.pcLayout = PL_PAPER

  ' Wir haben 5 Seiten
  DemoPrintControl.totalPageRange = 1, 5
  DemoPrintControl.userPageRange = 1, 5

  DemoPrintControl.printJobName$ = "Dokument Size Demo"
  DemoPrintControl.InitiatePrint
END ACTION
```

4.14.5 Der Druckprozess

4.14.5.1 Die Standardprozedur

Einige Schritte müssen für jeden Druckprozess durchgeführt werden. Diese sind hier beschrieben. Besonderheiten für Spezialfälle finden Sie in den weiteren Unterkapiteln.

OnPrint

Der OnPrint-Handler erledigt die eigentliche Arbeit des Druckens. Er wird aufgerufen, wenn der Nutzer den "Drucken" Button im "Drucken"-Dialog aktiviert. OnPrint-Handler müssen als DrawAction deklariert sein. Der Programmierer kann davon ausgehen, dass das zu bedruckende Blatt Papier der Bildschirm ist. Die können die Druckausgaben (Text und Grafik) in beliebiger Reihenfolge an beliebiger Position ausgeben. Es ist nicht erforderlich, dass Sie "von oben nach unten" vorgehen.
Um eine neue Seite zu beginnen müssen Sie die Methode PrintNewPage aufrufen. Am Ende des Druckens **muss** der Aufruf der Methode **Printing-Completed** erfolgen.

Syntax: UI-Code **OnPrint** = **<Handler>**
Schreiben **<obj>.OnPrint** = **<Handler>**
Lesen: —
<Handler>: Name des Actionhandlers
Der Handler muss als DrawAction deklariert sein.

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

Bedeutung der Parameter:

- Der Parameter "sender" enthält das PrintControl-Objekt.
- Der Parameter "height" enthält die Höhe des bedruckbaren Bereichs des Papiers, gemessen in pt, also die Höhe des Papiers abzüglich der Druckerränder.
- Der Parameter "width" enthält die Breite des bedruckbaren Bereichs des Papiers, gemessen in pt, also die Breite des Papiers abzüglich der Druckerränder.

Hingegen enthalten die globalen Variablen MaxX und MaxY die maximale x- bzw. y-Koordinate Ihres Dokuments, ohne Berücksichtigung der Ränder. Das entspricht den Werten, die in der Instancevariablen pcDocSize abgelegt wurden. Da die Grafikkoordinaten bei Null beginnen enthalten MaxX und MaxY jeweils den um 1 verminderten Wert.

Wichtig: Es ist nicht erlaubt während des Druckens ein anderes Objekt zum Screen zu machen, auch nicht kurzzeitig. Wenn Sie z.B. Debugging-Informationen

ausgeben wollen müssen Sie ein Textobjekt oder die Anweisung MsgBox verwenden.

Beim Drucken werden alle Koordinaten in der Einheit "typografischer Punkt" (pt) angegeben. Ein Zoll (2,54 cm) hat genau 72 pt. Damit gilt:

```
    1 pt ≈ 0,35278 mm
    1 mm ≈ 2,8346 pt
```

Für genauere Umrechnungen sollten Sie folgende Formeln verwenden

```
  position_in_Punkt  = position_in_cm * 72 / 2,54
  position_in_cm     = position_in_Punkt * 2,54 / 72
```

GEOS trennt grundsätzlich die Größe des Dokuments von der Größe des Papiers im Drucker. Wenn ihr Dokument nicht aufs Papier passt fragt der Spooler nach, ob Sie das Dokument über mehrere Seiten verteilen oder die Größe anpassen soll. Sie können also drucken, ohne sich um die Größe des Papiers im Drucker Gedanken zu machen. Wenn Sie wünschen, die Druckausgaben manuell an die Größe des Papiers und an die Randeinstellungen des Druckers anzupassen, lesen Sie bitte den nächsten Abschnitt.

Beispiel: Ein sehr einfacher OnPrint-Handler. Es wird vorausgesetzt, dass das PrintControl-Objekt bereits korrekt initialisiert ist.

```
DRAWACTION  PrintHandler

  FontSetGeos(FID_SANS, 12)
  Print atXY 200, 300;"R-BASIC Print Test"
  FillEllipse 220, 320, 420, 520, LIGHT_RED

  sender.PrintingCompleted
END ACTION
```

Moderne Drucker können häufig bis an den Blattrand drucken oder zumindest bis knapp davor, auch wenn der Druckertreiber einen breiteren Rand meldet. Da es erlaubt ist, auch über den vom Druckertreiber gemeldeten Rand hinaus zu drucken können Sie den folgenden OnPrintHandler verwenden um die wahren Fähigkeiten Ihres Druckers auszutesten. Es wird wieder vorausgesetzt, dass das PrintControl-Objekt bereits korrekt initialisiert ist. Insbesondere muss pcDocSize belegt worden sein, damit MaxX und MaxY gültige Werte enthalten.

```
DRAWACTION  PrintHandler
  INK LIGHT_BLUE
  FillEllipse -100, -100, 100, 100
  FillEllipse MaxX - 100, MaxY - 100, MaxX + 100, MaxY + 100

  sender.PrintingCompleted
END ACTION
```

Beispiele für typische Situationen finden Sie bei den R-BASIC Beispieldateien im Ordner R-BASIC\Beispiel\Objekte\Drucken.

PrintingCompleted

Die Methode PrintingCompleted informiert das PrintControl-Objekt, dass Ausgabe von Grafik und Text beendet ist und dass der Spooler die Daten an den Drucker senden kann.

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

PrintingCancelled

Die Methode PrintingCancelled informiert das PrintControl-Objekt, dass der Druck abgebrochen werden soll. Sie kann anstelle der Methode PrintingCompleted aufgerufen werden. Das PrintControl-Objekt fordert dann den Spooler auf, alle bisher gedruckten Daten zu verwerfen.

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

printMode

Einige Drucker bieten im "Drucken"-Dialog die Option, in hoher, mittlerer oder niedriger Qualität zu drucken. Das PrintControl-Objekt leitet die diesbezügliche Nutzerauswahl direkt an den Spooler weiter, so dass Sie sich darum nicht weiter kümmern müssen. In sehr seltenen Fällen könnte man diese Information jedoch benötigen, z.B. um die zu druckenden Daten auszuwählen. Dann kann man die Instancevariable printMode abfragen. Sie liefert einem numerischen Wert, je nachdem ob in hoher (4), mittlerer (2) oder niedriger Qualität (0) gedruckt werden soll.

Syntax: BASIC-Code **<numVar> = <obj>.printMode**

Hinweis: Der Wert von printMode ist erst gültig, nachdem die Drucken-Dialogbox angezeigt wurde (d.h. er ist gültig im OnVerifyPrint- und im OnPrint-Handler).

4.14.5.2 Die Papiergröße verwenden

Oftmals ist es gewünscht, die Druckausgaben manuell an die Größe des Papiers und an die Randeinstellungen des Druckers anzupassen. In diesem Fall sollten Sie Größe, Ränder und Layout des Dokuments nicht im Actionhandler des "Drucken"-Buttons, sondern im OnPrint-handler setzen.

Um die Größe des Papiers und die Druckerränder zu erfahren können Sie folgendermaßen vorgehen:

- Abfrage der Instancevariablen pcPaperSizeInfo. Diese liefert alle benötigten Informationen.
- Der Parameter "width" des Handlers enthält die Breite des bedruckbaren Bereichs des Papiers, gemessen in pt, also die Breite des Papiers abzüglich der Druckerränder.
- Der Parameter "height" des Handlers enthält die Höhe des bedruckbaren Bereichs des Papiers, gemessen in pt, also die Höhe des Papiers abzüglich der Druckerränder.

Hingegen enthalten die globalen Variablen MaxX und MaxY die maximale x- bzw. y-Koordinate Ihres Dokuments, ohne Berücksichtigung der Ränder. Das entspricht den Werten, die in der Instancevariablen pcDocSize abgelegt wurden. Da die Grafikkoordinaten bei Null beginnen enthalten MaxX und MaxY jeweils den um 1 verminderten Wert.

pcPaperSizeInfo

Die Instancevariable pcPaperSizeInfo enthält alle Informationen über das im Drucker befindliche Papier. Genau genommen enthält sie die Informationen über das Papier und den Drucker, die der Nutzer im "Drucken" -Dialog ausgewählt hat. PcPaperSizeInfo kann nur gelesen werden und ist nur innerhalb des OnPrint-Handlers und des OnVerifyPrint-Handlers gültig.

Syntax Lesen: **<numVar>** = **<obj>**.**pcPaperSizeInfo** (**n**)
n: Welche Information soll gelesen werden
Erlaubte Werte: siehe Tabelle

Tabelle: Informationen, die pcPaperSizeInfo liefern soll. Alle Werte werden in typografischen Punkt (pt) angegeben.

Konstante	Wert	Information
PS_LEFT_MARGIN	0	Linker Rand
PS_TOP_MARGIN	1	Oberer Rand
PS_RIGHT_MARGIN	2	Rechter Rand
PS_BOTTOM_MARGIN	3	Unterer Rand
PS_WIDTH	4	Breite des Papiers
PS_HEIGHT	5	Höhe des Papiers
PS_LAYOUT	6	Layout-Information

Das folgende Codefragment zeigt das prinzipielle Vorgehen: Wir passen Größe, Ränder und Layout des Dokuments dem Papier im Drucker an.

```
DRAWACTION  DoPrintHandler

  DemoPrintControl.pcDocMargins = \
        DemoPrintControl.pcpaperSizeInfo(PS_LEFT_MARGIN), \
        DemoPrintControl.pcpaperSizeInfo(PS_TOP_MARGIN), \
        DemoPrintControl.pcpaperSizeInfo(PS_BOTTOM_MARGIN), \
        DemoPrintControl.pcpaperSizeInfo(PS_RIGHT_MARGIN)

  DemoPrintControl.pcDocSize = \
        DemoPrintControl.pcPaperSizeInfo(PS_WIDTH), \
        DemoPrintControl.pcPaperSizeInfo(PS_HEIGHT)

  DemoPrintControl.pcLayout = \
        DemoPrintControl.pcPaperSizeInfo(PS_LAYOUT)

  < ... hier Grafik zeichnen ... >

  ' Printcontrol informieren, dass wir fertig sind
  DemoPrintControl.printJobName$ = "Papier Drucktest"
  DemoPrintControl.PrintingCompleted

END ACTION
```

4.14.5.3 Mehrere Seiten drucken

Wenn Sie ein mehrseitiges Dokument drucken wollen, müssen Sie die Instancevariable userPageRange des PrintControl-Objekts abfragen und selbst dafür sorgen, dass nur genau die Seiten gedruckt werden, die der Nutzer ausgewählt hat. Um eine neue Seite zu beginnen rufen Sie die Methode PrintNewPage auf. Außerdem sollten Sie den Nutzer über den Druckfortschritt informieren, indem sie regelmäßig, z.B. am Beginn jeder neuen Seite eine der Methoden ReportProgress oder ReportProgressText aufrufen. Das ermöglicht dem Nutzer auch, den Druckprozess vorzeitig abzubrechen.

PrintNewPage

Die Methode PrintNewPage beginnt eine neue Seite. Sie wird bei mehrseitigen Druckaufträgen als Trennung zwischen den einzelnen Druckseiten benötigt.

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

ReportProgress, ReportProgressText

Mit den Methoden ReportProgress und ReportProgressText können Sie dem Nutzer den Druckfortschritt anzeigen. Das empfiehlt sich bei langwierigen oder mehrseitigen Druckaufträgen. Voraussetzung ist, dass das Bit PCA_SHOW-_PROGRESS in der Instancevariablen pcAttrs gesetzt ist. Dieses Bit ist per Default gesetzt.
Der Fortschrittsdialog enthält einen "Abbrechen" Button. ReportProgress und ReportProgressText liefern TRUE zurück, wenn der Nutzer den "Abbrechen" Button gedrückt hat, andernfalls liefern sie FALSE. Es ist der Job des Programmierers, den Druckprozess abzubrechen, wenn der Nutzer den "Abbrechen"-Button gedrückt hat.

Syntax: **<numVar>** = **<obj>.ReportProgress** (**type**, **wert**)
type: RPT_PAGE oder RPT_PERCENT, siehe Tabelle
wert: Aktuelle Seite (bei RPT_PAGE) oder Prozentwert (bei RPT_PERCENT)
Syntax: **<numVar>** = **<obj>.ReportProgressText** (**text$**)
text$: Anzuzeigender Text

Damit die Methode **ReportProgress** verwendet werden kann muss in der Instancevariablen pcAttrs neben dem per Default gesetzten Bit PCA_SHOW-_PROGRESS mindestens eins der Bits PCA_PROGRESS_PAGE oder PCA_PROGRESS_PERCENT gesetzt sein. Üblicherweise wird ReportProgress am Anfang jeder zu druckenden Seite gerufen. Zur Arbeit mit ReportProgress sind die folgenden Konstanten definiert:

Konstante	Wert	Bedeutung
RPT_PAGE	0	Anzeige in der Form "Seite N von X Seiten ". Das Bit PCA_PROGRESS_PAGE muss gesetzt sein.
PRT_PERCENT	2	Anzeige in der Form: "Verlauf N". Das Bit PCA_PROGRESS_PERCENT muss gesetzt sein.

Damit die Methode **ReportProgressText** verwendet werden kann muss in der Instancevariablen pcAttrs nur das Bit PCA_SHOW_PROGRESS gesetzt sein. Das ist per Default der Fall. Sie sind völlig frei in der Gestaltung des anzuzeigenden Textes. Verwenden Sie ReportProgressText, wenn ReportProgress Ihren Ansprüchen nicht genügt.

Beispiel: Ein OnPrint-Handler der mehrere Seiten ausdrucken kann. Die Routine Printpage(n) erledigt den eigentlichen Ausdruck der Seite. Wir kapseln den Aufruf dieser Routine in die Anweisungen ScreenSaveState / ScreenRestoreState. Damit kann die Routine beliebige Koordinatentransformationen vornehmen (z.B. den Koordinatenursprung so setzen, dass die Ränder nicht bedruckt werden).
Den kompletten Sourcecode finden Sie im Ordner R-BASIC\Beispiel\Objekte\Drucken.

```
DRAWACTION  DoPrintHandler
DIM pstart, pend, n, cancel, info$

  ' Auslesen der vom Nutzer ausgewählten Seiten
  pstart = DemoPrintControl.userpageRange(0)
  pend = DemoPrintControl.userpageRange(1)

  ' ausgewählte Seiten drucken. Mit Abfrage ob Abbruch
  FOR n = pstart TO pend
    IF n <> pstart then DemoPrintControl.PrintNewPage
    info$ = "Drucke Seite"+Str$(n)
    cancel = DemoPrintControl.ReportProgressText$ (info$)
    IF cancel THEN
        DemoPrintControl.PrintingCancelled
        RETURN
        End IF

    ScreenSaveState       ' Ausgangszustand sichern
    Printpage(n)          ' Seite Drucken
    ScreenRestoreState    ' Ausgangszustand wieder herstellen

    NEXT N

  DemoPrintControl.PrintingCompleted

END ACTION
```

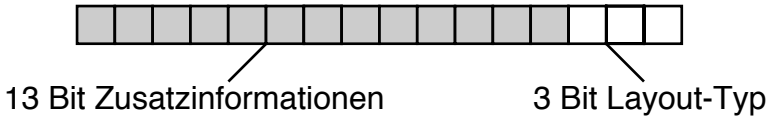
4.14.5.4 Hintergrundinformationen zum Thema Layout

Für die meisten Programmierer wird es ausreichen, die Instancevariable pcLayout auf einen der Werte PL_PAPER (Defaultwert, normales Papier, Hochformat) oder PL_PAPER_LANDSCAPE (normales Papier, Querformat) zu setzen. Das ist prinzipiell auch für Umschläge anwendbar.

PC/GEOS definiert jedoch weitergehende Layoutinformationen für Dokumente. So kann man z.B. festlegen ob Briefumschläge links oben oder rechts unten beschriftet werden sollen. Diese Layoutinformation sind jedoch offensichtlich als Information für den Programmierer gedacht, der dann die Adresse passend positionieren kann. Eine Auswirkung auf den Spooler konnte bei eigenen Tests jedenfalls nicht bestätigt werden.

Die folgenden Informationen sind direkt dem PC/GEOS SDK entnommen. Der Programmierer von R-BASIC garantiert nicht für Vollständigkeit und Richtigkeit.

Die Instancevariable pcLayout enthält die Layout-Informationen für das Dokument. Der numerische Wert von pcLayout ist ein 16-Bit Bitfeld, das folgendermaßen aufgebaut ist:

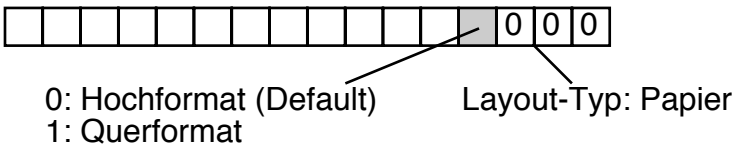


GEOS unterstützt drei Layout-Typen: Papier (PL_PAPER) Umschläge (PL_ENVELOPE) und Etiketten (PL_LABEL). Zur Arbeit mit dem Layout sind die folgenden Konstanten definiert.

Konstante	Wert	Bedeutung
PL_PAPER	0	Normales Papier, Hochformat
PL_PAPER_LANDSCAPE	8	Normales Papier, Querformat
PL_ENVELOPE	2	Umschlag
PL_LABEL	4	Etiketten

Normales Papier

Für normales Papier hat pcLayout die folgende Struktur:



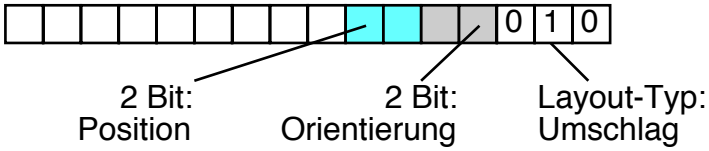
Die Konstanten PL_PAPER (= 0) und PL_PAPER_LANDSCAPE (= 8) können pcLayout direkt zugewiesen werden um das entsprechende Layout einzustellen.

Beispiel:

```
PrintControl MyPrintControl
  pcLayout = PL_PAPER_LANDSCAPE
  < ... >
End Object
```

Umschlag

Für Umschläge hat pcLayout die folgende Struktur:



Für Orientierung und Position sind die folgenden Werte zugelassen:

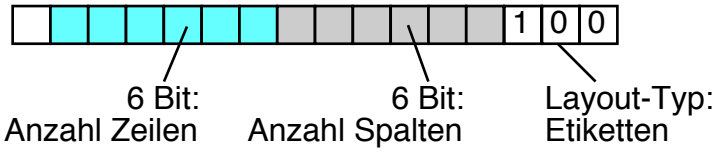
Wert	Wirkung Position	Wirkung Orientierung
0	Links	Hochformat links
1	Zentriert	Hochformat rechts
2	Rechts	Querformat links
3	—	Querformat rechts

Der Wert für pcLayout berechnet sich nach folgender Formel:

```
  pcLayout = PL_ENVELOPE + 8 * Orientierung + 32 * Position
```

Etiketten

Der Druck von Etiketten bedeutet, dass eine bestimmte Anzahl von Etiketten (Aufklebern) neben- und untereinander gedruckt werden sollen. Dieses Prinzip lässt sich z.B. auch für Visitenkarten verwenden. Für Etiketten hat pcLayout die folgende Struktur:



Für die Anzahl der Etiketten nebeneinander (Spalten) und untereinander (Zeilen) stehen jeweils 6 Bit zur Verfügung. Der Maximalwert ist also 63. Zur Berechnung des Layoutwerts benutzen Sie die folgende Formel:

```
  pcLayout = PL_LABEL + 8 * spalten + 512 * zeilen
```

Leider berücksichtigt der Spooler diese Informationen nicht, wenn er ein Dokument auf mehrere Seiten aufteilt.

4.14.6 Drucken von Text

Um Text auf den Drucker auszugeben haben Sie zwei Möglichkeiten. Die einfachste Möglichkeit ist die Verwendung der Print-Anweisung. Dieses Kapitel enthält einige Tipps zu dieser Möglichkeit. Die andere Möglichkeit ist, ein Text-Objekt direkt auszudrucken. Diese Möglichkeit wird im nächsten Kapitel beschrieben.

Grundsätzlich können Sie die Print-Anweisung genauso verwenden, wie dies bei der Ausgabe von Text in ein BitmapContent-Objekt oder im OnDraw-Handler eines beliebigen Objekts möglich ist. Das heißt insbesondere, dass Sie Text - wie jede andere Grafik auch - nicht von oben nach unten ausgeben müssen. Sehr häufig werden Sie dazu die Anweisung "Print atXY .. " verwenden.

Zusätzlich sollten sie folgende Fakten kennen:

- Intern verwaltet R-BASIC auch bei Drucken die Cursorposition. Mehrere Print-Anweisungen werden also linksbündig und untereinander erscheinen.
- Alle Formatierungsfunktionen der Print-Anweisung (Trennzeichen Komma und Semikolon, Positionierung mit at und atXY, Farbanweisungen) sind erlaubt.
- Der Standard-Font ist URW Mono (fontID: FID_MONO) 14 Punkt. Es empfiehlt sich daher mit der Anweisung FontSetGEOS() einen günstigeren Font einzustellen. Um den Texthintergrund transparent zu machen verwenden Sie die Anweisung "Paper BG_TRANSPARENT".
- Der Ausgabescreen ist im "Layout-Modus". Das heißt es gibt keine Restriktionen für die Textausgabe. Sie können negative Koordinaten verwenden oder über den Rand schreiben. Es erfolgt **kein** automatischer Zeilenumbruch an der rechten Blattseite.
- Das Drucken von Block-Grafik-Zeichen ist möglich.

Window und Locate

Es ist möglich, die Anweisungen Window und Locate (siehe Programmierhandbuch, Kapitel 2.9.2.1) zu verwenden. Um den automatischen Zeilenumbruch am rechten Fensterrand zu aktivieren, müssen Sie außerdem in den "Page-Mode" wechseln. Die verschiedenen Ausgabemodi sind im Programmierhandbuch, Kapitel 2.9.2.4 (Scrollmode, Pagemode und Layoutmode) beschrieben. Die folgende Codesequenz grenzt das Textausgabefenster auf 51 Zeichen Breite und 31 Zeilen Höhe ein, wobei die linke obere Fensterecke auf der Cursorposition 2:5 liegt. **Achtung!** Im Geos-Font-Mode sind die Zeichen unterschiedlich breit. Es passen damit meist mehr als 51 Zeichen in die Zeile!
Die Anweisung Print Chr(17); stellt den Pagemode ein. Gelangt der Cursor ans Fensterende springt er wieder nach links oben.

```
Window 2, 32, 5, 55
Print Chr$(17);
```

Profitipp: Größe eines Zeichens

Die Anweisungen Window und Locate ermitteln die (durchschnittliche) Breite und Höhe eines Zeichens aus den globalen Variablen "printFont.charWidth" und "printFont.lineHeight" (siehe Spezielle Themen, Kapitel 2.7).

4.14.7 Drucken spezieller Objekte

Es ist möglich, Objekte anzuweisen, sich selbst auf den Bildschirm zu zeichnen, das Objekt also auszudrucken. Dazu gibt es die Anweisung PrintObj. Folgende Objektklassen sind dafür vorgesehen, mit PrintObj korrekt zusammenzuarbeiten:

- Memo, InputLine, VisText und LargeText
- BitmapContent
- Canvas
- Image
- VisContent und VisObj

Alle anderen Objektklassen können auch mit PrintObj verwendet werden. Allerdings erwarten diese Objekte häufig, dass bestimmte Voraussetzungen erfüllt sind. Es kann z.B. sein, dass der Textfont korrekt gesetzt sein muss oder (sehr häufig), dass der Hintergrund bereits gelöscht (einfarbig) ist. Ob der Ausdruck dann Ihren Bedürfnissen entspricht, können Sie nur ausprobieren.

PrintObj

Die Routine PrintObj gibt ein Objekt an den Drucker aus. Das Objekt wird aufgefordert, sich selbst zu zeichnen. Die meisten Objekte müssen auf dem Bildschirm sein, um ausgedruckt werden zu können. Ausnahmen sind unten beschrieben. Sie können prüfen, ob ein generic Class Objekt druckbar ist, indem Sie die Instancevariable fullyVisible abfragen (siehe Objekthandbuch, Kapitel 3.2, Objekt States).

Syntax:	PrintObj **<obj>**, **x**, **y**	
	<obj>:	Objekt, dass ausgedruckt werden soll
	x:	x-Koordinate der linken oberen Ecke
	y:	y-Koordinate der linken oberen Ecke

Tipp für Profis: PrintObj kann für beliebige Screens, nicht nur beim Drucken, verwendet werden.

Textobjekte

Textobjekte drucken sich so aus, wie sie aktuell auf dem Bildschirm erscheinen. Das betrifft insbesondere die Schriftart, Schriftgröße usw. aber auch die Breite des Textobjekts. Es wird der komplette Text gedruckt, auch wenn Teile davon nicht auf dem Bildschirm zu sehen sind. Sie können die Anweisung ScreenSetClipRect verwenden (siehe Objekt-Handbuch, Kapitel 2.3.2, Clipping), um die Ausgabe auf einen bestimmten Bereich zu begrenzen.
Wenn Sie keinen druckbaren Font gesetzt haben (siehe Instancevariable fontID), wählt das Textobjekt selbst einen druckbaren Font aus.

Memo und InputLine: Je nach gesetzten Instancevariablen (backColor, readOnly, TextNoFrame, TextFrame) druckt das Textobjekt eine farbigen Hintergrund und/oder einen Rahmen. Rollbalken werden nicht mit gedruckt.

VisText und LargeText: Diese Textobjekte ignorieren die Hintergrundfarbe (Instancevariable backColor) und drucken den Text immer transparent. Sie können die Objekte selbst drucken oder das zugehörige VisContent.

BitmapContent

BitmapContent-Objekte können immer gedruckt werden, auch wenn
- das zugehörige View nicht auf dem Schirm ist,
- das BitmapContent mit einem View verbunden ist.

Um nur einen Ausschnitt aus einer Bitmap zu drucken können Sie die Anweisung ScreenSetClipRect verwenden (siehe Objekt-Handbuch, Kapitel 2.3.2, Clipping).

Neben der Nutzung der Routine PrintObj können Bitmaps auch über ihr Bitmaphandle gedruckt werden. Das folgende Codefragment demonstriert das.

```
DIM bmpHan as HANDLE
  bmpHan = DemoContent.GetBitmapHandle
  DrawBitmap bmpHan, 50, 100
```

Auf diese Weise können auch Offscreen Bitmaps (siehe Programmierhandbuch, Kapitel 2.8.6.4) gedruckt werden.

Canvas

Canvas-Objekte können immer gedruckt werden, auch wenn sie nicht auf dem Schirm sichtbar sind. Es müssen aber folgende Voraussetzungen erfüllt sein:
- Das Objekt arbeitet im "buffered" Modus.
- Der OnDraw-Handler muss mindestens einmal ausgeführt worden sein. Dazu muss das Objekt mindestens einmal auf dem Bildschirm sichtbar gewesen sein.

Hinweise:
- Captions von Canvas-Objekten werden eventuell nicht oder nicht korrekt gedruckt.
- Anstatt das Canvas-Objekt mit der Anweisung PrintObj zu drucken könnten Sie vielleicht den Code des OnDraw-Handlers in eine SUB auslagern und diese dann zum Drucken aufrufen.

Image

Image-Objekte können immer gedruckt werden, auch wenn sie nicht auf dem Schirm sichtbar sind. Voraussetzung ist, dass das Objekt seine Grafik geladen hat. Dazu muss es mindestes einmal auf dem Schirm sichtbar gewesen sein. Um nur einen Ausschnitt aus dem Bild des Image-Objekts zu drucken können Sie die Anweisung ScreenSetClipRect verwenden (siehe Objekt-Handbuch, Kapitel 2.3.2, Clipping).

VisContent und VisObj

VisContent und VisObj-Objekte können immer gedruckt werden, auch wenn
- das zugehörige View nicht auf dem Schirm ist,
- das VisContent mit einem View verbunden ist,
- das VisObj kein Parent hat.

Es müssen aber folgende Voraussetzungen erfüllt sein:
- Wenn das Objekt einen OnDraw-Handler hat, muss es im "buffered" Modus arbeiten.
- Der OnDraw-Handler muss mindestens einmal ausgeführt worden sein. Dazu muss das Objekt mindestens einmal auf dem Bildschirm sichtbar gewesen sein.

Wird ein VisContent-Objekt gedruckt, so werden seine Children automatisch mit gedruckt. Der Ausdruck beschränkt sich dabei nicht auf den im View sichtbaren Bereich. Um den zu druckenden Bereich einzuschränken können Sie die Anweisung ScreenSetClipRect verwenden (siehe Objekt-Handbuch, Kapitel 2.3.2, Clipping).

Falls Sie ein VisObj-Objekt drucken, das Children hat, werden die Children ebenfalls mit gedruckt.

Anstatt VisContent- oder VisObj-Objekte mit der Anweisung PrintObj zu drucken könnten Sie vielleicht den Code des OnDraw-Handlers in eine SUB auslagern und diese dann zum Drucken aufrufen. Dieses Vorgehen druckt allerdings die Children des VisContent-Objekts nicht mit.

View

Die Verwendung von PrintObj mit einem View-Objekt druckt nur einen Rahmen und die Rollbalken. Das Content-Objekt wird nicht mit gedruckt.

4.14.8 Unterstützung des Dokument-Interfaces

Falls Sie mit druckbaren Dokumenten arbeiten kann es passieren, dass der Nutzer das Dokument direkt über den Dateimanager (GeoManager, Menüpunkt "Drucken" im Dateimenü) ausdrucken will. Damit wird das Dokument zum Drucken an Ihr Programm übergeben.

Das Application-Objekt Ihres Programms aktiviert in diesem Fall automatisch das PrintControl-Objekt (und damit den "Drucken" Dialog). Allerdings müssen Sie noch sicherstellen, dass das PrintControl-Objekt korrekt initialisiert ist. Das passiert meistens im Actionhandler des "Drucken" Buttons. Deshalb müssen Sie dafür sorgen, dass dieser Actionhandler auch aufgerufen wird. Das ist sehr einfach.

Wenn Sie mit Dokumenten arbeiten haben Sie im Allgemeinen sowohl einen OnStartup-Handler als auch einen OnConnection-Handler. Diese Handler wissen bereits wie man ein übergebenes Dokument öffnet. Die Details dazu finden Sie im Handbuch "Spezielle Themen", Kapitel 15. Das einzige was Sie noch zusätzlich tun müssen ist, die Methode "Activate" des "Drucken" Buttons aufzurufen, wenn das Dokument zum Drucken übergeben wurde. Die entsprechende Information ist im Parameter "flags" des Handlers gespeichert. Wenn das Bit AF_FOR_PRINT gesetzt ist wurde das Dokument zum Drucken übergeben. In diesem Fall müssen Sie als allerletzte Aktion die Activate-Methode des "Drucken" Buttons aufrufen:

```
SYSTEMACTION MyOnStartupHandler

  <... hier alles andere erledigen ...>

  IF flags AND AF_FOR_PRINT THEN MyPrintButton.Activate
End Action
```

```
SYSTEMACTION MyOnConnectionHandler

  <... hier alles andere erledigen ...>

  IF flags AND AF_FOR_PRINT THEN MyPrintButton.Activate
End Action
```

Falls Sie nur einen "leeren" ActionHandler für den "Drucken" Button haben, das heißt, er macht nichts weiter als die InitiatePrint-Methode des PrintControl-Objekts aufzurufen, können Sie sich den Aufruf des Button-Handlers im OnStartup und im OnConnection Handler auch schenken. Das Application-Objekt ruft die Initiate-Print-Methode automatisch auf.

4.14.9 Tipps und Tricks

Es druckt nicht ...

Zu den häufigsten Fehlern beim Drucken gehören:

- Sie haben vergessen, die Instancevariablen pcDocSize und / oder pcDocMargins zu belegen oder Sie haben sie widersprüchlich belegt.
- Sie haben das PrintControl-Objekt nicht in den Objekt-Tree eingebunden.
- Sie haben die Methode PrintingCompleted nicht gerufen
- Sie haben das Bit PCA_VERIFY_PRINT in der Instancevariablen pcAttrs gesetzt, aber keinen OnVerifyPrint-Handler gesetzt oder dieser ruft die Methode PrintingVerified nicht.
- Sie haben ScreenSetClipRect verwendet und vergessen, das Clipping-Rechteck zurückzusetzen. Die Kombination ScreenSaveState / ScreenRestoreState ist hier leider **nicht geeignet**. Sie müssen ein neues, ausreichend großes Clipping-Rechteck setzen. Auf der sicheren Seite sind Sie mit der Anweisung:

```
ScreenSetClipRect -32000, -32000, 32000, 32000
```

Farbprobleme

Die Standardpalette von GEOS und dem Hostsystem (Windows, Linux) stimmt im Allgemeinen nicht überein. Bei Druckertreibern, die das nicht berücksichtigen oder bei der Umrechnung Fehler machen kann daher beim Ausdruck zu "unerklärlichen" Farbverfälschungen kommen. Beim Ausdruck über den PostScript-Treiber tritt das Problem sogar auf, wenn Sie mit einer der 16 Grundfarben (oder dem zugehörigen RGB-Wert) in eine 24-Bit-Bitmap zeichnen. In vielen Fällen stört dies nicht, aber falls doch können Sie das Problem mit folgenden Strategien umgehen:

- Vermeiden Sie die stark betroffenen Farben. Unter Windows und Linux sind das CYAN, LIGHT_CYAN, VIOLET und LIGHT_VIOLET.
- Benutzen Sie eine RGB-Farbe, deren RGB-Farbwerte geringfügig von den RGB-Werten der Standardfarben abweichen. Die folgende Routine ändert dazu den Rotanteil der Farbe um 1.

```
Function GetSaveColorRGB(index as real) as Real
DIM r,g,b
 r = RedOf(index);
 g = GreenOf(index);
 b = BlueOf(index);
 IF r > 0 THEN r = r-1 : else r=1
 Return RGB(r,g,b)
End Function
```

- Wenn Sie auf 256 Farben angewiesen sind können Sie statt der 16 Grundfarben (Farbwerte 0 bis 15) eine "ähnliche" Farbe aus der oberen 256-Farb-Palette (dem sogenannten Farbwürfel) verwenden. Die folgende Routine zeigt, wie man das machen kann.

```
Function GetSaveColor(index as real) as Real
  ON index SWITCH
     CASE BLACK: RETURN 0          ' 0 -> 0
     CASE BLUE: RETURN 43          ' 1 -> 43
     CASE GREEN: RETURN 58         ' 2 -> 58
     CASE CYAN: RETURN 61          ' 3 -> 61
     CASE RED: RETURN 148          ' 4 -> 148
     CASE VIOLET: RETURN 151       ' 5 -> 151
     CASE BROWN: RETURN 161        ' 6 -> 161
     CASE LIGHT_GRAY: RETURN 27    ' 7 -> 27
     CASE DARK_GRAY: RETURN 20     ' 8 -> 20
     CASE LIGHT_BLUE: RETURN 129   ' 9 -> 129
     CASE LIGHT_GREEN: RETURN 144  ' 10 -> 144
     CASE LIGHT_CYAN: RETURN 147   ' 11 -> 147
     CASE LIGHT_RED: RETURN 234    ' 12 -> 234
     CASE LIGHT_VIOLET: RETURN 237 ' 13 -> 237
     CASE YELLOW: RETURN 252       ' 14 -> 252
     CASE WHITE: RETURN WHITE      ' 15 -> 15
  End SWITCH
  RETURN index                  ' ansonsten: nicht ändern
End Function
```

Das Programm "TunePro" (© by Rabe-Soft) ermöglicht Ihnen, einen passenden Farbcode zu finden. Klicken Sie auf "UI anpassen" und dann auf "Farbe wählen". Alternativ werden Ihnen die R-G-B-Anteile einer Index-Farbe z.B. im "Flächenattribute" Dialog angezeigt, wenn Sie auf "Weitere Farben" klicken. Sie können dann die BASIC-Funktion IndexOf (r, g, b) verwenden um den zugehörigen Index (Farbcode) zu erfahren.

Drucken mit höherer Präzision

Beim Ausdruck können die Koordinaten mit einer Präzision von einem typografischen Punkt (1 pt, ca. 0,35 mm) angegeben werden. Will man Objekte präzisier platzieren muss man den Ausgabescreen herunterskalieren und dann die Koordinaten mit dem Skalierungsfaktor multiplizieren. Die folgende Sequenz zeichnet ein rotes Viereck der Kantenlänge 100.25 pt x 200.5 pt an die Position x = 50.25 und y = 75.5 bei einer Präzision von 0.25 pt.

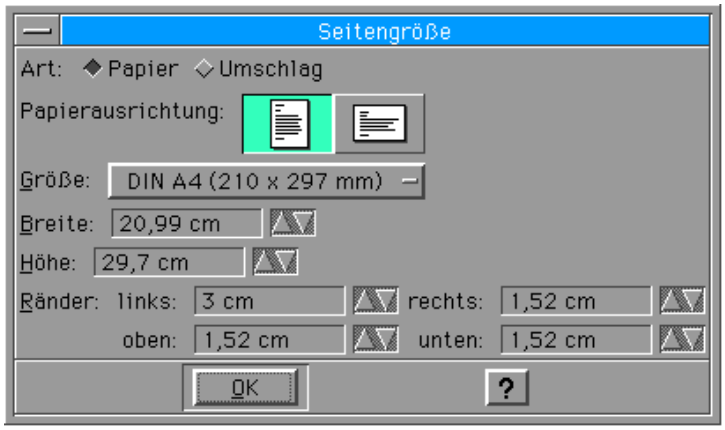
```
DIM x, y
  x = 50.25
  y = 75.5
  ScreenSetScale 1/4, 1/4
  FillRect 4*x, 4*y, 4*(x+110.25), 4*(y + 200.5), LIGHT_RED
```

Beachten Sie, dass GEOS die Breite von Linien nicht herunterskaliert. Um die Linienbreite auf 1/4 pt zu setzen verwenden Sie die Anweisung:

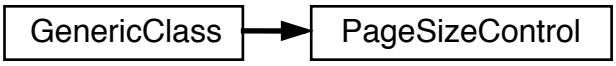
```
  graphic.lineWidth = 0.25
```

4.15 PageSizeControl

Das PageSizeControl-Objekt stellt in einer Dialogbox die UI bereit, um Größe, Ränder und Layout eines Dokuments auszuwählen. Das PageSizeControl-Objekt stellt nur die UI bereit, es ist Aufgabe des Programmierers die Werte auch zu verwenden.



Abstammung:



Spezielle Instance-Variablen:

Variable	Syntax im UI-Code	Im BASIC-Code
pageSize	—	lesen, schreiben
pageMargins	—	lesen, schreiben
pageLayout	—	lesen, schreiben
pscFeatures	pscFeatures = **featuresWert**	lesen, schreiben

Methoden:

Methode	Aufgabe
ShowDialog	PageSize-Dialogbox anzeigen

Beim Programmstart lädt das PageSizeControl-Objekt Seitengröße, Ränder und Layout vom Treiber des Standarddruckers des Systems (auch wenn der Drucker nicht angeschlossen ist). Es ist deshalb nicht möglich, die Werte für pageSize, pageMargins, und pageLayout im UI-Code zu setzen.

pscFeatures

Die Instancevariable pscFeatures enthält die Information, welche der möglichen UI-Elemente des PageSizeControl-Objekts angezeigt werden sollen.

Syntax	UI-Code:	**<obj>.pscFeatures** = **featuresWert**
	Lesen:	**<numVar>** = **<obj>.pscFeatures**
	Schreiben:	**<obj>.pscFeatures** = **featuresWert**
		featuresWert: Anzuzeigende UI, siehe Tabelle

Der Wert für pscFeatures enthält Bitflags. Jedes Bit schaltet ein UI-Element ein oder aus. **Wichtig!** das Bit PSCF_CUSTOM_SIZE muss **immer** gesetzt sein, sonst crasht das System. Es sind folgende Konstanten definiert:

Konstante	Wert	Anzuzeigende UI
PSCF_MARGINS	16	Eingabefelder für die Ränder
PSCF_CUSTOM_SIZE	8	Breite und Höhe des Dokuments
PSCF_LAYOUT	4	Hoch- oder Querformat
PSCF_SIZE_LIST	2	Liste verfügbarer Papiersorten
PSCF_PAGE_TYPE	1	Papier oder Umschlag
PSC_DEFAULT_FEATURES	15	CustomSize, Layout, Liste, PageType

Beispiel: Ein PageSizeControl-Objekt, dass die Eingabefelder für die Ränder anzeigt, aber die Auswahl Papier/Umschlag nicht zulässt. Beachten Sie, dass das PageSizeControl-Objekt einen Caption-Wert benötigt. In vielen Fällen wird das PageSizeControl-Objekt als FileMenuChild eines Primary-Objekts in der generic Tree eingebunden.

```
PageSizeControl DemoPageSizeControl
  Caption$ = "Seitengröße"
  pscFeatures = PSC_DEFAULT_FEATURES \
              + PSCF_MARGINS - PSCF_PAGE_TYPE
End OBJECT
```

pageSize

Die Instancevariable pageSize enthält die Größe des zu druckenden Dokuments, angegeben in typografischen Punkt (pt). Sie können die Größe des Dokuments völlig unabhängig vom Papier im Drucker wählen, allerdings ist es oft eine gute Idee, die Dokumentgröße an die Papiergröße anzupassen. Deswegen initialisiert das PageSizeControl-Objekt die Werte für pageSize mit der Papiergröße im Standarddrucker.
PageSize kann im UI-Code nicht gesetzt werden. Wenn Sie einen von der Papiergröße im Standarddrucker abweichenden Wert einstellen wollen müssen Sie das im OnStartup-Handler ihres Programms tun (siehe Beispiel unten).

Synatx	Schreiben:	**<obj>.pageSize** = **width**, **height**
	Lesen:	**<numVar>** = **<obj>.pageSize** (**n**)
	n:	Information, welcher Wert gelesen werden soll, siehe Tabelle (PS steht für Page-Size)

Tabelle: Mögliche Parameter beim Lesen von pageSize. Alle Werte werden in typografischen Punkt (pt) angegeben.

Konstante	Wert	Information
PS_WIDTH	4	Breite des Dokuments
PS_HEIGHT	5	Höhe des Dokuments

pageMargins

Die Instancevariable pageMargins enthält die Ränder des zu druckenden Dokuments, angegeben in typografischen Punkt (pt). Das PageSizeControl-Objekt initialisiert die Werte für pageMargins mit den Rändern des Papiers im Standarddrucker.
PageMargins kann im UI-Code nicht gesetzt werden. Wenn Sie einen von den Rändern im Standarddrucker abweichenden Wert einstellen wollen, müssen Sie das im OnStartup-Handler ihres Programms tun (siehe Beispiel unten).

Synatx	Schreiben:	**<obj>.pageMargins** = **left**, **top**, **right**, **bottom**
	Lesen:	**<numVar>** = **<obj>.pageMargins** (**n**)
	n:	Information, welcher Wert gelesen werden soll, siehe Tabelle (PS steht für Page-Size)

Tabelle: Mögliche Parameter beim Lesen von pageMargins. Alle Werte werden in typografischen Punkt (pt) angegeben.

Konstante	Wert	Information
PS_LEFT_MARGIN	0	Linker Rand
PS_TOP_MARGIN	1	Oberer Rand
PS_RIGHT_MARGIN	2	Rechter Rand
PS_BOTTOM_MARGIN	3	Unterer Rand

pageLayout

Die Instancevariable pageLayout enthält die Layoutinformationen des zu druckenden Dokuments. Detaillierte Informationen zur Bedeutung des Werts von pageLayout finden Sie weiter oben, bei der Instancevariablen pcLayout des PrintControl-Objekts. Das PageSizeControl-Objekt unterstützt die Layouttypen PL_PAPER und PL_ENVELOPE. PL_LABEL wird nicht unterstützt.

Synatx	Schreiben:	**<obj>.pageLayout** = **layoutWert**
	Lesen:	**<numVar>** = **<obj>.pageLayout**
	layoutWert: Siehe pcLayout des PrintControl-Objekts	

Beispiel: Verwendung eines PageSizeControl-Objekts zur Einstellung der Dokument-Größe. Den kompletten Sourcecode finden Sie im Ordner R-BASIC\Beispiel\Objekte\Drucken.

UI-Code: Wir wollen auch die Seitenränder einstellen. Die logische OR-Operation setzt das Bit PSCF_MARGINS ohne die anderen Bits zu beeinflussen. Zur Demonstration haben wir auch noch ein Tastenkürzel (Strg-e) vergeben.

```
PageSizeControl DemoPageSizeControl
  Caption$ = "Seitengröße" , 1
  kbdShortcut = ASC("e") + KSM_CTRL + KSM_PHYSICAL
  pscFeatures = PSC_DEFAULT_FEATURES OR PSCF_MARGINS
End OBJECT
```

Wir möchten, dass unser PageSizeControl die Werte von Standarddrucker durch eigene Werte ersetzt. Deswegen müssen wir einen OnStartup-Handler vereinbaren.

```
Application DemoApplication
    Children = DemoPrimary
    OnStartup = AppStartupHander
END Object
```

Im AppStartupHander überschreiben wir die Werte, die unser PageSizeControl vom Standarddrucker gelesen hat. Der Faktor 72/2,54 rechnet einen cm-Wert in typografische Punt (pt) um.

```
SYSTEMACTION AppStartupHander
  ' Seitengröße: A4 (21,0 cm x 29,7 cm)
  ' Seitenränder: links 3 cm, sonst 1,5 cm
  ' Randgröße: 1,5 cm -> 1.5*72/2.54 = 42.52 pt
  DemoPageSizeControl.pageSize = 21*72/2.54, 29.7*72/2.54
  DemoPageSizeControl.pageMargins = 85, 43, 43, 43
  DemoPageSizeControl.pageLayout = PL_PAPER
END ACTION
```

Vor dem Drucken müssen wir die Daten aus dem PageSizeControl-Objekt noch an das PrintControl-Objekt weiterrechen. Das passiert im Handler des "Drucken"-Buttons:

```
BUTTONACTION PrintButtonHandler

  DemoPrintControl.pcDocSize = \
    DemoPageSizeControl.pageSize(PS_WIDTH), \
    DemoPageSizeControl.pageSize(PS_HEIGHT)

  DemoPrintControl.pcDocMargins = \
    DemoPageSizeControl.pagemargins(PS_LEFT_MARGIN), \
    DemoPageSizeControl.pagemargins(PS_TOP_MARGIN), \
    DemoPageSizeControl.pagemargins(PS_RIGHT_MARGIN), \
    DemoPageSizeControl.pagemargins(PS_BOTTOM_MARGIN)

  DemoPrintControl.pcLayout = DemoPageSizeControl.pagelayout

  DemoPrintControl.printJobName$ = "PageSizeControl Demo"
  DemoPrintControl.InitiatePrint
END ACTION
```

ShowDialog

Wenn Sie das PageSizeControl-Objekt in den generic Tree einbinden erzeugt es, wie bei einem Dialog, einen Button, mit dem man die Dialogbox des Objekts aufrufen kann. Die Methode ShowDialog wird nur benötigt, wenn Sie das PageSizeControl-Objekt von einer zweiten Stelle im Programm aus aufrufen wollen. Das kann zum Beispiel ein Tool-Button sein.

Syntax: **<obj>.ShowDialog**

Beispiel:

```
BUTTONACTION PageSizeToolButtonHandler
  DemoPageSizeControl.ShowDialog
END ACTION
```

(Leerseite)

(Leerseite)