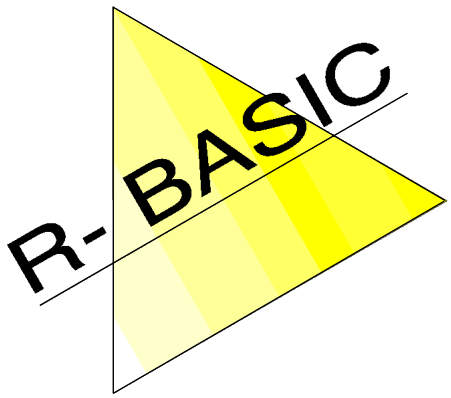
R-BASIC

Einfach unter PC/GEOS programmieren



Spezielle Themen

Volume 1
Zahlenformatierung, Schriften, Blockgrafik,
Hilfedateien, Zwischenablage,
Dateisystem, Pfade, Ordner

Version 1.0

(Leerseite)

Inhaltsverzeichnis

(Leerseite)

1 Formatierung von Zahlen

Um einzustellen, wie R-BASIC Zahlen darstellt, haben Sie drei Möglichkeiten:

1. Sie verwenden das voreingestellte Zahlenformat. Das sollte für viele Standard-anwendungen ausreichen.

2. Sie verwenden die Funktion **SetNumberFormat**, die im Abschnitt 1.1 beschrieben wird. Damit können Sie weitere typische Standardformate einstellen, wie z.B. eine feste Anzahl von Nachkommastellen.

3. Wenn Sie damit nicht auskommen haben Sie mit Hilfe der globalen Variablen **numberFormat** weitgehende Kontrolle darüber, wie R-BASIC Zahlen ausgibt. Im Abschnitt 1.2 finden Sie ausführliche Informationen dazu. Sie können z.B. vorgeben wie viele Nachkommastellen ausgegeben werden oder wann in die Exponentialdarstellung gewechselt wird.

1.1 Standard-Zahlenformate

Für viele Zwecke reichen die Standard-Zahlenformate aus, die mit der Funktion SetNumberFormat eingestellt werden können. Intern belegt SetNumberFormat die globale Variable **numberFormat** mit vorgegebenen Werten.

SetNumberFormat

Stellt das Zahlenformat für die Anzeige von Zahlen ein. Die verfügbaren Zahlenformate finden Sie in der Tabelle auf der nächsten Seite.

Syntax 1: **SetNumberFormat** **format**
Die Systemvariable **numberFormat** wird belegt.
Das neue Zahlenformat wird sofort wirksam.
format: einzustellendes Zahlenformat (ein numerischer Wert)

```
Beispiele:   SetNumberFormat  NF_NORMAL
             SetNumberFormat ( NF_SCI_4 )   ' Klammern sind OK
```

Weitere Beispiele finden Sie im Abschnitt 1.3 (Komplexe Beispiele)

Für 'format' stehen folgende Werte zur Verfügung:

Wert	Bezeichnung	Wirkung
0	NF_NORMAL	Standardeinstellung von R-BASIC. Genauigkeit: 5 Stellen. Zahlen unter 0.0001 und über 9999999 werden im Exponentialformat dargestellt.
1	NF_CURRENCY	Währungs-typische Darstellung mit 2 Nachkommastellen (gerundet). Zahlen unter 0.005 werden als Null dargestellt(1)
2	NF_MAX_PREC	Maximale Genauigkeit mit 15 Stellen Darstellung immer im Exponentialformat (2)
3	NF_SCI_3	Wissenschaftliche Darstellung (3) mit 3 Stellen z.B. 34.8E+06
4	NF_SCI_4	Wissenschaftliche Darstellung (3) mit 4 Stellen z.B. 34.82E+06
5	NF_INTEGER	Ganzzahlige Darstellung
6	NF_FIXED_3	Immer 3 Nachkommastellen, auch ein Exponentialdarstellung z.B. 123.456 oder 4.567E+8
7	NF_FIXED_4	Immer 4 Nachkommastellen, auch ein Exponentialdarstellung z.B. 234.8765

(1) Durch die interne Zahlendarstellung kann es zu Rundungsfehlern kommen. Intern wird deswegen ein Flag gesetzt (FF_NO_EXP_LOW, siehe Kapitel 1.2) dass bewirkt, dass Werte sehr nahe an Null (z.B. 1E-07) als Null angezeigt werden.

(2) Durch die feste Struktur der Zahl (als Text) gut als Ausgangspunkt für eigene Zahlenformate geeignet.

(3) Wissenschaftliche Darstellung heißt, das der Exponent immer durch 3 teilbar ist.

Für Fortgeschrittene:

Sie können SetNumberFormat auch benutzen um eine Variable des Typs NumberFormatStruct für die spätere Verwendung vorzubereiten.

Syntax 2: **<nf>** = **SetNumberFormat** (**format**)
<nf> Variable vom Typ **NumberFormatStruct**. Die Systemvariable **numberFormat** wird nicht geändert, d.h. das eingestellte Zahlenformat ändert sich nicht. Stattdessen wird die Variable nf mit den von **SetNumberFormat** gelieferten Daten belegt.

1.2 Die numberFormat Variable

Wenn Sie mit den Standard Zahlenformaten nicht auskommen haben Sie mit Hilfe der globalen Variablen **numberFormat** weitgehende Kontrolle darüber, wie R-BASIC Zahlen ausgibt. Das betrifft sowohl die PRINT-Anweisung als auch die Konvertierungsfunktionen **Str$**() und **StrLocal$**(). In diesem Abschnitt finden Sie ausführliche Erläuterungen zur **numberFormat**-Variablen und dem dazugehörigen Typ **NumberFormatStruct,** damit Sie in der Lage sind, die **numberFormat**-Variable manuell Ihren Bedürfnissen anzupassen.

1.2.1 Überblick

Die globale Variable **numberFormat** ist vom Typ **NumberFormatStruct**. In diesem Abschnitt erhalten Sie einen Überblick, eine nähere Beschreibung der Felder finden Sie in den folgenden Unterkapiteln dieses Abschnitts.

```
STRUCT NumberFormatStruct
    minDigits, maxDigits, digitMode    AS   Integer
    highLimit, lowLimit                AS   Integer
    plusSign                           AS   Integer
    exponentMode                       AS   Integer
    preChars, addChars                 AS   String[7]
    formatFlags                        AS   Word
    END STRUCT
```

Alle Felder lassen sich lesen und schreiben.

Bedeutung der einzelnen Felder:

minDigits, maxDigits, digitMode
Diese Felder bestimmen, mit wie vielen Stellen Zahlen dargestellt werden. Details dazu finden Sie im Abschnitt 1.2.2.

highLimit, lowLimit, exponentMode
Diese Felder bestimmen wann in die Exponentialdarstellung gewechselt wird, also z.B. 1.234567E+05 statt 123456.7 ausgegeben wird und ob der Exponent in 3er-Schritten angezeigt wird. Details dazu finden Sie im Abschnitt 1.2.3.

plusSign Dieses Feld bestimmt ob bei positiven Zahlen ein '+' geschrieben werden soll oder nicht. Negative Vorzeichen werden immer geschrieben. plusSign kann sein:
PS_NONE (0, kein Plus schreiben z.B. 12.567)
PS_SPACE (1, statt Plus ein Leerzeichen schreiben z.B. 12.567)
PS_ALWAYS (2, Plus immer schreiben z.B. +12.567)
Der Standard ist PS_SPACE.

preChars, addChars

Enthalten jeweils bis zu 7 Zeichen, die vor (preChars) oder nach (addChars) der Zahl angefügt werden. Das ist z.B. bei Währungsangaben oder für Sperrzeichen sinnvoll. Standardmäßig sind diese Felder leer.

Beispiel: **numberFormat**.preChars = "---- "
numberFormat.addChars = " Euro"
PRINT 123,87
Auf dem Bildschirm erscheint --- 123.87 Euro

formatFlags

Formatflags enthält einige Werte, die das Verhalten der Zahlenausgabe modifizieren können. Die Werte sind so gewählt, dass jeweils einzelne Bits gesetzt sind (sog. Bit-Flags), so dass sie mit OR, AND und NOT, aber auch mit Plus (+) verbunden werden können. Folgende Werte sind definiert, die anderen Bits sind reserviert. Der Standard ist, dass kein Bit gesetzt ist (formatFlags = 0).

FF_PRINT_ADD_NO_SPACE (= 1) bewirkt, dass Print kein zusätzliches Leerzeichen an die Zahlen hängt. Normalerweise wird ein zusätzliches zu den in **addChars** angegebenen Zeichen ein weiteres Leerzeichen angehängt, um die Lesbarkeit zu verbessern. Str$ und StrLocal$ hängen grundsätzlich kein Leerzeichen an.

FF_NO_EXP_LOW (= 2) bewirkt das bei Zahlen, deren Betrag zwischen Null und 1 liegt, nicht in die Exponentialdarstellung gewechselt wird. Das ist sinnvoll z.B. bei Währungsangaben.
Hinweis 1: Der Wert des Feldes **lowLimit**, der sonst festlegt, wann in die Exponentialdarstellung gewechselt wird, wird ignoriert, statt dessen wird auf die geforderte Stellenzahl gerundet.
Hinweis 2: Dieses Flag ist im digitMode DM_VALID_DIGITS nicht verwendbar.

FF_NO_EXP_HIGH (= 4) bewirkt, dass bei großen Zahlen nicht in die Exponentialdarstellung gewechselt wird.
Hinweis: Der Wert des Feldes **highLimit**, der sonst festlegt, wann in die Exponentialdarstellung gewechselt wird, wird ignoriert. Zahlen, die sich nicht mehr darstellen lassen, werden z.B. als "#####.###" angezeigt. Das ist z.B. bei der Fehlersuche sinnvoll.

Beispiel: Immer eine Nachkommastelle erzwingen:

```
numberFormat.digitMode = DM_FRAC_DIGITS
                              Nur Nachkommastellen zählen
numberFormat.minDigits = 1    Mindestens eine Nachkommastelle
numberFormat.maxDigits = 1    höchstens eine Nachkommastelle
numberFormat.highLimit = 15   Nur im Notfall (zu viele Stellen)
                              ins Exponentialformat wechseln
numberFormat.formatFlags = FF_NO_EXP_LOW
                              Zahlen nahe Null: runden
```

Wegen der vielen Möglichkeiten, die Sie hier haben, ist das korrekte Belegen der Systemvariablen numberFormat unter Umständen unübersichtlich. Deswegen gibt es im Ordner Beispiel\Mathe zwei Beispielprogramme.
Das Beispiel "**Zahlenformatierung**" gibt ein paar Zahlen in verschiedenen Standardformaten aus.
Das Beispiel "**NumberFormat Einstellungen**" erlaubt das interaktive Belegen der Systemvariablen **numberFormat** und Sie können so direkt die Auswirkungen von bestimmten Einstellungen sehen.

1.2.2 Einstellen der Stellenzahl

Die Felder **minDigits**, **maxDigits**, **digitMode** bestimmen, mit wie vielen Stellen eine Zahl dargestellt wird. Wenn nicht anders angegeben, wird in den folgenden Beispielen vorausgesetzt, dass das Standardformat (NF_NORMAL) voreingestellt ist.

Die Standardeinstellungen von R-BASIC sind: minDigits = 1
maxDigits = 5
digitMode = DM_VALID_DIGITS

Beispiele:	1/3	⇒	0.33333	
	0.1	⇒	0.1	
	1001.2	⇒	1001.2	
	123456.7	⇒	123457	(gerundet, aber 6 Stellen weil Vorkommastellen nie gerundet werden)

minDigits bestimmt die minimale Stellenzahl, die ausgegeben werden soll. Hat die Zahl weniger Stellen so werden nach dem Komma Nullen angehängt.
Beispiel: 12.34 wird als 12.340 ausgegeben

maxDigits bestimmt die maximale Stellenzahl, die ausgegeben werden soll. Hat die Zahl mehr Stellen, so wird gerundet.
Beispiel für maxDigits = 5: 10/3 wird als 3.3333 ausgegeben

Hinweis: Vorkommastellen werden nicht gerundet. Nur Nachkommastellen werden gerundet.
Beispiel: 123456.78 wird mit 6 Stellen als 123457 ausgegeben

digitMode bestimmt, welche Stellen für minDigits und maxDigits zählen. digitMode kann sein:

- DM_ALL_DIGITS (Wert 0, alle Stellen zählen),
- DM_FRAC_DIGITS (Wert 1, nur Nachkommastellen zählen)
- DM_VALID_DIGITS (Wert 2, gültige Stellen zählen)

Beispiel: Wir wollen minimal 3 und maximal 5 Stellen mit verschiedenen Werten von digitMode ausgeben.

```
SetNumberFormat(NF_NORMAL)      ' setzt schon maxDigits = 5
numberFormat.minDigits = 3
numberFormat.digitMode = ....
```

digitMode = DM_ALL_DIGITS (alle Stellen zählen)
Es macht keinen Unterschied, ob die Ziffern vor oder nach dem Komma stehen, so wie es keinen Unterschied macht, ob man 1,230 km oder 1230 m schreibt.

Beispiele:	1/3	⇒	0.3333	(5 Stellen insgesamt)
	0.1	⇒	0.10	(mind. 3 Stellen insgesamt)

1001.27 ⇒ 1001.3 (max. 5 Stellen insgesamt)

digitMode = DM_FRAC_DIGITS (nur Nachkommastellen zählen)
Es werden immer Nachkommastellen geschrieben. Maximal maxDigits, mindestens jedoch minDigits Nachkommastellen.

Beispiele:	1/3	⇒	0.33333	(5 Nachkommastellen)
	0.1	⇒	0.100	(mind. 3 Nachkommastellen)
	1001.27	⇒	1001.270	(mind. 3 Nachkommastellen)

digitMode = DM_VALID_DIGITS (gültige Stellen zählen)
Führende Nullen werden ignoriert, die Zählung der Stellenzahl beginnt bei der ersten von Null verschiedenen Ziffer, egal wo das Komma steht.
Dies ist auch die Art und Weise, wie wir intuitiv Zahlen aufschreiben. Die Körpergröße einer Person wird als 1,73 m oder als 173 cm angegeben, niemals als 173,00 cm, weil die Angabe auf 1/10 mm hier sinnlos ist.
Die Länge einer Brücke können wir auf den Meter genau angeben, indem wir entweder 0.346 km oder 346 m schreiben.

Beispiele:	1/3	⇒	0.33333	(5 gültige Ziffern)
	0.1	⇒	0.100	(3 gültige Ziffern)
	1001.27	⇒	1001.3	(5 gültige Ziffern)

Beispiel: Wir wollen immer 3 Nachkommastellen ausgeben. Dazu verwenden wir **minDigits = maxDigits**.

```
SetNumberFormat(NF_NORMAL)
numberFormat.minDigits = 3
numberFormat.maxDigits = 3
numberFormat.digitMode = DM_FRAC_DIGITS
```

Beispiele:	1/3	⇒	0.333
	0.1	⇒	0.100
	1001.27	⇒	1001.270

Zugehöriges Formatflag: FF_NO_EXP_LOW
Mit dem Flag FF_NO_EXP_LOW im Feld formatFlags können Sie bewirken, dass bei Zahlen unter 1 (wenn digitMode DM_FRAC_DIGITS oder DM_ALL_DIGITS ist) nicht in den Exponentialmodus gewechselt wird, sondern stattdessen Null angezeigt wird. Das ist z.B. sinnvoll für Währungsangaben, wenn sich durch Rundungsfehler ein Betrag von beispielsweise 0.00000000001 Euro ergibt. Im Kapitel 1.2.3 finden Sie entsprechende Beispiele.
Im digitMode DM_VALID_DIGITS wird bei Zahlen unter 1 immer in den Exponentialmodus gewechselt, wenn sich nicht mehr alle Stellen anzeigen lassen.

1.2.3 Zahlen in Exponentialdarstellung

Lässt sich eine Zahl nicht mehr sinnvoll in "normaler" Darstellung ausdrücken, wechselt R-BASIC in die Exponentialdarstellung. Die Felder **highLimit** und **lowLimit** der Systemvariablen numberFormat bestimmen die Grenze für diesen Übergang, das Feld **exponentMode** bestimmt die Darstellung des Exponenten.

Für die folgenden Beispiele nehmen wir an, dass das Feld formatFlags der globalen Variablen numberFormat den Wert Null hat, so dass das Wechseln ins Exponentialformat nicht verhindert wird. Das entspricht der Standardeinstellung.

highLimit gibt die maximale Anzahl der Vorkommastellen an, bevor ins Exponentialformat gewechselt wird. Erlaubt sind Werte von 0 bis 15. Der Standardwert ist highLimit = 7

Beispiele:

Bei den blau markierten Beispielen wird der Wert gerundet, weil die Standardeinstellung maximal 5 Ziffern insgesamt vorsieht.

1234.5 ⇒ 1234.5
12345.6 ⇒ 12346
123456.7 ⇒ 123457
1234567.8 ⇒ 1234568
12345678.9 ⇒ 1.2346E+07

Beispiele für unterschiedliche highLimit-Werte

```
numberFormat.highLimit = 3
```

12.3 ⇒ 12.3
123.4 ⇒ 123.4
1234.5 ⇒ 1.2345E+03
12345.6 ⇒ 1.2346E+04

```
numberFormat.highLimit = 4
```

12.3 ⇒ 12.3
123.4 ⇒ 123.4
1234.5 ⇒ 1234.5
12345.6 ⇒ 1.2346E+04

Das Umschalten in die Exponentialdarstellung kann für große Zahlen verhindert werden, wenn man das Bit FF_NO_EXP_HIGH im Feld formatFlags der globalen Variablen numberFormat setzt. Statt der Umschaltung ins Exponentialformat erfolgt dann eine Fehlerausgabe der Form ####.##

lowLimit gibt die maximale Anzahl von führenden Nullen an, einschließlich der Null vor dem Komma, bevor ins Exponentialformat gewechselt wird. Der Standardwert ist lowLimit = 4

Beispiele

0.0123	⇒	0.0123
0.00123	⇒	0.00123
0.000123	⇒	0.000123
0.0000123	⇒	1.23E-04

Beispiele für numberFormat.lowLimit = 3

0.123	⇒	0.123
0.0123	⇒	0.0123
0.00123	⇒	1.23E-03
0.000123	⇒	1.23E-04

Hinweis: Es kann auch hier sein, dass die Zahl gerundet wird.
Beispiel für maximal 4 Nachkommastellen. In der blau markierten Zeile wird die Zahl daher gerundet.

```
SetNumberFormat(NF_NORMAL)
numberFormat.maxDigits = 4
numberFormat.lowLimit = 3
numberFormat.digitMode = DM_FRAC_DIGITS
```

0.123	⇒	0.123
0.0123	⇒	0.0123
0.00123	⇒	0.0012
0.000123	⇒	1.23E-04

Das Umschalten in die Exponentialdarstellung kann für Zahlen mit einem Betrag kleiner als 1 verhindert werden, wenn man das Bit **FF_NO_EXP_LOW** im Feld formatFlags setzt. Werte, die nicht mehr dargestellt werden können, werden als Null angezeigt.
Das gilt nicht, wenn digitMode den Wert DM_VALID_DIGITS hat. Dort wird immer in der Exponentialmodus gewechselt, wenn die Zahl nicht mehr "normal" dargestellt werden kann.

Beispiel für die Wirkung von FF_NO_EXP_LOW

```
SetNumberFormat(NF_NORMAL)
numberFormat.maxDigits = 3
numberFormat.lowLimit = 1           ' Keine Null nach dem Komma
numberFormat.digitMode = DM_FRAC_DIGITS
```

FF_NO_EXP_LOW ist noch **nicht gesetzt**

0.678	⇒	0.678
0.0678	⇒	6.78E-02
0.00678	⇒	6.78E-03

```
.....
numberFormat.formatFlags = FF_NO_EXP_LOW
```

FF_NO_EXP_LOW ist jetzt **gesetzt**

0.678	⇒	0.678	
0.0678	⇒	0.068	' gerundet
0.00678	⇒	0.007	' gerundet
0.000678	⇒	0.001	' gerundet
0.0000678	⇒	0.000	

exponentMode bestimmt, in welcher Weise der Exponent dargestellt wird. Der Standard ist exponentMode = EXP_NORMAL

Erlaubte Werte sind:

EXP_NORMAL (= 0) normale Darstellung

EXP_SCI (= 1) wissenschaftliche Darstellung in 3er-Schritten
Die 12er-Reihe sieht dann so aus:

12E+00
144E+00
1.728E+03
20.736E+03
248.832E+03
2.985984E+06

EXP_FORCE (= 2) Erzwingen der Exponentialdarstellung.
Zahlen werden auch dann im Exponentialformat dargestellt, wenn dies eigentlich nicht erfoderlich ist. z.B. 1 als 1E+00
EXP_FORCE hat absoluten Vorrang, auch vor den Flags **FF_NO_EXP_LOW** und **FF_NO_EXP_HIGH**.

EXP_FORCE + EXP_SCI (= 3)
Wissenschaftliche Exponentialdarstellung erzwingen.

Beispiele

```
SetNumberFormat(NF_NORMAL)
```

1234567	⇒	1234567	
12345678	⇒	1.2346E+07	' gerundet, 5 Stellen
123456789	⇒	1.2346E+08	' gerundet, 5 Stellen
1234567890	⇒	1.2346E+09	' gerundet, 5 Stellen

```
SetNumberFormat(NF_NORMAL)
numberFormat.exponentMode = EXP_SCI
```

1234567	⇒	1234567	
12345678	⇒	12.346E+06	' gerundet, 5 Stellen
123456789	⇒	123.46E+06	' gerundet, 5 Stellen
1234567890	⇒	1.2346E+09	' gerundet, 5 Stellen

1.3 Komplexe Beispiele

Allgemeine Hinweise:

- Sie können Variablen vom Typ **NumberFormatStruct** definieren. Wenn Sie der Systemvariablen **numberFormat** eine solche Variable zuweisen (**numberFormat** = ..), werden die in der selbst definierten Variablen gespeicherten Werte aktiviert.
- Achten Sie auf konsistente Werte (z.B. **minDigits** <= **maxDigits**), wenn Sie **numberFormat** oder andere Variablen vom Typ **NumberFormatStruct** belegen. Die Ergebnisse könnten sonst unerwartet und verwirrend sein. Typische Fehler korrigiert R-BASIC selbständig ohne Fehlermeldung!
- Sie können die mit **SetNumberFormat** eingestellten Zahlenformate nachträglich modifizieren, indem sie die **numberFormat**-Variable ändern.

Beispiel 1: Zeitweise Exponentialdarstellung erzwingen. Der aktuelle Wert der numberFormat-Variable wird in einer anderen Variablen zwischengespeichert.

```
DIM nf AS NumberFormatStruct
DIM n

   nf = numberFormat                ' aktuelle Belegung merken
   numberFormat.exponentMode = EXP_FORCE
   FOR n = 1 TO  8 STEP 0.7
      Print n
   NEXT
   numberFormat = nf                ' Wieder herstellen
```

Beispiel 2: Vorbereiten eines anderen Zahlenformats in einer Variablen

```
DIM  nf  AS NumberFormatStruct
  Print 20/3                              '  6.6667
  nf  = SetNumberFormat ( NF_INTEGER ) ' erweiterte Syntax!
                           ' numberFormat wird nicht geändert
  Print 20/3                              '  6.6667
  numberFormat = nf
  Print 20/3                              '  7
```

Beispiel 3: Eine Funktion, die eine NumberFormat-Struktur bearbeitet.

```
FUNCTION SetPlusSign(nf AS NumberFormatStruct ) as
  NumberFormatStruct
    nf.plusSign = PS_ALWAYS
    RETURN nf
    END FUNCTION
```

Beispiel 4: **SetNumberFormat (NF_FIXED_4)** bewirkt, das Zahlen immer mit 4 Stellen nach dem Komma dargestellt werden, auch dort , wo es nicht nötig ist, z.B. bei 1.0000 und 0.2500
Wenn Sie die nachgestellten Nullen entfernen wollen, setzten Sie **minDigits** auf Null.

```
SetNumberFormat ( NF_FIXED_4 )
numberFormat.minDigits = 0
Print 1                          '  1
Print 1/4                        ' 0.25
Print 1/3                        ' 0.3333
```

Hinweis zu Beispiel 4: Der Standard für **maxDigits** ist 5. Es wurde aber von SetNumberFormat (NF_FIXED_4) auf den Wert 4 gesetzt.

Beispiel 5: Sie wollen Geldbeträge mit führendem Plus, einem führenden Leerzeichen und der Währung "Euro" ausgeben.

```
SetNumberFormat ( NF_CURRENCY )
numberFormat.preChars = " "
numberFormat.plusSign = PS_ALWAYS
numberFormat.addChars = " Euro"
```

Beispiel 6: Ergänzung zu Beispiel 5.
Das von Print automatisch angehängte Leerzeichen soll unterdrückt werden. Um bereits gesetzte Flags zu erhalten (**FF_NO_EXP_LOW** wird bereits von **SetNumberFormat (NF_CURRENCY)** gesetzt) wird die Kombination mit einer logischen OR-Verknüpfung verwendet.

```
  Print  307.87              '    +307.87 Euro  <-
                             '        Leerzeichen am Ende

numberFormat.formatFlags \
        = numberFormat.formatFlags OR FF_PRINT_ADD_NO_SPACE

  Print  307.87              '   +307.87 Euro <- kein
                             '        Leerzeichen mehr
```

Hinweis zu Beispiel 6: Die Variante **numberFormat.formatFlags + FF_PRINT_ADD_NO_SPACE** würde in diesem konkreten Fall ebenfalls zulässig, da man in diesem Beispiel sicher sein kann, dass **FF_PRINT_ADD_NO_SPACE** noch nicht gesetzt ist.

2 Verwendung von Schriften

2.1 Überblick

R-BASIC kann alle im GEOS-System installierten Schriften (Fonts) und Schriftstile verwenden.

FontID's
GEOS (und auch R-BASIC) identifiziert Schriften über eine Font-Identifikations-Nummer, die FontID. Näheres dazu erfahren Sie im Abschnitt 2.2

Die Font-Modi
Um die Fähigkeiten von PC/GEOS voll ausreizen zu können, kennt R-BASIC drei Arten von Schriften (Font-Modi):

- Im **Fixed-Font-Modus** (Abschnitt 2.3) hat jedes Zeichen eine feste Breite und eine feste Höhe. R-BASIC kennt und verwaltet die Position jedes einzelnen Zeichens.
- Im **GEOS-Font-Modus** (Abschnitt 2.4) übernimmt GEOS die Ausgabe des Textes. Es stehen alle im System installierten Schriften und alle Textstile (Fett, Kursiv, hochgestellt usw.) zur Verfügung.
- Im **Block-Font-Modus** (Abschnitt 2.5, ausführliche Beschreibung im Kapitel 3) wird für jeden Buchstaben eine kleine Grafik ausgegeben (z.B. 14x8 Pixel). Damit ist es möglich, sehr einfach grafische Elemente auf den Bildschirm zu bringen.

Textstile
Die Eigenschaften der Schrift (Textstile: fett, kursiv, unterstrichen usw.) werden in R-BASIC über die Systemvariable **printFont.style** eingestellt. Das Feld style ist das einzige "öffentliche" Feld der printFont-Variablen und wird im Abschnitt 2.6 beschrieben. Alle anderen Felder werden automatisch beim Einstellen des Font-Modus gesetzt.

Die **printFont**-Variable
Die Systemvariable **printFont** ist der Kern der R-BASIC-Schriftverwaltung. Zuweisungen zu dieser Variablen oder einem ihrer Felder bestimmen die von R-BASIC verwendete Schrift und ihre Eigenschaften. Fortgeschrittene Programmierer können die printFont - Variable auch direkt modifizieren. Die notwendigen Informationen dafür finden Sie im Abschnitt 2.7.

2.2 Zugriff auf GEOS-Fonts

PC/GEOS identifiziert Schriften (genannt: Fonts) über eine sogenannte Font-ID-Nummer. Diese Font-ID ist an viele Font-Routinen zu übergeben, z.B. zum Einstellen des Font-Modus (Abschnitte 2.3, 2.4, 2.5). In R-BASIC sind einige Fonts, die auf allen GEOS-Systemen installiert sind, namentlich vordefiniert.

Tabelle: Namentlich verfügbare Font-ID's in R-BASIC

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

(1)Hinweis: Die Fonts mit den ID's FID_BISON, FID_UNIVERSITY und FID_BERKELEY sind Bitmap-Fonts, die sich nicht zur Ausgabe auf den Drucker eignen.

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

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

Weitere Font-ID's bekommen Sie aus dem PC/GEOS-SDK oder mit dem FontExplorer, der über das Menü "Extras"-"Tools" erreichbar ist.

Um Informationen über die im System installierten Fonts zu erhalten, gibt es die Routinen **FontAvail**, **FontFind**, **FontGetName$** und **FontGetSysInfo**, die im Folgenden beschrieben werden.

FontAvail

FontAvail prüft, ob ein Font mit der übergeben fontID (Font-Identifikations-Nummer) im System installiert ist. Es liefert TRUE (-1, Font ist installiert) oder FALSE (0, Font ist nicht installiert).

Syntax: **<variable>** = FontAvail (**fontID**)
Parameter: <variable> ist eine numerische Variable
fontID: Identifikationsnummer des Fonts

```
Beispiele:   found = FontAvail ( FID_ROMAN )
             found = FontAvail ( 53267 )        '  Auf Animal-Font
                                                ' testen
```

FontFind

FontFind prüft, ob ein Font mit dem übergeben Namen im System installiert ist. Es liefert die Font-ID-Nummer oder 0 (Font ist nicht installiert).

Syntax: **<variable>** = **FontFind** (**"FontName"**)
Parameter: <variable> ist eine numerische Variable
"FontName": Stringausdruck, Fontname

```
Beispiel:    fontID = FontFind ( "Cartoon" )
```

FontGetName$

FontGetName$ liefert den GEOS-Namen des Fonts mit der übergeben fontID. Ist der Font nicht installiert, liefert es eine Leerstring.

Syntax: **<variable>** = **FontGetName$** (**fontID**)
Parameter: <variable> ist eine Stringvariable
fontID: Identifikationsnummer des Fonts

```
Beispiel:    nameOfRoman$ = FontGetName$ ( FID_ROMAN )
```

FontGetSysInfo

FontGetSysInfo liefert Informationen über die Fonts, die GEOS für bestimmte Zwecke verwendet, indem es die zugehörigen INI-Einträge ausliest.

Syntax: **<variable>** = **FontGetSysInfo** (**x**)
Parameter: <variable> ist eine numerische Variable
x: Welche Info wird angefordert. Siehe Tabelle.

```
Beispiel:    fontSize = FontGetSysInfo ( 7 )
```

Tabelle: Informationen, die **FontGetSysInfo** liefert:

Wert	INI-Eintrag	Information
0	[system] fontid	Default-Font, wenn kein anderer oder ein fehlerhafter angegeben wurde
1	[system] fontsize	Größe für Default Font
2	[ui] fontid	Font für Menüs und nicht editierbare Texte
3	[ui] fontSize	Größe für Menüs und nicht editierbare Texte
4	[ui] editableTextFontID	Font für editierbare Texte
5	[ui] editableTextFontsize	Größe für editierbare Texte
6	[fileManager] fontid	Font für Dateinamen im GeoManager
7	[fileManager] fontsize	Größe für Dateinamen im GeoManager

2.3 Der Fixed-Font-Modus

Der **Fixed-Font-Modus** ist die Standardeinstellung von R-BASIC. Jedes Zeichen hat eine feste Breite und eine feste Höhe. R-BASIC kennt und verwaltet die Position jedes einzelnen Zeichens. R-BASIC stehen 5 verschiedene Fonts in jeweils 5 verschiedenen Größen zur Verfügung. Es stehen einige Textstile zur Verfügung. Die Einstellung der Textstile erfolgt über das Feld **printFont.style**. Beachten Sie, dass die Textstile sog. Bit-Flags sind (d.h. bestimmte Bits haben bestimmte Bedeutung). Details dazu finden Sei im Abschnitt 2.6.

FontSetFixed

Die Anweisung **FontSetFixed** versetzt R-BASIC in den Fixed-Font Modus. Dabei wird intern die Systemvariable printFont mit den passenden Werten belegt.

Syntax: **FontSetFixed fontID**, **size** [,**lineHeight**]
fontID: ID-Nummer des Fonts. Zulässige Werte: siehe Tabelle unten.
size: Größe der Schrift in Point.
Zulässig sind die Werte 10, 12, 14, 18 und 22
lineHeight: (optional) Zeilenabstand. **FontSetFixed** stellt standardmäßig günstige Werte ein (siehe Tabelle unten). Wählen Sie den Zeilenabstand zu klein, überlappen sich die Buchstaben.

```
Beispiel 1: FontSetFixed (FID_UNIVERSITY, 18)  ' Klammern sind OK
Beispiel 2:
    ' Doppelten Zeilenabstand verwenden (vgl. Tabelle unten)
    FontSetFixed FID_UNIVERSITY, 18, 48
```

Hinweise:

- **FontSetFixed** stellt automatisch das maximale Textfenster ein, der Cursor wird nach links oben gesetzt (siehe Window-Befehl). Sie können das Fenster anschließend mit dem WINDOW-Befehl ändern und/oder den Cursor mit LOCATE positionieren.

- Verwenden Sie bei Bedarf den Befehl PRINT atXY(x,y); "Text..." um die Textausgaben pixelgenau zu positionieren.

- Im Fixed-Font-Modus wird der Texthintergrund per Default in der aktuellen Hintergrundfarbe gelöscht. Um Texte transparent auszugeben müssen Sie die Hintergrundfarbe auf BG_TRANSPARENT setzen.

```
PAPER   BG_TRANSPARENT
```

Hinweise für fortgeschrittene Programmierer:

- Die von PRINT verwendete Schriftart (Font), die Textgröße und weitere Eigenschaften werden von der Systemvariable **printFont** bestimmt, die im Abschnitte 2.7 beschrieben ist. Fortgeschrittene Programmierer können die printFont - Variable auch direkt modifizieren.

- Fortgeschrittene Programmierer finden eventuell die Syntax
  <variable> = **FontSetFixed**(fontID, size [,lineHeight])
  hilfreich. <variable> ist eine Variable vom Typ PrintFontStruct, die dann **statt** der Systemvariablen printFont belegt wird.

Tabelle: Zeichengrößen, die von **FontSetFixed** gesetzt werden. Blau kursiv bezeichnet das jeweilige printFont-Feld.

Font-ID schirmgröße	Größe *size*	Zeichen Breite *charWidth*	Zeilen-abstand *lineHeight*	Spalten x Zeilen bei Bil16 640 x 400 Pixel
FID_BISON	10	7	13	91 x 30
	12	8	16	80 x 25
	14	10	20	64 x 20
	18 ‡	12	22	53 x 18
	22 ‡	15	25	42 x 16
FID_UNIVERSITY	10	10	14	64 x 28
	12	11	16	58 x 25
	14	14	19	45 x 21
	18	16	24	40 x 16
	22	22	30	29 x 13
FID_BERKELEY	10	11	13	58 x 30
	12	14	16	45 x 25
	14	16	20	40 x 20
	18	16	23	40 x 17
	22 ‡	16	25	40 x 16
FID_MONO	10	8	12	80 x 33
	12	9	16	71 x 25
	14	10	16	64 x 25
	18	13	20	49 x 20
	22	14	25	45 x 16
FID_SANS	10	10	14	64 x 28
	12	12	16	53 x 25
	14	14	18	45 x 22
	18	16	22	40 x 18
	22	21	38	30 x 10

‡ : Bison unterstützt nicht die Größen 18 und 22 Point, Berkeley nicht 22 Point. Hier wird nur die Zeichenbox (Breite und Höhe) vergrößert.

2.4 Der GEOS-Font-Modus

Im GEOS-Font-Modus übernimmt das GEOS-System die Ausgabe des Textes. Es stehen alle im System installierten Schriften und alle Textstile (Fett, Kursiv, hochgestellt usw.) zur Verfügung. Die Einstellung der Textstile erfolgt über das Feld printFont.style (siehe Abschnitt 2.6). Beachten Sie, dass die Textstile sog. Bit-Flags sind (d.h. bestimmte Bits haben bestimmte Bedeutung).

Da die meisten GEOS-Fonts sogenannte proportional-Fonts sind, d.h. ein 'm' ist viel breiter als ein 'l', kennt R-BASIC die Position der einzelnen Buchstaben bei einer Textausgabe nicht. Die Kommandos WINDOW und LOCATE arbeiten deswegen mit einer "durchschnittlichen" Buchstabenbreite und viele Print-Steuercodes arbeiten nicht oder nur eingeschränkt.

R-BASIC hat keine Kontrolle über die genaue Position der einzelnen Zeichen innerhalb eines ausgegebenen Textstrings. Verwenden Sie bei Bedarf den Befehl PRINT atXY(x,y); "Text..." um die Textausgaben präzise zu positionieren.

FontSetGEOS

Die Anweisung **FontSetGEOS** versetzt R-BASIC in den GEOS-Font Modus. Dabei wird intern die Systemvariable printFont mit den passenden Werten belegt.

Syntax:	**FontSetGEOS** **fontID**, **size** [,**lineHeight**]	
Parameter:	fontID:	ID-Nummer des Fonts. Weitere Infos zur fontID und einige verfügbare Werte finden Sie hier.
	size:	Größe der Schrift in Point.
	lineHeight:	optional: Zeilenabstand. **FontSetGEOS** stellt standardmäßig günstige Werte ein (Zeichengröße + ca. 30%, exakt: INT(1.35*size)). Wählen Sie den Zeilenabstand zu klein, überlappen sich die Buchstaben.

```
Beispiel 1: FontSetGEOS FID_SYMBOLPS, 18
Beispiel 2   ' Zeilenabstand = 150% der Schriftgröße
    FontSetGEOS(FID_ROMAN, 14, 1.5 * 14)   ' Klammern sind erlaubt
```

Hinweise:

- **FontSetGEOS** stellt automatisch das maximale Textfenster ein, der Cursor wird nach links oben gesetzt (siehe Window-Befehl). Sie können das Fenster anschließend mit dem WINDOW-Befehl ändern und/oder den Cursor mit LOCATE positionieren.

- Die Befehle WINDOW, LOCATE und POS arbeiten mit einer "durchschnittlichen" Zeichenbreite, der Befehl VGet$ steht nicht zur Verfügung.

- Im GEOS Font Modus wird der Texthintergrund per Default in der aktuellen Hintergrundfarbe gelöscht. Um Texte transparent auszugeben müssen Sie die Hintergrundfarbe auf BG_TRANSPARENT setzen.

```
PAPER    BG_TRANSPARENT
```

Hinweise für fortgeschrittene Programmierer:

- Die von PRINT verwendete Schriftart (Font), die Textgröße und weitere Eigenschaften, z.B. der Wert für die oben erwähnte "durchschnittliche" Zeichenbreite werden von der Systemvariable printFont bestimmt, die im Abschnitte 2.7 beschrieben ist. Fortgeschrittene Programmierer können die printFont - Variable auch direkt modifizieren.

- Fortgeschrittene Programmierer finden eventuell die Syntax
  <variable> = **FontSetGEOS**(fontID, size [,lineHeight])
  hilfreich. <variable> ist eine Variable vom Typ PrintFontStruct, die dann **statt** der Systemvariablen printFont belegt wird.

2.5 Der Block-Font-Modus (Block-Grafik-Modus)

Im **Block-Font-Modus** wird statt eines GEOS-Fonts für jeden Buchstaben eine kleine, quadratische Grafik ausgegeben (z.B. 16x16 Pixel). Damit entstehen die typischen "Computer-Schriften" und es ist möglich, sehr einfach grafische Elemente auf den Bildschirm zu bringen. Es stehen keinerlei Textstile zur Verfügung.

In diesem Abschnitt finden Sie eine kurze Übersicht zum Thema Block-Font-Modus. Eine ausführliche Beschreibung der Möglichkeiten finden Sie im Kapitel 3.

FontSetBlock

Die Anweisung **FontSetBlock** versetzt R-BASIC in den Blockgrafik-Modus. Dabei wird intern die Systemvariable **printFont** mit den passenden Werten belegt. Falls Sie **FontSetBlock** erstmalig im Programm verwenden oder eine andere als die aktuell gesetzte Zeichengröße verwenden wird der Zeichengenerator-Speicher gelöscht, d.h. alle Grafikzeichen sind leer. Der Parameter "colored" bestimmt, ob die Grafikzeichen in diesem Fall als monochrom (einfarbig, Default) oder als farbig (256 Farben) behandelt werden sollen.

Syntax: **FontSetBlock** **sizex**, **sizey** [, **colored**]

sizex, sizey: Größe der Block-Grafiken in Pixel. Erlaubte Werte liegen zwischen 2 und 64 (jeweils einschließlich).

colored: Ungeladenen Zeichensatz als monochrom (FALSE, Default) oder als farbig (256 Farben) behandeln..

```
Beispiel:    FontSetBlock 16, 24
```

Hinweise:

- Eine ausführliche Beschreibung der Möglichkeiten der Block-Font-Modus (auch Block-Grafik-Modus) finden Sie im Kapitel 3.
- Sie verwenden ganz normal die Print-Anweisung, statt der Buchstaben erscheinen aber Grafik-Symbole auf dem Schirm.
- Im Block-Grafik-Modus stehen keine Textstile zur Verfügung.
- R-BASIC unterstützt das Verwenden von Grafikzeichensätzen sowohl durch den direkten Zugriff auf den Zeichengenerator (den Speicher, in dem die Grafiksymbole abgelegt sind) als auch durch die Verwendung von Dateien, die Grafiksymbole enthalten.
- Das Laden eines Zeichensatzes aus einer Datei (Befehl BlockLoad) überschreibt die durch den Parameter colored gesetzte Einstellung.
- Ein intuitives Erstellen von Grafiksymbolen ist mit dem Block-Grafik-Editor möglich, den Sie im Menü "Extras"-"Tools" finden.

2.6 Textstile

R-BASIC kann verschiedene Textstile (wie unterstrichen, fett, kursiv usw.) verwenden. Welcher Stil verwendet werden soll, wird über die Systemvariable **printFont.style** bestimmt. Je nach Font-Modus stehen unterschiedliche Stile zur Verfügung. Im Block-Font-Modus stehen keine Stile zur Verfügung.

```
Beispiel:        printFont.style = TS_BOLD
```

Für Hinweise und weitere Beispiele: siehe nächste Seite.

Achtung! Die hier dargestellten Zusammenhänge beziehen sich auf die Textausgabe mit dem PRINT Kommando. Textobjekte (Memo, InputLine) haben ihren eigenen Weg Textstile zu verwenden. Das ist im Objekt-Handbuch, Kapitel 4.10 (Text-Objekte) erklärt.

Tabelle: Textstile zur Benutzung mit printFont.style
F: Verfügbar im Fixed-Font Modus
G: Verfügbar im GEOS-Font-Modus
B: Im Block-Font-Modus ist nur TS_DONT_EXEC_CONTROLS verfügbar

Textstil	Wert	Modus	Bedeutung
TS_UNDERLINE	1	G, F	unterstrichene Schrift
TS_STRIKE_THRU	2	G	~~durchgestrichene Schrift~~
TS_SUBSCRIPT	4	G	tiefgestellte Schrift $_{\text{Schrift}}$
TS_SUPERSCRIPT	8	G	hochgestellte Schrift $^{\text{Schrift}}$
TS_ITALIC	16	G	*kursive Schrift*
TS_BOLD	32	G, F	**fette Schrift**
TS_OUTLINE	64	G	Wenn der Font sowohl Bitmap- und als auch Outline-Schrift enthält: Verwendung der Outline Schrift erzwingen
TS_CENTER	256	F	Buchstaben einzeln zentrieren (langsamer) (A) Standard bei einigen FID_-Werten im Fixed-Font-Modus
TS_ERRORLINE	1024	F	Unterstrichen mit rot gepunkteter Linie(A)
TS_DONT_EXEC_CONTROLS	32768	**B**	Kein Textstil sondern bewirkt, dass Steuerzeichen (ASCII-Code < 32) nicht ausgeführt, sondern als druckbare Zeichen behandelt werden. Sinnvoll nur für Block-Fonts.(A)

(A) Wird von R-BASIC realisiert, keine Systemfunktion.

Hinweise:

- Texte werden standardmäßig mit Hintergrund ausgegeben. Um Texte transparent auszugeben müssen Sie die Hintergrundfarbe auf BG_TRANSPARENT setzen (z.B. PAPER BG_TRANSPARENT)
- Im GEOS-Font-Modus ist standardmäßig kein Stil gesetzt.
- Die Stile sind sog. Bit-Flags, d.h. sie haben bestimmte Bits gesetzt (Flag = Flagge, in der Computertechnik oft ein bestimmtes Bit) und sollten mit OR (setzen) und AND NOT (löschen) kombiniert werden.

Man setzt einen Stil mit OR:

printFont.style = printFont.style OR **TS_UNDERLINE**

Der Stil "unterstrichen" wird gesetzt, egal ob er bereits gesetzt war oder nicht.

Man löscht einen Stil mit AND NOT:

printFont.style = **printFont.style** AND NOT **TS_UNDERLINE**

Es ist kein Fehler, wenn der Stil gar nicht gesetzt war.

Man fragt einen Stil mit AND ab:

IF **printFont.style** AND **TS_UNDERLINE** THEN ...

Man überschreibt alte Stile mit neuen so:

printFont.style = **TS_ITALIC** OR **TS_UNDERLINE**

Beide Stile werden gesetzt, printFont.style erhält den Wert 17.

Sie sollten Addition (+) von Stilen vermeiden. Je nachdem, welche Stile schon gesetzt sind, können die Ergebnisse unerwartet sein.

Beispiel:

Das Setzen von Stilen mit + ist kein Problem:

printFont.style = TS_ITALIC + TS_UNDERLINE

printFont.style ist hat jetzt den Wert 17. Schreibt man jetzt

printFont.style = printFont.style + TS_ITALIC

so erhält printFont.style den Wert 17 + 16 = 33
Das entspricht aber der Stilkombination **TS_UNDERLINE** und **TS_BOLD**.

Komplexes Beispiel:
Fetten gedruckten Text nicht mehr fett drucken (**TS_BOLD** zurücksetzen) und unterstrichen **(TS_UNDERLINE**) ausgeben:

```
printFont.style = (printFont.style  OR  TS_UNDERLINE)   AND  (NOT
  TS_BOLD)
```

Die Klammern sind notwendig, um dem Compiler mitzuteilen, in welcher Reihenfolge die logischen Operatoren abgearbeitet werden sollen.

2.7 Direkter Zugriff auf die printFont Systemvariable

Achtung!
Dieser Abschnitt richtet sich an erfahrene oder ambitionierte Programmierer. Die Kenntnis oder das Verständnis der hier dargestellten Zusammenhänge ist für die meisten Anwendungsfälle von Schriften nicht erforderlich. Lediglich das Feld **printFont.style** enthält "öffentliche" Informationen. Ändern Sie die anderen Felder, könnten zunächst unerwartete Ergebnisse auftreten. Üblicherweise verwendet man eine der Befehle **FontSetFixed**, **FontSetGEOS** und **FontSetBlock**, welche die Systemvariable printFont mit einer stimmigen printFont-Struktur belegen.

Die Systemvariable **printFont** ist folgendermaßen definiert:

```
STRUCT PrintFontStruct
  type       as   word
  fontID     as   word
  fontSize   as   word
  charWidth  as   word
  lineHeight as   word
  style      as   word ' öffentlich
  base       as   word
  END STRUCT

DIM    printFont    AS    PrintFontStruct
```

Bedeutung der einzelnen Felder

type Speichert den aktuell von PRINT verwendeten Font-Typ. Gültige Werte sind **FT_FIXED** (0, gesetzt von **FontSetFixed**), **FT_GEOS** (1, gesetzt von **FontSetGEOS**) und **FT_BLOCK** (2, gesetzt von **FontSetBlock**).

fontID Die GEOS-Font-ID für FT_FIXED und FT_GEOS. Bei ungültigen Werten wählt GEOS einen Ersatzfont, häufig die BISON-Schrift.

fontSize Die Größe der Schrift. Bitmap-Schriften (z.B. FID_BISON) unterstützen nicht alle Größen.

charWidth Breite eines Zeichens.
Für type = FT_FIXED gilt: Wert in Pixeln
Für type = FT_GEOS gilt: Wert in % von **printFont.fontSize**
Für type = FT_BLOCK gilt: Wert in Pixeln

lineHeight Zeilenabstand, in Pixeln

style Textstil für FT_GEOS und FT_FIXED: Kombination von TS_xxx-Werten. Siehe Abschnitt 2.6.

base Abstand der Text-Grundline vom oberen Rand der Zeile.

Tipps und Beispiele für Fortgeschrittene

- Sie können selbst Variablen vom Typ **PrintFontStruct** definieren und verwenden. Hier können Sie veränderte Varianten der **printFont**-Variablen zwischenspeichern und später wieder verwenden. Auswirkungen auf den PRINT-Befehl hat aber nur das Setzen des Systemvariablen **printFont**.

Beispiel: modifizierte **printFont** Variable speichern

```
DIM  newFont  AS  PrintFontStruct
  FontSetGEOS(FID_SANS, 12)      ' belegt printFont
  newFont = printFont               ' Kopie erstellen
  newFont.style = TS_STRIKE_THRU    ' noch wirkungslos

  FontSetGEOS(FID_ROMAN, 20)        ' belegt printFont
  printFont.style = TS_BOLD
  Print "Ich bin FETT, URW Roman, 20 Point"

  printFont = newFont
  Print "Ich bin Durchgestrichen, URW Sans, 12 Point"
```

- Sie können praktisch jeden GEOS-Font im Modus FT_FIXED einsetzen. Die nötigen Werte für **charWidth**, **lineHeight** und **base** erfordern aber Experimentieren. Denke Sie daran, dass der Buchstabe W und das Zeichen '_' mit TS_BOLD in das Raster passen sollte.

Beispiel: URW Roman als Fixed Font

```
FontSetFixed(FID_SANS, 14)
                      ' gültige Startwerte für printFont setzen
printFont.fontID = FID_ROMAN
printFont.charWidth = <ausprobieren>
printFont.lineHeight = <ausprobieren>
printFont.base = <ausprobieren>
```

- Das Feld '**base**' bestimmt die vertikale Position der Grundlinie der Buchstaben. Ändern Sie nur das Feld 'size', erscheinen die Buchstaben auf der alten Grundlinie, aber größer (wie in GeoWrite auch):

Beispiel: Hallo  Welt

```
FontSetGEOS(FID_CRANBROOK, 14)
Print "Hallo";
printFont.size = 35
Print " große ";
printFont.size = 14
Print "Welt"
```

- Die Zuweisung eines Wertes zur **printFont**-Variablen (implizit durch Verwendung einer der **FontSet**-Funktionen oder explizit mit einer anderen Variablen) stellt automatisch das maximale Textfenster ein.

(Leerseite)

3 Verwendung des Block-Grafik-Modus (Block-Font-Modus)

3.1 Übersicht über die Verwendung von Block-Grafiken

Aus Sicht des Computers sind Buchstaben einfach nur kleine Bilder, Grafiken, die einem bestimmten ASCII-Zeichen zugeordnet sind. Im **Blockgrafik-Modus** schreibt R-BASIC mit dem PRINT-Befehl anstelle der "systemdefinierten" Buchstaben kleine selbst definierte Grafiken mit bis zu 64x64 Pixeln. Diese Grafiken stehen in einem von R-BASIC verwalteten Speicherbereich, den man "Zeichengenerator" nennt. Damit können Sie eigene Zeichensätze definieren oder sehr einfach grafische Elemente auf den Bildschirm ausgeben, die aus einzelnen "Blöcken" (z.B. 8x14 Pixeln) bestehen. Diese Technik wird schon seit der Anfangszeit der Computer verwendet und "Block-Grafik" genannt. Mit dem Befehl **FontSetBlock** aktivieren Sie den Blockgrafik-Modus. Falls Sie **FontSetBlock** erstmalig im Programm verwenden oder eine andere als die aktuell gesetzte Zeichengröße verwenden wird der Zeichengenerator-Speicher gelöscht, d.h. alle Grafikzeichen sind nutzbar, aber leer.

R-BASIC unterstützt monochrome (einfarbig, mono = Eins, chromos = Farbe) und farbige Blockgrafikzeichen (256 Farben). Monochrome Grafikzeichen werden in der aktuellen Vordergrund/Hintergrund Farbkombination gezeichnet, wobei der Hinterfrund auch transparent sein kann (die Hintergrundfarbe ist dann auf den speziellen Wert BG_TRANSPARENT gesetzt).
Farbige Blockgrafikzeichen werden in ihren eigenen Farben gezeichnet, nur der Farbindex 255 wird durch die aktuelle Hintergundfarbe ersetzt (bzw. transparent gezeichnet).

Der einfachste Weg um Blockgrafik-Zeichen zu verwenden ist die Benutzung des **Blockgrafik-Editors**, den Sie im Menü "Extras"-"Tools" finden. Der Blockgrafik-Editor erlaubt das intuitive Erstellen von Grafikzeichen und schreibt sie in eine Datei (*.RBF), die vom Befehl **BlockLoad** in den Zeichengenerator gelesen werden kann.
Zur Verwaltung der Blockgrafik-Dateien stehen weiterhin die Befehle **BlockSave**, **BlockSize**, **BlockInfo** zur Verfügung.

Sie haben einen direkten Zugriff auf den Zeichengenerator durch die Verwendung der Befehle **BlockPoke** (Schreiben eines einzelnen Bytes in den Zeichengenerator) **BlockPeek** (Lesen eines einzelnen Bytes aus dem Zeichengenerator) **BlockREAD** (Lesen eines oder mehrerer Zeichen aus DATA-Zeilen. Die DATA-Werte werden in den Zeichengenerator kopiert).

BlockSelect schaltet zwischen zwei Zeichengeneratoren um.

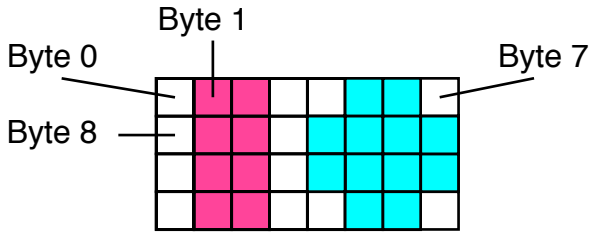
Wichtig! Die Print-Anweisung erstellt eine Kopie des Zeichens (aus dem Zeichengenerator) auf dem Bildschirm. Eine nachträgliche Änderung des Zeichengenerators wirkt sich daher nicht auf den Bildschirm aus, wird aber bei der nächsten Print-Anweisung berücksichtigt.

3.2 Interner Aufbau eines Zeichens im Blockgrafik Modus

Je nachdem, ob die Blockgrafik monochrome oder farbige Zeichen enthält unterscheidet sich die internen Struktur des Zeichens etwas.

Farbige Zeichen

In diesem Modus besteht jedes Zeichen aus einer kleinen Farbgrafik mit 256 Farben aus der Systempalette. Eigene Paletten werden nicht unterstützt. Für jedes Pixel wird ein Byte benötigt. Das heißt, ein Zeichen der Größe 8x10 Pixel benötigt 80 Bytes. Die Zählung der Bytes beginnt immer bei Null und erfolgt zeilenweise, d.h. zuerst von **links nach rechts**. Soll ein Pixel transparent (oder in in der aktuellen Hintergrundfarbe) dargestellt werden so muss der Farbindex (Farbwert) 255 verwendet werden. Zu diesem Farbwert gehört die Farbe Weiß. Weiß hat aber auch den Index 16, so dass durch diese Wahl keine Einschränkungen entstehen. Beispiel:

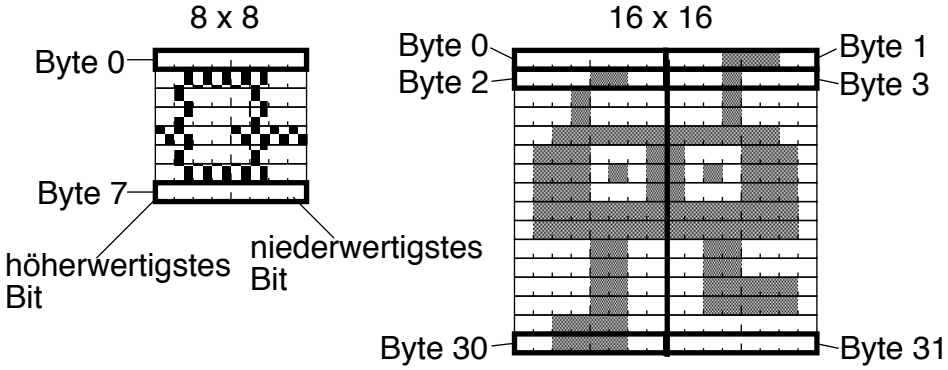


Das im Bild oben dargestellte Zeichen der Größe 8x4 Pixel wird durch folgende Bytes beschrieben (Rot: 12, Blau: 9, transparent: 255):

```
255, 12,  12,  255, 255, 9,   9,   255
255, 12,  12,  255, 9,   9,   9,   9
255, 12,  12,  255, 9,   9,   9,   9
255, 12,  12,  255, 255, 9,   9,   255
```

Monochrome Zeichen

In diesem Modus besteht jedes Zeichen aus einer kleinen Monochrom-Grafik. Dabei kann jedes Pixel die Vordergrund- oder die Hintergrundfarbe annehmen. Für ein 8x8 Zeichen benötigt man 8 Byte, für ein 16x16 Zeichen bereits 32 Byte. Die Zählung der Bytes beginnt immer bei Null und erfolgt zeilenweise, d.h. zuerst von **links nach rechts**.



Den zu einem Byte gehörenden Zahlenwert erhält man, wenn man die gesetzten Bits (in der oberen Darstellung grau) entsprechend folgender Zuordnung addiert:

128	64	32	16	8	4	2	1

Für die ersten beiden Zeilen (Byte 0 bis 3) im Bild oben rechts (16x16 Pixel) stellt sich das so dar:

128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1
128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1

woraus sich ergibt:
Byte 0: 0
Byte 1: 16 + 8 + 4 = 28
Byte 2: 8 + 4 = 12
Byte 3: 16

Ganz anlog kann man sich die Werte für das 8x8 Grafikzeichen oben links ermitteln. Es setzt sich aus den folgenden Werten 8 zusammen:

0, 124, 68, 68, 207, 68, 124, 0

oder gleichbedeutend hexadezimal:
&h00, &h7C, &h44, &h44, &hCF, &h44, &h7C, &h00

Beispiel:
Der folgende Code schaltet in den Blockgrafik-Modus. Mini8x8.rbf ist eine Block-Font-Datei, die gemeinsam mit R-BASIC installiert wurde. Dann liest es das oben dargestellte 8x8 Zeichen in den Zeichengenerator auf die Position des Zeichens 'b'. Der Befehl
PRINT "b"
schreibt dann das Grafikzeichen auf den Schirm.

```
LABEL  ZG
DATA  0, 124, 68, 68, 207, 68, 124, 0

FontSetBlock   8, 8
BlockLoad  "mini8x8.rbf" , 0, 256
Restore ZG
BlockREAD  ASC("b"), 1
Color 7, 0                    ' Grau auf Schwarz
Print "aabbbb"                ' aa
Print INK(RED); "aabbcc"      ' aa cc
Print COLOR(WHITE, RED); "bb" '
```

3.3 Aufrufen des Blockgrafik Modus

FontSetBlock

Die Anweisung **FontSetBlock** versetzt R-BASIC in den Blockgrafik-Modus. Dabei wird intern die Systemvariable **printFont** mit den passenden Werten belegt. Falls Sie **FontSetBlock** erstmalig im Programm verwenden oder eine andere als die aktuell gesetzte Zeichengröße verwenden wird der Zeichengenerator-Speicher gelöscht, d.h. alle Grafikzeichen sind leer. Der Parameter "colored" bestimmt, ob die Grafikzeichen in diesem Fall als monochrom (einfarbig, Default) oder als farbig (256 Farben) behandelt werden sollen.

Syntax: **FontSetBlock** **sizex**, **sizey** [, **colored**]

sizex, sizey:	Größe der Block-Grafiken in Pixel. Erlaubte Werte liegen zwischen 2 und 64 (jeweils einschließlich).
colored:	Ungeladenen Zeichensatz als monochrom (FALSE, Default) oder als farbig (256 Farben) behandlen..

```
Beispiel:    FontSetBlock  16, 24
```

Hinweise:

- Im Block-Grafik-Modus stehen keine Textstile zur Verfügung.
- Sie können das Flagbit TS_DONT_EXEC_CONTROLS im Feld printFont.style verwenden um die Steuerzeichen mit den ASCII-Codes unter 32 als druckbare Zeichen auszugeben statt sie "auszuführen".
- Ein intuitives Erstellen von Grafiksymbolen ist mit dem Block-Grafik-Editor möglich, den Sie im Menü "Extras"-"Tools" finden.
- FontSetBlock stellt automatisch das maximale Textfenster ein, der Cursor wird nach links oben gesetzt (siehe Window-Befehl). Sie können das Fenster anschließend mit dem WINDOW-Befehl ändern und/oder den Cursor mit LOCATE positionieren.
- Verwenden Sie bei Bedarf den Befehl PRINT atXY(x,y); "Text..." um die Textausgaben pixelgenau zu positionieren.
- Falls Sie FontSetBlock erstmalig im Programm verwenden wird der Zeichengenerator-Speicher gelöscht, d.h. alle Grafikzeichen sind leer. Das passiert auch, wenn die Größe der Zeichen (Parameter sizex oder sizey) seit dem letzten FontSetBlock geändert wird.

  Im Umkehrschluss bedeutet das, das im folgenden Code beim zweiten FontSetBlock(8, 8) der Zeichensatz nicht neu geladen wird, sondern der mit BlockLoad geladenen Zeichensatz wieder verwendet wird. Der Parameter "colored" wird in diesem Fall (d.h. wenn die Größe nicht geändert wird) ignoriert.

```
FontSetBlock (8, 8)              ' Klammern sind erlaubt
BlockLoad "MYFONT.RBF", 0, 256   ' farbiger Zeichensatz erlaubt
<.. diverse Befehle ..>
FontSetFixed (FID_UNIVERSITY, 18)
<.. diverse Befehle ..>
FontSetBlock (8, 8)                ' wieder "MYFONT" einstellen
                                   ' auch wenn MYFONT ein
                                   ' farbiger Zeichensatz ist
```

- Das Laden eines Zeichensatzes aus einer Datei (Befehl BlockLoad) überschreibt die durch den Parameter colored gesetzte Einstellung.

Hinweise für fortgeschrittene Programmierer:

- Prinzipiell ist es möglich, die **printFont** - Variable direkt zu modifizieren. Die meisten Felder haben im Blockgrafik-Modus jedoch keine Bedeutung.
- Fortgeschrittene Programmierer finden eventuell die Syntax

  <variable> = **FontSetBlock**(sizex, sizey)

  hilfreich. <variable> ist eine Variable vom Typ **PrintFontStruct**, die dann **statt** der Systemvariablen **printFont** belegt wird.

3.4 Direkter Zeichengenerator-Zugriff

Die im Blockgrafik-Modus ausgegebenen Grafiken stehen in einem von R-BASIC verwalteten Speicherbereich, den man "Zeichengenerator" nennt. Mit den folgenden Befehlen haben Sie direkten Zugriff auf den Zeichengenerator. Zum Verständnis der Arbeitsweise der Befehle finden Sie im Abschnitt 3.2 Informationen zum internen Aufbau eines Zeichens im Blockgrafik-Modus, d.h. der Organisation des Zeichengenerators.

BlockPoke

Schreibt ein einzelnes Byte in den Zeichengenerator. Die Änderungen werden beim nächsten "Print" des entsprechenden Zeichens berücksichtigt.

Syntax: **BlockPoke** **zeichen**, **byteNr**, **wert**
zeichen ASCII-Code des zu ändernden Zeichens
byteNr zu änderndes Byte. Zulässige Werte:
z.B. bei 8 x 8 - Zeichen: 0 .. 7
bei 16 x 16 - Zeichen: 0 .. 32
bei 32 x 32 - Zeichen: 0 .. 128
Hinweis: In x-Richtung gibt es evt. nicht genutzte Bits!
z.B. bei 20 x 30 - Zeichen 4 Bits --> byteNr ist von 0 .. 90 erlaubt

---

Beispiel:

```
! Das Zeichen 192 soll als schwarzer Block erscheinen
! 16 x 16 - Zeichen vorausgesetzt
FOR n = 0 TO 31
      BlockPoke   192, n , 255
      NEXT
Print Chr$(192)
```

BlockPeek

Liest ein einzelnes Byte aus dem Zeichengenerator.

Syntax: **<numVar>** = **BlockPeek** (**zeichen**, **byteNr**)
zeichen ASCII-Code des zu lesenden Zeichens
byteNr zu lesendes Byte.
Erlaubte Werte: siehe **BlockPoke**

Beispiel:

```
! Lesen der Daten des Zeichens 192
! 8 x 8 - Zeichen vorausgesetzt
DIM n, x as REAL
FOR n = 0 TO 7
     x = BlockPeek ( 192, n )
     Print x; Hex$(x)
     NEXT
```

BlockREAD

Liest ein oder mehrere ganze Zeichen aus DATA-Zeilen in den Zeichengenerator. Die Änderungen werden beim nächsten "Print" des entsprechenden Zeichens berücksichtigt.

Syntax:	**BlockREAD** **zeichen**, **anzahl**
zeichen	ASCII-Code des ersten zu ändernden Zeichens
anzahl	Anzahl der zu lesenden Zeichen. Pro Zeichen werden so viele Werte gelesen, wie entsprechend der gesetzten Größe nötig sind.

Beispiel:

```
! Statt des Zeichens "b" soll ein Muster erscheinen, das in
einer DATA-Zeile definiert ist.
!
LABEL  ZG
DATA  0, 124, 68, 68, 207, 68, 124, 0

Restore ZG
FontSetBlock  8, 8
BlockLoad  "mini8x8.rbf" , 0, 256
BlockREAD  ASC("b"), 1
Print "abbc"                    ! es erscheint a□□c
```

BlockSelect

R-BASIC verwaltet zwei Zeichengeneratoren. Damit ist ein schnelles Umschalten zwischen zwei Grafikzeichensätzen oder zwischen Text- und Grafikzeichen im Blockgrafik-Modus möglich. Standardmäßig ist Zeichengenerator 0 aktiv.

Syntax:	**BlockSelect** (**nr**)
nr	Nummer des Zeichengenerators erlaubte Werte: 0 und 1

Hinweise:

- Alle **Block~** Befehle wirken immer auf den aktuellen Zeichengenerator.
- Wird erstmalig in den alternativen Zeichengenerator gewechselt, so wird der aktuelle Zeichensatz dorthin kopiert.
- Wird mit **FontSetBlock**() eine andere Größe eingestellt (Parameter sizex oder sizey), so wird auf Zeichensatz 0 gewechselt und der alternative Zeichengenerator zurückgesetzt. Im Umkehrschluss bedeutet das, bei einem **FontSetBlock**() mit der gleichen Größe der Zeichensatz nicht verändert wird, auch wenn zwischenzeitlich z.B. ein **FontSetFixed** erfolgte.
- Es ist zulässig, dass in einem der Zeichengenaratoren ein monochromer Zeichensatz geladen ist, im anderen ein farbiger.

3.5 Zugriff auf RBF-Dateien

Der einfachste Weg um Blockgrafik-Zeichen zu verwenden ist die Benutzung des Blockgrafik-Editors, den Sie im Menü "Extras"-"Tools" finden. Der Blockgrafik-Editor erlaubt das intuitive Erstellen von Grafikzeichen und schreibt sie in eine Datei (*.RBF). In diesem Abschnitt finden Sie die Befehle, die mit diesen R-BASIC-Blockgrafik-Font-Dateien arbeiten. Informationen zur Organisation des Zeichengenerators finden Sie im Abschnitt 3.2.

BlockLoad

Lädt ein oder mehrere Zeichen aus einer RBF-Datei in den Zeichengenerator. Die RBF-Datei befindet sich üblicherweise im Ordner "USERDATA\R-BASIC\Font". Sie können jedoch mit dem Parameter "local" festlegen, dass sie im aktuellen Ordner zu finden ist.

Syntax:	**BlockLoad** **fileName$**, **firstChar**, **count** [, **local**]
fileName$	Name der Blockgrafikdatei, z.B. "MYFONT.RBF"
firstChar	ASCII-Code des ersten zu lesenden Zeichens
count	Anzahl der zu lesenden Zeichen Es muss gelten: firstChar + count <= 256
local	optional: TRUE oder FALSE (Default: FALSE) Suchen der Blockgrafikdatei im Ordner "USERDATA\R-BASIC\Font" (local = FALSE, Default-Einstellung) oder im aktuellen Ordner (local = TRUE).

BlockLoad ignoriert die von FontSetBlock vorgegebene Farbtiefe und benutzt den Wert entsprechend der RBF-Datei. Insbesondere ist es zulässig in den einen Zeichengenerator einen monochromen Zeichensatz und in den anderen Zeichengenerator einen farbigen Zeichensatz zu laden, solange die Zeichengröße übereinstimmt. Das Mischen von monochromen und farbigen Zeichen im gleichen Zeichensatz ist nicht möglich.

Fehlerbehandlung:

- Enthält die Datei keinen passenden Zeichensatz oder tritt ein anderer Fehler auf (z.B. Datei nicht gefunden), so wird der Zeichengenerator nicht geändert.
- Die globale Variable **fileError** wird belegt, im Erfolgsfall mit Null, sonst mit einer Fehlernummer. Ist die Datei keine gültige RBF-Datei so wird **fileError** auf den Wert -12 (INVALID_FONT_FILE) gesetzt, stimmt die Größe der Grafikzeichen in der Datei nicht mir der aktuell eingestellten Größe überein, so wird **fileError** auf den Wert -13 (FONT_SIZE_MISMATCH) gesetzt.

Beispiel:

```
! Lesen der oberen 128 Zeichen aus einer Datei im Ordner
"USERDATA\R-BASIC\Font"
BlockLoad "MYFONT.RBF", 128, 128
! Lesen der Zeichen a-f aus einer Datei im aktuellen Ordner
BlockLoad  "NEWFONT.RBF",  ASC("a"), 6, TRUE
```

BlockSave

Speichert einen Zeichensatz in eine RBF-Datei. Existiert die Datei bereits, wird sie ohne Meldung überschrieben. Welche der ASCII-Zeichen in die Datei geschrieben werden hängt vom Parameter "saveBits" ab.

Syntax: **BlockSave fileName$, local, saveBits**

fileName$	Name der Blockgrafikdatei, z.B. "MYFONT.RBF"
local	TRUE oder FALSE Datei ins aktuelle Verzeichnis (TRUE) oder nach "USERDATA\R-BASIC\Font" (FALSE) schreiben.
saveBits	16 Bit Wert, der bestimmt, welche Zeichen in die Datei zu schreiben sind. Jedes Bit steht für einen Block von 16 Zeichen. Siehe Tabelle unten.

Fehlerbehandlung:

- Die globale Variable **fileError** wird belegt, im Erfolgsfall mit Null, sonst mit einer Fehlernummer (z.B. Datei in Benutzung).

Verwendung des Parameters saveBits:

RBF-Dateien enthalten meist nicht für jeden ASCII-Code ein Grafikzeichen. Der Parameter saveBit legt jeweils für eine Gruppe von 16 ASCII-Codes fest, ob sie in die Datei geschrieben werden sollen oder nicht. Bit 0 steht dabei für die Codes 0 bis 15, Bit 1 für 16 bis 31 usw. Die folgenden Tabelle enthält die entsprechenden Werte für die einzelnen Gruppen. Den Wert für SaveBits erhält man, indem man die zu den entsprechenden Bits gehörenden Werte (rot markiert) addiert.

Bit Nr.	Wert dez.	Wert hex.	ASCII-Codes (dezimal)	Bit Nr.	Wert dez.	Wert hex.	ASCII-Codes (dezimal)
0	1	&h1	0 ... 15	8	256	&h100	128 ... 143
1	2	&h2	16 ... 31	9	512	&h200	144 ... 159
2	4	&h4	32 ... 47	10	1024	&h400	160 ... 175
3	8	&h8	48 ... 63	11	2048	&h800	176 ... 191
4	16	&h10	64 ... 79	12	4096	&h1000	192 ... 207
5	32	&h20	80 ... 95	13	8192	&h2000	208 ... 223
6	64	&h40	96 ... 111	14	16384	&h4000	224 ... 239
7	128	&h80	112 ... 127	15	32768	&h8000	240 ... 255

Beispiele für typische Bereiche	zugehöriger Wert	
Zahlen und Sonderzeichen (ASCII Codes 32 ... 63)	12	(= 4 + 8)
Großbuchstaben (ASCII Codes 64 ... 95)	48	(= 16+32)
Alle ASCII-Zeichen von 32 (Leerzeichen) bis 127	252	(= &hFC)
Alle ASCII-Zeichen außer den Steuerzeichen (0...31)	65532	(= &hFFFC)

BlockSize

Liest die Größe der Grafikzeichen aus einer Datei. Der Rückgabewert enthält sowohl die Höhe als auch die Breite und berechnet sich zu **256*breite + höhe**.

Ermittlung der Höhe / Breite aus dem Rückgabewert x:

breite = ShR(x,8) AND 255	oder :	breite = INT (x / 256)
höhe = x AND 255		höhe = x - 256 * breite

Syntax:	**<numVar>** = **BlockSize** (**fileName$** [, **local**])
fileName$	Name der Blockgrafikdatei, z.B. "MYFONT.RBF"
local	optional: TRUE oder FALSE (Default: FALSE) Datei im aktuellen Verzeichnis (TRUE) oder in "USERDATA\R-BASIC\Font" (FALSE, Default) suchen.

Fehlerbehandlung:

- Die globale Variable **fileError** wird belegt, im Erfolgsfall mit Null, sonst mit einer Fehlernummer (z.B. Datei nicht gefunden). Konnte die Datei gefunden werden, ist aber keine gültige RBF-Datei, so wird **fileError** auf den Wert -12 (INVALID_FONT_FILE) gesetzt.
- Im Fehlerfall (z.B. Datei nicht gefunden oder keine gültige RBF-Datei) liefert die Funktion den Wert 0 zurück.

BlockInfo

RBF-Dateien enthalten für jedes Zeichen die Information, ob es "belegt" ist oder nicht. **BlockInfo** liest diese Information für ein Zeichen aus (Rückgabewerte TRUE bzw. FALSE) . Zeichen die "nicht belegt" sind existieren trotzdem (können also mit **BlockLoad** gelesen werden), enthalten aber i.a. keine Daten. Beim Print wirken sie daher wie ein Leerzeichen.

Syntax:	**<numVar>** = **BlockInfo** (**fileName$**, **zeichen** [, **local**])
Parameter:	zeichen: ASCII-Codes des Zeichens Restliche Parameter: siehe **BlockSize**
Fehlerbehandlung:	siehe **BlockSize**

4 Einbinden von Hilfedateien

4.1 Überblick

R-BASIC stellt Ihnen das GEOS-weite Hilfesystem zur Verfügung. Damit können Sie Hilfedateien schreiben und einbinden, die sich in ihrer Handhabung nicht von den Hilfedateien der PC/GEOS-SDK-Programme unterscheiden.

Das Grundprinzip des GEOS Hilfesystems
Hilfedateien haben ein spezielles Format befinden sich grundsätzlich im Ordner USERDATA\HELP. Jede Hilfedatei wird über ihren Namen angesprochen. In der Hilfedatei befinden sich die einzelnen Hilfeseiten. Jede Hilfeseite hat einen internen Namen, der als "Help Context" bezeichnet wird. Wenn ein Programm wünscht, eine bestimmte Hilfeseite zu öffnen muss es dem Hilfesystem den Dateinamen (Help File) und den Help Context mitteilen. Das Hilfesystem öffnet dann die entsprechende Seite in der Hilfedatei und zeigt sie an. Beim Erstellen der Datei können Sie Verweise ("Hyperlinks") auf andere Hilfeseiten der gleichen Datei oder auch von anderen Hilfedateien festlegen. Damit kann der Nutzer zwischen den verschiedenen Hilfeseiten navigieren. Ein Hyperlink auf eine bestimmte Stelle innerhalb einer Seite (z.B. ans Seitenende) ist leider nicht möglich.

4.2 Ansprechen der Hilfe in R-BASIC

Instance-Variable	Syntax im UI-Code	Im BASIC-Code
helpFile$	helpFile$ = **"FileName"**	lesen, schreiben
helpContext$	helpContext$ = **"ContextName"**	lesen, schreiben

Um das Hilfesystem von GEOS nutzen zu können müssen Sie nur die Hilfedatei (Help File) und die aufzurufende Hilfeseite (Help Context) festlegen. Dazu stehen Ihnen die Instancevariablen **helpFile$** und **helpContext$** zur Verfügung. Den Rest erledigt das System. Beispielsweise kann über die Taste F1 automatisch die Hilfe aufgerufen werden. HelpFile$ und helpContext$ sind für alle GenericClass-Objekte definiert.
Um zu verstehen, wie sie diese Instancevariablen am besten einsetzten müssen Sie verstehen, wie das Hilfesystem arbeitet.

So arbeitet das Hilfesystem
Nehmen wir an, die haben einen Dialog offen, der über einen "Hilfe" Button verfügt. Klicken Sie auf den Button oder drücken Sie "F1" so wird das Hilfesystem von GEOS aktiviert. Diese sucht zuerst im Dialogobjekt nach einer Hilfedatei und einem HelpContext. Wenn der Programmierer im Dialog weder einen Wert für **helpContext$** noch einen für **helpFile$** definiert haben wendet sich das System an das Parent-Objekt des Dialogs, dann an dessen Parent usw., bis hin zum Application-Objekt.

Nehmen wir weiterhin an, der Programmierer hat dem Dialog einen **helpContext$** gegeben, aber keinen Wert für **helpFile$** festgelegt. Dann durchsucht das System die Parent-Objekt nur noch nach dem Namen der Hilfedatei (**helpFile$**). Sobald das System HelpFile und HelpContext zusammen hat zeigt es die entsprechende Hilfeseite an.

helpFile$

Syntax	UI- Code:	**helpFile$** = **"FileName"**
	Lesen:	**<stringVar>** = **<obj>** . **helpFile$**
	Schreiben:	**<obj>.helpFile$** = **"Text"**

HelpFile$** ordnet einem Objekt eine Hilfedatei zu. Da das GEOS Hilfesystem den Objekttree aufwärts (in Richtung der Parents) durchsucht ist diese Hilfedatei auch für alle Children des Objekts gültig. Deswegen wird dem Application Objekt mit **helpFile$ eine Hilfedatei zugeordnet. Die Instancevariable **helpFile$** ist aber für alle GenericClass Objekte zulässig. Häufig wird sie für Groups, Dialoge und Buttons verwendet, wenn diese eine Hilfedatei nutzen sollen, die von der im Applicationobjekt definierten "Haupthilfe" abweicht.
Die Hilfedatei ist eine GEOS-Datei. Deswegen sind für **helpFile$** bis zu 32 Zeichen zulässig.
Hinweis: Damit der Button "Inhalt" im Hilfefenster arbeiten kann müssen Sie für das Applicationobjekt einen Namen für die Hilfedatei (**helpFile$**) festlegen.

helpContext$

Syntax	UI- Code:	**helpContext$** = **"ContextName"**
	Lesen:	**<stringVar>** = **<obj>.helpContext$**
	Schreiben:	**<obj>.helpContext$** = **"Text"**

HelpContext$ ordnet einem Objekt eine Hilfe**seite** (Help Context) zu. Da das GEOS Hilfesystem den Objekttree aufwärts (in Richtung der Parents) durchsucht ist diese Hilfeseite auch für alle Children des Objekts gültig. Die Instancevariable **helpContext$** ist für alle GenericClass Objekte zulässig. Sehr häufig wird ein Help Context für Groups, Dialoge und Buttons verwendet, um die entsprechende Hilfeseite anzuzeigen. Den Namen der Hilfedatei bezieht das System dabei sehr häufig vom Applicationobjekt.
Die Textlänge für **helpContext$** ist auf 20 Zeichen begrenzt. Mehr lässt der GEOS Hilfeeditor in GeoWrite nicht zu.

Die Standardkonfiguration

Die meisten Programme verwenden nur eine einzige Hilfedatei und beim Aufruf der Hilfe wird die immer Startseite (Inhaltsverzeichnis, engl. Top Of Content, TOC) angezeigt. Das ist sehr einfach zu realisieren. Vereinbaren Sie im UI Code für das Applicationobjekt eine Hilfedatei (**helpFile$**) und für das Primaryobjekt den Help Context "**TOC**". Dadurch erscheint automatisch der "Hilfe" Button (blaues Fragezeichen) in der Titelzeile des Primaryobjekts und die Taste F1 zum Aufruf der Hilfe wird aktiviert.

```
Application DemoApplication
    Children = DemoPrimary
    helpfile$ = "R-BASIC Demo Help"
END Object

Primary DemoPrimary
    Children = .....
    SizeWindowAsDesired
    helpContext$ = "TOC"
END Object
```

Um den automatisch erzeugten Hilfebutton aus der Titelzeile des Primary wieder zu entfernen verwenden Sie bei bedarf den Hint PrimaryNoHelpButton.

Eigene Hilfeseiten für Dialoge

Ein sehr häufiger Fall ist, dass ein Dialog offen ist und der Nutzer Hilfe zu genau diesem Dialog benötigt. Dann kann er entweder die Taste F1 drücken oder einen speziellen "Hilfe" Button in diesem Dialog anklicken. Dieses Funktionalität lässt sich sehr einfach implementieren. Geben Sie einfach dem Dialogobjekt eine Help Context. Als Nebenwirkung erzeugt der Dialog automatisch einen Hilfebutton. Das Hilfesystem ruft dann beim Aufruf der Hilfe (F1 oder anklicken des Hilfebuttons) automatisch die dem Dialog zugeordnete Hilfeseite auf.

```
Dialog HelpedDialog
  Caption$ = "Dialog mit Hilfe"
  Children = ....
  helpContext$="SpecialDialogHelp"
  End Object
```

Bei Bedarf können Sie auch eine andere Hilfedatei (helpFile$) festlegen:

```
Dialog HelpedDialog
  Caption$ = "Dialog mit Hilfe"
  Children = ....
  helpContext$ = "SpecialDialogHelp"
  helpFile$ = "SpecialHelpFile"
  End Object
```

Der vom Dialog automatisch erzeugte Helpbutton hat ein Fragezeichen als Aufschrift. Um diese Aufschrift zu ändern muss man einen Button anlegen, dessen **interactionCommand** Wert auf IC_HELP gesetzt ist. Dieser Button ersetzt dann den vom Dialog erzeugten Button. Zusätzlich sollten Sie dem Button die Anweisung

placeObject = REPLY_BAR

geben, falls das angebracht ist.

```
Dialog HelpedDialog
  caption$ = "Dialog mit Hilfe"
  children = ... , DialogHelpButton
  dialogtype = DT_COMMAND
  helpContext$="MoreHelp"
  End Object
...
Button DialogHelpButton
  Caption$ = "Hilf mir"
  interactionCommand = IC_HELP
  placeObject = REPLY_BAR
  End Object
```

Direkter Aufruf spezieller Hilfeseiten

Insbesondere in einem "Hilfe" Menü oder auch in einem Dialog kann der Wunsch bestehen, mehrere konkrete Hilfeseiten direkt aufzurufen. Das lässt sich sehr einfach über Buttons realisieren, denen als ActionHandler das Schlüsselwort "BringUpHelp" zugewiesen wird. Diese Buttons sollten einen eigen Help Context und ggf. eine eigene Hilfedatei zugeordnet bekommen.

```
Dialog HelpedDialog
  caption$ = "Dialog mit mehr Hilfen"
  children = ... , HelpButton1, HelpButton2
  dialogtype = DT_COMMAND
  End Object
...
Button HelpButton1
  Caption$ = "Hilfe Thematisch"
  ActionHandler = BringUpHelp
  helpContext$ = "ThemaHelp"
  End Object

Button HelpButton2
  Caption$ = "Spezielle Hilfe"
  ActionHandler = BringUpHelp
  helpContext$ = "HelpSpecial"
  helpFile$ = "Help File 2"
  End Object
```

4.3 Unterstützung für "Virtual Desktop"

Wenn Sie das Programm "Virtual Desktop" von Jens-Michael Groß installiert haben möchten Sie vielleicht die dort verfügbare Funktionalität "BubbleHelp" auch für ihre R-BASIC Programme einsetzen. J.-M. Groß hat die BubbleHelp Funktion über die Help Contexte realisiert. Um z.B. einem Button die Bubble-Hilfe "Öffnet die Dateiauswahl" zu geben, verwenden Sie **helpContext$** mit drei führenden $-Zeichen.

```
Button FileSelectButton
  Caption$ = "Wählen"
  helpContext$ = "$$$Öffnet die Dateiauswahl"
  End Object
```

Virtual Desktop erkennt an den führenden $$$, dass es sich um eine Bubble-Hilfe handelt. Der Text hinter den $$$ kann bis zu 35 Zeichen lang sein.

Sie können BASIC Programme, denen Sie auf diese Weise eine BubbleHelp Funktionalität gegeben haben, problemlos an User verteilen, die Virtual Desktop nicht installiert haben. Sie laufen dort ohne Einschränkung.

4.4 Erstellen von Hilfedateien

Hilfedateien werden einfach mit GeoWrite erstellt. Dazu muss zuerst der Hilfeeditor freigeschaltet werden. Sie erkennen am Vorhandensein des Menüs "Hilfe-Editor" dass der Hilfeeditor bereits aktiviert ist.

So schalten Sie den Hilfeeditor frei

Wenn Sie noch kein Hilfe-Editor Menü haben gehen Sie folgendermaßen vor:

1. Starten Sie GEOS und stellen Sie sicher, dass GeoWrite nicht läuft. Fahren Sie GEOS dann herunter.

2. Fertigen Sie eine Sicherheitskopie ihrer GEOS.INI an und öffnen Sie sie mit einem Texteditor (nicht mit MSWord oder sowas!). Suchen Sie die Kategorie [configure] und tragen darunter die Zeile
   helpeditor = true
   ein. Wenn die Kategorie [configure] noch nicht existiert müssen Sie sie anlegen. Dieser Schritt stellt sicher, das man in GeoWrite den Hilfeeditor zuschalten kann. Speichern Sie die GEOS.INI und starten Sie GEOS und dann GeoWrite.

3. Wählen Sie in GeoWrite im Menü "Optionen" den Punkt "Benutzerebene ändern". Klicken Sie auf "Feineinstellung" und aktivieren Sie das

Kontrollkästchen "Hilfe-Editor". Wundern Sie sich bitte nicht, dass jetzt keine Benutzerebene mehr selektiert ist. Mehrfach OK klicken nicht vergessen.

4. Wählen Sie "Konfiguration speichern" aus dem Optionen-Menü.

5. Sollte das Hilfe-Editor Menü jetzt noch nicht zu sehen sein, schließen Sie bitte GeoWrite und starten Sie GEOS neu.

So erstellen Sie eine Hilfedatei

1. Planen Sie ihre Hilfedatei gründlich!

2. Öffnen Sie in GeoWrite eine neue Datei und speichern Sie sie unter einen möglichst eindeutigen Namen.

3. Schreiben Sie das "Inhaltsverzeichnis". Das ist die Startseite, von der aus auf die wichtigsten Unterthemen gesprungen werden soll. Sie muss natürlich die Texte für die Hyperlinks zu den Unterthemen enthalten. Ein Beispiel:

```
Hilfe zu Supergame
    Spielregeln
    Copyright
```

Wählen Sie für die Hyperlinks ein geeignetes Format, z.B. dunkelblau und unterstrichen.
Diese Seite kann, wie jede andere Hilfeseite auch, Grafiken oder Bilder enthalten. Sie sollten Grafiken und Bilder aber immer in die Textebene einfügen, sonst könnten Layoutprobleme beim der Anzeige der Hilfe die Folge sein.

4. Drücken Sie Strg-Enter um eine neue Seite zu öffnen. Es ist wichtig dass jede Hilfeseite auf einer neuen Seite beginnt.

5. Schreiben Sie den Hilfetext, z.B.:

```
Spielregeln
Wer gefressen wird hat verloren.
```

Denken Sie bei der Formatierung daran, dass der Nutzer das Hilfefenster später in seiner Größe verändern kann. Arbeiten Sie zur Formatierung nicht mit der TAB-Taste oder Leerzeichen sondern verwenden Sie konsequent Seitenränder und Absatzeinzüge! Auch Tabulatoren sind mit Vorsicht zu genießen, da die Hilfe meist nicht so breit ist wie das Textfenster von GeoWrite. Farben und Zeichenattribute sind natürlich auch erlaubt.

6. Jetzt müssen Sie die internen Seitennamen (Help Contexte) vereinbaren. Öffnen Sie dazu das Hilfe-Editor Menü und wählen Sie "Kontext erstellen". Erstellen Sie unbedingt einen Kontext namens **TOC**, der auch die Eigenschaft "TOC" aus der Auswahlliste oben bekommen sollte. TOC steht für Table of Content (Inhaltsverzeichnis) und jede Hilfedatei muss zwingend einen Kontext namens TOC haben, sonst arbeitet das Hilfesystem nicht richtig. Alle anderen Kontexte behalten die Eigenschaft "Text". Erstellen Sie weitere Kontexte mit beliebigen Namen, z.B. "Spielregeln" oder "Copyright". Leerzeichen sind zulässig, aber unpraktisch.

7. Nun müssen Sie die vereinbarten Help Contexte den einzelnen Seiten zuordnen. Gehen Sie dazu in GeoWrite zunächst auf Ihre Seite mit dem Inhaltsverzeichnis und markieren Sie die erste Zeile oder die ersten Zeilen. Dabei gelten die folgenden Regeln:
   - Der Anfang der ersten Zeile muss mit markiert sein. Das gilt auch, wenn es sich um eine Leerzeile handelt. Leerzeilen enthalten keine Zeichen, auch keine Leerzeichen oder Sondertext wie Tabulatoren oder Grafiken.
   - Beginnt die Seite mit einer oder mehreren Leerzeilen so sollten Sie alle führenden Leerzeilen, einschließlich der ersten Zeile mit Text, markieren.
   - Der Text der ersten Zeile, die ein Zeichen enthält, wird vom Hilfesystem für das Menü "Letzte Schritte" verwendet.
   - Das Markieren der ganzen Seite ist unzulässig.

   Öffnen Sie nun wieder das Hilfe-Editor Menü und wählen Sie "Kontext setzen". Klicken Sie auf "TOC" in der Liste der Kontexte und dann auf" Anwenden".
   Markieren Sie nun die ersten Buchstaben der erste Hilfeseite und weisen Sie ihm auch einen Kontext zu. Wichtig ist hier, dass nicht aus Versehen der Seitenumbruch mit markiert ist, sonst wird der Kontext der vorhergehenden Seite zugeordnet!
   Tipp: Beginnt die erste nicht-Leerzeile mit einer Grafik so findet das Hilfesystem keinen Text für die "Letzte Schritte" Liste. Um das zu vermeiden schreiben sie den Text für die "Letzte Schritte" Liste in die allererste Zeile, setzen ihn aber auf "unsichtbar", indem Sie im Menü "Zeichen" - "Textfarbe" den Wert für "Raster (%)" auf Null setzen.

8. Jetzt müssen Sie die Seiten verlinken. Markieren Sie auf der Seite mit dem Inhaltsverzeichnis den Link auf das erste Hilfethema. Öffnen Sie nun wieder das Hilfe-Editor Menü und wählen Sie "Hyperlink setzen". Wählen Sie einen Kontext und klicken Sie auf "Anwenden".
   Tipp: Setzen Sie hinter jeden Hyperlink, der allein auf einer Zeile steht ein einzelnes Leerzeichen, das nicht Teil des Links ist. Das verhindert, dass das Hilfesystem den Mauscursor bis an Ende der Zeile, wo gar nichts mehr steht, als "Hyperlink-Cursor" anzeigt.

9. Nun können Sie weitere Hilfeseiten, Kontexte und Hyperlinks nach belieben in das Dokument einfügen.

10. Zum Abschluss muss die Hilfedatei im USERDATA\HELP Ordner angelegt werden. Dazu öffnen Sie wieder das Hilfe-Editor Menü und wählen "Hilfedatei erstellen". Den Punkt "Daten komprimieren" sollten Sie aktiviert lassen.

GeoWrite legt eine **Kopie** des aktuellen Dokuments in USERDATA\HELP ab und ändert deren Format so, dass das Hilfesystem sie verwenden kann. Die GeoWrite-Datei bleibt erhalten. So können Sie ihre Hilfedatei später beliebig erweitern oder verändern.

Hilfesysteme mit mehreren Hilfedateien

Es ist möglich einen Hyperlink auf einen Kontext in einer anderen als der aktuellen Datei zu setzen. Damit können Sie sehr umfangreiche und komplexe Hilfesystem aufbauen. Um einen Hyperlink auf einen Kontext in einer fremden Datei zu setzten müssen Sie den Namen der Datei und den Namen des anzuspringenden Kontextes manuell vereinbaren.

1. Öffnen Sie den Punkt "Datei definieren" im Menü "Hilfe-Editor". Tragen Sie den Namen der Hilfedatei ein und klicken Sie auf "Datei hinzufügen".

2. Öffnen Sie den Punkt "Kontext erstellen" im Menü "Hilfe-Editor". Klicken Sie auf " • aktuelle Datei •" und wählen Sie die eben hinzugefügte Datei aus. Definieren Sie jetzt den Namen des Kontextes, den Sie in dieser Datei anspringen wollen.

3. Markieren Sie den Hyperlinktext in ihrem Dokument und öffnen Sie den Menüpunkt "Hyperlink setzen" aus dem "Hilfe-Editor" Menü. Wählen Sie dort die gewünschte Datei sowie den gewünschten Kontext aus und klicken Sie auf "Anwenden".

4. Erzeugen Sie die beiden betroffenen Hilfedateien (Menüpunkt "Hilfedatei erstellen"), starten Sie die zugehörige Anwendung bzw. Ihr BASIC Programm und testen Sie ihre Hilfedateien. Sollte es ein Problem geben prüfen Sie bitte die Schreibweise (Groß/Kleinschreibung, Leerzeichen) des Dateinamens und des Kontextes.

5 Arbeit mit der Zwischenablage

5.1 Überblick

Zugriff auf die Zwischenablage

Die Arbeit mit der Zwischenablage (auch Clipboard genannt) ist unter PC/GEOS auf zwei Arten möglich. Zum einen weiß jedes Objekt, z.B. ein Memo-Objekt, ob es mit der Zwischenablage zusammenarbeiten kann und wie es Daten (z.B. einen Text oder eine Grafik) in die Zwischenablage kopiert oder aus ihr herausholt. Zusätzlich unterstützt PC/GEOS den direkten Zugriff auf die Zwischenablage, d.h. eine Applikation kann ihre eigenen Daten in einem eigenen Format in die Zwischenablage kopieren bzw. von dort lesen. R-BASIC bietet ebenfalls beide Wege an. Auf Objektebene stehen die Methoden (Objektanweisungen) **ClpTestCopy**, **ClpTestPaste**, **ClpCopy** und **ClpPaste** zur Verfügung. Das Programm selbst kann mit den Befehlen **ClipboardTest**, **ClipboardPut**, **ClipboardGet**, **ClipboardPutGS**, **ClipboardGetGS**, **ClipboardPutBitmap** und **ClipboardGetBitmap** direkt auf die Zwischenablage zugreifen. Die globale Variable **clipboardError** wird auf TRUE gesetzt, wenn es ein Problem bei der Arbeit mit der Zwischenablage gab.

Überwachung der Zwischenablage

Ein R-BASIC Programm kann sich bei Bedarf über Änderungen des Inhalts der Zwischenablage informieren lassen. Jedes Mal wenn irgendein Programm (z.B. GeoWrite) Änderungen an der Zwischenablage vornimmt wird vom Application-Objekt ein spezieller Handler (**OnClpChange**-Handler) gerufen.

Das ClipboardFormat

Will ein Programm oder Objekt z.B. einen Text aus der Zwischenablage entnehmen, so muss es sicher sein, dass sich auch ein Text in der Zwischenablage befindet. Deshalb muss beim Kopieren irgendwelcher Daten in die Zwischenablage immer eine Information mit abgespeichert werden, worum es sich handelt. Diese Information heißt ClipboardFormat und besteht aus zwei WORD-Werten, der **Manufacturer-ID** und der **FormatNummer**. Der erste Wert, die Manufacturer-ID, frei übersetzt die "Hersteller-Kennung", beschreibt, wer die Software, die Daten ins Clipboard kopiert, geschrieben hat. Der zweite Wert, die FormatNummer, ist einfach eine laufende Nummer, die verschiedene Clipboardformate der gleiche Softwareschmiede unterscheiden soll.

Standardformate

Bei allen von PC/GEOS selbst definierten Formaten (z.B. Text, Bitmap, Graphic String) ist die Manufacturer-ID Null. Null ist die Herstellerkennung von GeoWorks. BreadBox hat die Nummer 16431 und der Programmierer von R-BASIC hat die Nummer 16480. GeoWorks hat einige grundlegende Formate z.B. Text (FormatNr. 0), Graphic String (Folge von Grafikbefehlen, Nr. 1) und Bitmap (Nr. 7) sowie einige mehr definiert. Diese werden als Standardformate bezeichnet und sind teilweise im PC/GEOS-SDK dokumentiert. Unter R-BASIC ist ein Zugriff auf die Formate Text, Graphic String und Bitmap über einige R-BASIC Objekte möglich. Ein BitmapContent-Objekt kann z.B. sowohl eine Bitmap als auch einen Graphic String aus dem Clipboard lesen und Textobjekte (Memo, InputLine,

VisText und LargeText) können selbstverständlich mit Texten im Clipboard umgehen.
Weiter unten wird beschrieben, wie sie selbst zu einer Manufacturer-ID kommen, falls Sie eine benötigen.

Mehrere Formate im Clipboard
Unter PC/GEOS können die Daten in der Zwischenablage in mehr als einem Format gleichzeitig abgelegt werden. Beispielsweise speichert GeoDraw seine Grafiken immer sowohl als editierbare GeoDraw-Objekte als auch als Graphic String (der z.B. von GeoWrite oder vom R-BASIC Bitmap-Objekt gelesen werden kann). Der Grafikbetrachter Gonzo und auch das R-BASIC BitmapContent Objekt kopiert Bilder sowohl als reine Bitmap als auch als Graphic String in die Zwischenablage. Wenn Sie in R-BASIC eigene Clipboardformate verwenden, können Sie allerdings nur genau ein Format gleichzeitig in die Zwischenablage kopieren.

5.2 Clipboardoperationen

Aus dem Bearbeiten-Menü kennen Sie die Clipboardoperationen "Kopieren", "Einfügen" und "Ausschneiden". Diese sind unter PC/GEOS auf ganz elementarer Ebene und für alle Objekte definiert - auch wenn die meisten Objekte (z.B. ein Button) gar nicht mit dem Clipboard zusammenarbeiten können. R-BASIC übernimmt das und deswegen sind die Clipboardoperationen auch unter R-BASIC für alle Objekte erlaubt. Sehr viele, wie z.B. ein Button, ignorieren die entsprechenden Anweisungen aber. Bei denjenigen Objekten, die mit dem Clipboard arbeiten können, z.B. BitmapContent und Textobjekte, finden Sie bei der Beschreibung dieser Objekte die entsprechenden Detailinformationen.

Methoden:

Methode	Aufgabe
ClpTestCopy	Prüfen, ob Daten ins Clipboard kopiert werden können
ClpCopy	Daten ins Clipboard kopieren
ClpTestPaste	Prüfen, ob passende Daten im Clipboard sind
ClpPaste	Daten aus dem Clipboard holen

globale Variablen:

Variable	Aufgabe
clipboardError	enthält im Fehlerfall TRUE, sonst FALSE

Das Kopieren von Daten ins Clipboard wird als Copy-Operation bezeichnet (engl. to copy: etwas kopieren), das Einfügen von Daten aus dem Clipboard in ein Objekt wird als Paste-Operation bezeichnet (engl. to paste: etwas einkleben, etwas einfügen). Die Operation "Ausschneiden" wird von R-BASIC nicht direkt unterstützt. Sie ist identisch mit "Kopieren ins Clipboard" und anschließendem Löschen der Daten aus dem Objekt. Das lässt sich bei Bedarf sehr leicht manuell implementieren.

clipboardError

Es ist möglich, wenn auch extrem unwahrscheinlich, dass in der kurzen Zeitspanne zwischen den Prüfen der Zwischenablage mit **ClpTest** bzw. **ClipboardTest** und der Verwendung der Daten mit **ClpPaste** bzw. **ClipboardGet** eine andere Applikation den Inhalt der Zwischenablage verändert hat. Die Clipboard "Lese" Operationen **ClipboardGet** und **ClpPaste** setzen deswegen die globale Variable **clipboardError** auf TRUE (Fehler) oder FALSE (OK), je nachdem ob sie erfolgreich waren oder nicht.

ClpTestCopy, ClpCopy

Die Methode **ClpTestCopy** weist das Objekt an zu prüfen, ob es Daten ins Clipbord kopieren kann. Die Methode liefert den Wert TRUE (–1, entspricht ja) oder FALSE (Null, entspricht nein). Objekte, die nicht mit dem Clipboard arbeiten können liefern hier immer Null, also nein. Die globale Variable **clipboardError** wird von ClpTestCopy nicht verändert.

Syntax BASIC-Code: **<numVar>** = **<obj>**.**ClpTestCopy**
Liefert: TRUE: Daten zum Kopieren vorhanden
FALSE: Kann keine Daten kopieren

Die Methode **ClpCopy** weist das Objekt an, seine Daten ins Clipboard zu kopieren. Ist es dazu nicht in der Lage, entweder weil das Objekt gar nicht mit dem Clipboard arbeiten kann oder weil gerade keine Daten verfügbar sind, wird die Anweisung ignoriert und die globale Variable **clipboardError** wird auf TRUE gesetzt.

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

Beispiel:

```
DIM n
n = QuestionBox ("Text kopieren?")
IF n = YES then
  DemoText.ClpCopy
  end if
```

ClpTestPaste, ClpPaste

Mit der Methode **ClpTestPaste** können Sie erfahren, ob sich Daten im Clipboard befinden, die vom Objekt akzeptiert werden. Dazu prüft das Objekt, ob sich ein geeignetes Format im Clipboard befindet. Objekte, die nicht mit dem Clipboard arbeiten können liefern hier immer FALSE (Wert Null, bedeutet nein). Die globale Variable **clipboardError** wird von ClpTestPaste nicht verändert.

Syntax BASIC-Code: **<numVar>** = **<obj>**.**ClpTestPaste**
Liefert: TRUE: Daten zum Einfügen gefunden
FALSE: Kann nichts einfügen

Die Methode **ClpPaste** weist das Objekt an, Daten aus dem Clipboard bei sich selbst "einzufügen". Ob das ein "Hinzufügen" oder ein "Ersetzen" ist hängt vom Objekt ab. Ist das Objekt nicht in der Lage, die Clipboarddaten einzufügen, entweder weil das Objekt gar nicht mit dem Clipboard arbeiten kann oder weil es ein Problem gibt, wird die Anweisung ignoriert. Je nach Objekt und Situation kann es eine Fehlermeldung geben oder auch nicht. Die meisten Objekte (mit Ausnahme der Textobjekte) setzen die globale Variable **clipboardError** auf TRUE bzw. auf FALSE.

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

Beispiel:

```
IF DemoBitmap.ClpTestPaste THEN
    DemoBitmap.ClpPaste
  else
    MsgBox "Keine Grafik im Clipboard"
  end if
```

5.3 Das Clipboard überwachen

Um in R-BASIC z.B. ein "Bearbeiten" Menü zu implementieren müssen Sie wissen wenn jemand etwas ins Clipboard kopiert und was es ist. Dann können Sie z.B. einen "Einfügen" Schalter enablen oder disablen. Für dieses Zweck verfügt das Application Objekt (und nur dieses) über einen speziellen Actionhandler, der immer dann aufgerufen wird, wenn sich im Clipboard etwas tut.

Instance-Variable:

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

Action-Handler-Typen:

Handler-Typ	Parameter
SystemAction	(sender as object, state as word, data1 as word, data2 as word)

Der **OnClpChange** Handler wird automatisch immer dann aufgerufen wenn sich die Daten im Clipboard ändern. Die übergebenen Parameter sind hier ohne Bedeutung und sollten ignoriert werden. Wenn die Instancevariable **OnClpChange** erstmalig belegt wird, meldet sich das Applicationobjekt beim System für die Clipboardüberwachung an und erhält fortan automatisch die

entsprechenden Informationen. Erstmalig wird der Handler bereits gerufen, wenn sich das Applicationobjekt anmeldet. Da **OnClpChange** üblicherweise im UI-Code belegt wird erfolgt der erstmalige Aufruf schon beim Programmstart. Damit ist das BASIC Programm stets über den Stand des Clipboards informiert.

Das folgende Beispiel zeigt das Fragment einer typischen Implementation. Der "Einfügen" Button ist nur dann aktiv wenn sich auch eine Grafik im Clipboard befindet.

Im UI Code:

```
Application DemoApplication
    Children = DemoPrimary
    OnClpChange = ClpChangeHandler
END Object

BitmapContent DemoBitmap
  ....
END Object

Button PasteButton
  Caption$="Einfügen"
  enabled = FALSE         ' sicherheitshalber
  ActionHandler = PasteImageHandler
END Object
```

Im BASIC Code:

```
'
' Der Handler enabled oder disabled den Einfügen Button
'
SYSTEMACTION ClpChangeHandler
DIM ok
  ok = DemoBitmap.ClpTestPaste
  IF ok THEN
    PasteButton.enabled = TRUE
    ELSE
    PasteButton.enabled = FALSE
    END IF

END Action

'
' Der Button Handler ist sehr simpel
'
BUTTONACTION PasteImageHandler
  DemoBitmap.ClpPaste
END Action
```

5.4 Eigene Formate verwenden

Bei komplexen Anwendungen kann es sinnvoll oder nötig sein, eigene Daten von R-BASIC aus ins Clipboard zu kopieren und die wieder von dort zu lesen. Ein einfacher Fall wäre das Kopieren von Inhalten von einem Dokument in ein anderes. R-BASIC unterstützt das über die drei Befehle: **ClipboardTest**, **ClipboardPut** und **ClipboardGet** sowie mit der globalen Variaben **clipboardError**. Dabei wird jeweils der Inhalte einer Strukturvariablen ins Clipboard kopiert bzw. von dort gelesen. Sie sollten daher mit den Grundlagen der Verwendung von Strukturen (Schlüsselwort **STRUCT**) vertraut sein um die folgenden Abschnitte vollständig zu verstehen.

clipboardError

Enthält im Fehlerfall TRUE, sonst FALSE. Details siehe Abschnitt "Clipboard-operationen".

ClipboardTest

ClipboardTest prüft, ob sich Daten mit einem bestimmten Format im Clipboard befinden. Das Format kann ein GEOS Standardformat oder eine eigenes Format sein. Die globale Variable **clipboardError** wird nicht verändert.

Syntax BASIC Code:	**<numVar>** = **ClipboardTest** (**manufID**, **formatNr**)
	manufID: Manufacturer-ID des Formats
	formatNr: Nummer des Formats
	Return: TRUE: Format gefunden
	FALSE: Format nicht im Clipboard

Die folgenden Tabelle enthält eine Auswahl der von GeoWorks definierten Clipboardformate. Einige dieser Formate sind im PC/GEOS-SDK dokumentiert, andere nicht.

Manufacturer-ID	Format-Nummer	Inhalt
0	0	Text
0	1	Graphic String (eine Folge von Zeichenbefehlen)
0	3	Tabellenkalkulationsdaten
0	5	GeoDraw Objekte
0	6	GeoDex Daten
0	7	Bitmap

Unter R-BASIC ist ein Zugriff auf die Formate Text, Graphic String und Bitmap über R-BASIC Objekte möglich.

ClipboardPut

Mit **ClipboardPut** können Sie den Inhalt einer Strukturvariablen (Strukturausdrücke sind auch erlaubt) ins Clipboard kopieren. Dazu müssen Sie ein eigenes ClipboardFormat "definieren" indem sie eine eindeutige Kombination von Manufacturer-ID und Formatnummer verwenden. **ClipboardPut** kann nicht fehlschlagen, die globale Variable **clipboardError** wird immer auf FALSE gesetzt.

Syntax BASIC Code: **ClipboardPut** **<structExpr>**, **manufID**, **formatNr**
<structExpr> Struktur Variable oder Ausdruck
manufID: Manufacturer-ID ihres Formats
formatNr: Nummer ihres Formats

ClipboardPut erkennt alle sonstigen nötigen Daten wie Größe der Struktur automatisch. Es kopiert die Struktur 1:1 ins Clipboard und legt als ClipboardFormat die Kombination aus manufID und formatNr fest.

Beispiel

```
STRUCT   Mydata
  x, y   AS   INTEGER
  r,g,b  AS   BYTE
  text$  AS   STRING(60)
  info$  AS   STRING(30)
  END STRUCT

DIM      dat  AS Mydata
  ...                                ' hier dat mit Werten belegen
  ClipboardPut dat, 16480, 12        ' RABE-Soft ID + Nr.12
```

Wenn Sie bereits eine eigene Manufacturer-ID besitzen ist das "Definieren" eines eigenen ClipboardFormats ganz einfach: Sie verwenden ausschließlich ihre eigene Manufacturer-ID und denken sich für jedes Programm, dass ein eigenes ClipboardFormat benötigt, eine (oder mehrere) willkürliche Formatnummer(n) aus. Schreiben Sie sich alle verwendeten Werte auf und legen Sie die Liste gut weg, damit Sie keinen Wert doppelt verwenden. Das ist schon alles. Es gibt keine zentrale Stelle wo Sie ihr Format "anmelden" oder gar "genehmigen lassen" müssen.

Falls Sie noch keine eigene Manufacturer-ID besitzen sollten Sie sich bei BreadBox eine besorgen. Wenn sie das wirklich nicht wollen ist die Sache komplizierter. Sie müssen sich dann eine "ausdenken" - was unter (wenn auch sehr unwahrscheinlichen) Umständen zu Konflikten führen kann. Lesen Sie dazu bitte den Abschnitt 3.4 (Über die Manufacturer ID) im R-BASIC Benutzer Handbuch.

ClipboardGet

Mit **ClipboardGet** können Sie den Inhalt des Clipboards in eine Strukturvariable kopieren. **ClipboardGet** prüft dabei, ob sich das ClipboardFormat, dass Sie mit

den Parametern manufID und formatNr spezifiziert haben, auch wirklich im Clipboard befindet. Sollte das nicht der Fall sein liefert **ClipboardGet** eine "leere" Struktur (alles Nullen) zurück. Es erfolgt keine weitere Fehlermeldung. Verwenden Sie vorher ClipboardTest, und prüfen Sie auch die globale Variabe **clipboardError** ab, wenn Sie sicher sein wollen.

Syntax BASIC Code: **<structVar>** = ClipboardGet (**manufID**, **formatNr**)
<structVar> Struktur-Variable
manufID: Manufacturer-ID ihres Formats
formatNr: Nummer ihres Formats

ClipboardGet erkennt alle sonstigen nötigen Daten wie Größe der Struktur automatisch.

Achtung! **ClipboardGet** führt keine weiteren Prüfungen aus! Wenn das ClipboardFormat 'manufID' + 'formatNr' gefunden wurde, wird versucht die Daten zu kopieren. Das heißt z.B.

- Wenn Sie statt ihres gewünschten Formats zufällig ein Format spezifiziert haben, dass sich im Clipboard befindet, greift **ClipboardGet** darauf zu. Im günstigsten Fall erhalten Sie Müll, im schlechtesten crasht das System.
- Wenn der Typ der Strukturvariablen links vom Gleichheitszeichen nicht mit dem Type der Variablen, die bei ClipboardPut verwendet wurde, übereinstimmt, erhalten Sie Müll.

Beispiel

```
DIM                         dat AS Mydata                          '
  Siehe ClipboardPut
  dat = ClipboardGet (16480, 12)
```

5.5 Bitmaps und GStrings

R-BASIC kann sowohl auf Bitmapdaten (ManufacturerID = 0, formatNr = 7) als auch auf Graphic Strings (ManufacturerID = 0, formatNr = 1) im Clipboard direkt zugreifen bzw. diese ins Clipboard kopieren. Dabei werden die Bitmap bzw. der GString über Handles referenziert.
Um zu prüfen, ob sich das richtige Format im Clipboard befindet können Sie die Routine ClipboardTest (siehe oben) verwenden. Um zu prüfen, ob die Operation erfolgreich war können Sie die globale Variable clipboardError abfragen.

ClipboardPutBitmap, ClipboardGetBitmap, FreeBitmap

Diese Befehle ermöglichen es eine Bitmap unabhängig vom BitmapContent-Objekt in das Clipboard zu kopieren oder von dort zu lesen. Eine ausführliche Beschreibung dieser Befehle finden Sie im Kapitel 2.8.6.4 (Bitmaps und BitmapHandles) des Programmierhandbuchs.

ClipboardPutGS, ClipboardGetGS

Diese Befehle ermöglichen es einen Graphic String in das Clipboard zu kopieren oder von dort zu lesen. Eine ausführliche Beschreibung dieser Befehle finden Sie im Kapitel 2.8.5 (Arbeit mit Graphic Strings) des R-BASIC Programmierhandbuchs.

(Leerseite)

6 Das Dateisystem

6.1 Dateitypen

GEOS kennt verschiedene Dateitypen. Aus Gründen des Systematik werden in diesem Zusammenhang auch Ordner als spezielle Type von Dateien angesehen. Die Tabelle unten enthält eine Übersicht sowie die Namen der numerischen Konstanten, die R-BASIC für diesen Zweck zur Verfügung stellt. Der Vorsatz GFT_ steht für "GeosFileType"

FileType

Mit der Funktion **FileType** ermitteln Sie den Typ einer Datei. Die Datei kann durch ihren Namen oder eine Dateivariable spezifiziert werden. Die Systemvariable **fileError** wird gesetzt oder gelöscht.

Syntax: **<numVar>** = **FileType** (**fileName$**)
<numVar> = **FileType** (**<fh>**)
fileName$ Name der Datei. Pfadangaben im Namen sind zulässig.
<fh>: Variable (oder Ausdruck) vom Typ FILE. Bezeichnet die Datei.
Return: Dateityp entsprechend der Tabelle unten

Beispiel:

```
DIM  type as word
type = FileType ("Bilder")
IF ( type = GFT_DIRECTORY) THEN Print "Es ist ein Ordner!"
```

Wert	Name der Konstante	Bedeutung
0	GFT_NOT_GEOS_FILE	Keine GEOS-Datei, sondern eine DOS-Datei
1	GFT_EXECUTABLE	Ausführbare GEOS-Datei (Applikation, Library)
2	GFT_VM	GEOS-VM-Datei (z.B. ein Write-Dokument)
3	GFT_DATA	GEOS-Daten-Datei (unterstützt von R-BASIC)
4	GFT_DIRECTORY	Keine Datei, sondern ein Ordner

Tabelle: GeosFileTypen

Hinweis für PC/GEOS-SDK-Programmierer: Die PC/GEOS-SDK-Routinen hinter diesem Befehl (FileGetHandleExtAttributes und FileGetPathExtAttributes) liefern beim Auffinden einer DOS-Datei oder eines DOS-Verzeichnissen den Fehler ERROR_ATTR_NOT_FOUND. R-BASIC fängt das ab und liefert den korrekten FileType-Wert.

6.2 Fehlerbehandlung, die Variable *fileError*

Bei Dateioperationen kann es immer vorkommen, dass ein Problem auftritt, z.B. dass eine Datei nicht gefunden wird. Das erfordert meistens keinen Programmabbruch, sondern das Programm kann darauf reagieren, indem es die numerische Systemvariable **fileError** abfragt.

Alle Befehle, die mit Dateien und Pfaden arbeiten (mit Ausnahme der FileFind-Routinen) belegen die **fileError**-Variable.

Das bedeutet: Trat eine Fehler auf, wird ein Fehlercode (Fehlernummer) in der Variablen angelegt. Trat kein Fehler auf, wird die Variable mit Null belegt. Der Wert wird dort bis zur nächsten Dateioperation gespeichert und kann beliebig oft abgefragt werden. Im Anhang finden Sie eine Liste der Fehlercodes.

ErrorText$

In GEOS sind sehr viele Fehlercodes definiert, nicht nur für Dateioperationen. Einige davon sind MS-DOS-Fehler, andere sind GEOS-intern. Bei der Arbeit an einem Programm, das Fehlercodes auswerten und entsprechend reagieren soll, aber auch bei der Fehlersuche im eigenen Programm selbst, ist es sehr hilfreich, die Bedeutung dieser Codes zu kennen.

Die Funktion **ErrorText$** liefert zu einem Fehlercode den passenden Text. Das sind weitgehend selbsterklärende, aber englische Bezeichnungen, die 1:1 dem PC/GEOS-SDK entnommen wurden, z.B. ERROR_PATH_NOT_FOUND (Pfad nicht gefunden).
Bei unbekannten Fehlern wird ein Text in der Form <ERROR_CODE: 301> geliefert.

Syntax: **<stringVar>** = ErrorText$(**fehlerCode**)

Beispiel:

```
IF ( fileError ) THEN  Print   ErrorText$(  fileError )
```

6.3 Arbeit mit FILE Variablen

Eine komplette Beschreibung der Variablentypen, auch des Datentyps FILE , finden Sie im R-BASIC Programmierhandbuch.
Variablen vom Typ **FILE** können behandelt werden wie alle anderen Variablen in R-BASIC auch. Man kann z.B. Felder von Dateivariablen anlegen (z.B. DIM dateien(10) AS FILE), sie als Elemente von Strukturen verwenden, als Parameter an SUB's oder FUNCTION's übergeben oder als Rückgabetyp von FUNCTION's benutzen. Nur "rechnen" kann man mit ihnen nicht.

An dieser Stelle soll noch einmal auf die Funktionen NullFile und FileInfo$ hingewiesen werden.

NullFile

NullFile( ) ist eine Funktion, die eine "leere" Dateivariable liefert, d.h. sie dient zum Löschen einer Dateivariable. **Achtung! NullFile( )** schließt die Datei nicht. Verwenden Sie dazu vorher **FileClose()**.

Syntax: **<han>** = **NullFile**( )
Die Klammern sind erforderlich, weil NullFile eine Funktion ist.
<han>: Variable vom Typ **FILE**

FileInfo$

Die Funktion **FileInfo$** liefert einen Text, der interne Informationen über eine Dateivariable liefert. Sie können diese Funktion zur Fehlersuche einsetzen.

Syntax: **<stringVar>** = **FileInfo$**(**<f>**)
<f>: Variable oder Ausdruck vom Typ **FILE**
<stringVar>: Stringvariable

(Leerseite)

7 Arbeit mit Pfaden und Ordnern

7.1 Angabe von Pfaden

Für R-BASIC-Befehle, die einen Pfad erwarten (z.B. CreateDir, DeleteDir SetCurrentPath, SetStandardPath, PushDir, PopDir und viele **File~** Befehle) gelten einheitliche Konventionen, wie Pfade angegeben werden können.

Relative Pfade können ein einfacher Ordnername (z.B. "Bücher") oder eine einfacher Pfad (z.B. "Bücher\\Informatik\\BASIC") sein.
Der Pfad bezieht sich auf das aktuelle Arbeitsverzeichnis.
Man beachte, dass ein Backslash in Stringkonstanten doppelt angegeben werden muss.

Pfade relativ zum Wurzelverzeichnis beginnen mit einem Rückwärts-Strich ("Backslash", '\')
z.B. "\\Bücher" oder "\\Bücher\\Informatik\\BASIC"
Der Pfad bezieht sich auf das Wurzelverzeichnis des aktuellen Laufwerks.

Absolute Pfadangaben enthalten am Beginn eine Laufwerksbuchstaben.
z.B. "F:\\Bücher" oder "F:\\Bücher\\Informatik\\BASIC"
Diese beziehen sich immer auf das angegebene Laufwerk.

Eines Sonderfall stellen "reine" Laufwerksbezeichner dar,
z.B. "F:" oder "D:\\" (doppelten Backslash beachten)
Beide Schreibweisen kann man in SetCurrentPath verwenden und BEIDE beziehen sich auf das Wurzelverzeichnis des angegebenen Laufwerks. Im Gegensatz zu DOS unterstützt GEOS keinen "aktuellen" Pfad auf jedem Laufwerk.

Standardpfade beziehen sich auf typische GEOS-Ordner, wie z.B. WORLD oder DOCUMENT. Sie werden über numerische Konstanten (SP_TOP, SP_DOCUMENT usw.) eingestellt. Die entsprechenden Befehle finden Sie weiter unten im Kaptiel 7.4.

Weitere Hinweise:

- Jeder Pfad kann ein beliebiger Stringausdruck sein.
- In Stringkonstanten muss der Backslash ('\') doppelt angegeben werden, da ein einfacher Backslash ein Sonderzeichen einleitet (z.B. Zeilenumbruch "\r") oder einen ASCII Code (z.B. "\201"):

7.2 Anlegen und Löschen von Ordnern und Pfaden

Ordner werden auch als "Verzeichnisse" (englisch: Directory) bezeichnet. Viele diesbezügliche Befehle enthalten deswegen ein "DIR" im Namen.
R-BASIC kann ganze Pfade, bestehend aus mehreren Verzeichnissen, auf einmal anlegen oder löschen. Die meisten Programmiersprachen können das nicht.

CreateDir

CreateDir (erzeuge Directory, lege Ordner an) legt einen Ordner oder einen kompletten Pfad auf einem Datenträger an. R-BASIC kann, im Gegensatz zu vielen anderen Programmiersprachen und dem PC/GEOS-SDK einen kompletten Pfad aus mehreren Unterverzeichnissen auf einmal anlegen.

Syntax: **CreateDir** **path$**

path$: Bezeichnung des anzulegende Pfades. Es darf eine absolute oder eine relative Pfadangabe (siehe Kapitel 7.1) sein.

CreateDir setzt die Systemvariable **fileError**, z.B. wenn das spezifizierte Laufwerk nicht existiert. Trat kein Fehler auf, enthält **fileError** den Wert 0.

DeleteDir

DeleteDir (lösche Directory) löscht einen Ordner oder einen kompletten Pfad vom Datenträger. DeleteDir setzt voraus, das sich keine Dateien mehr im zu löschenden Ordner befinden, ansonsten kann der Ordner nicht gelöscht werden und die Variable **fileError** wird entsprechend gesetzt.

Syntax: **DeleteDir** **path$** [,**killAll**]

path$ Bezeichnung des zu löschenden Pfades. Es darf eine absolute oder eine relative Pfadangabe sein.
killAll Wenn angegeben und ungleich Null (z.B. 1, YES oder TRUE): Es soll der gesamte Pfad gelöscht werden. Ansonsten (killAll nicht angegeben oder Null) löscht **DeleteDir** nur den letzten Ordner eine Pfades.

DeleteDir setzt die Systemvariable **fileError**, z.B. wenn der zu löschende Ordner nicht existiert oder nicht leer ist. Soll ein Pfad gelöscht werden, so müssen alle zu löschenden Ordner leer sein. Trat kein Fehler auf, enthält **fileError** den Wert 0.

Beispiele:

```
CreateDir "Bücher"
CreateDir "G:\\Daten\\Programme"
s$ = "\\daten" : CreateDir   s$ + "\\Programme"
```

```
DeleteDir "Bücher"
DeleteDir "G:\\Daten\\Programme"
                              ' Der Ordner Programme aus dem
                              Ordner "G:\Daten" wird gelöscht
DeleteDir "G:\\Daten\\Programme",  YES
                              ' Der komplette Pfad
                              "G:\Daten\Programme" wird gelöscht

' Fehlerabfrage
DeleteDir "G:\\Daten\\Programme",  YES
IF ( fileError ) THEN MsgBox "G: \\Daten\\Programme konnte nicht
                              vollständig gelöscht werden"
```

Hinweis: Zur Abwärtskompatibilität mit älteren BASIC-Programmen kann man statt CreateDir auch MKDIR ("Make Directory") und statt DeleteDir auch RMDIR ("Remove Directory") schreiben.

7.3 Der aktuelle Ordner

Alle Dateioperationen in R-BASIC beziehen sich auf ein bestimmtes Verzeichnis, das "aktuelle Verzeichnis". Der Begriff "Pfad" (englisch "path") wird verwendet, wenn ein Weg über mehrere Verzeichnisse gemeint ist bzw. gemeint sein kann.

currentPath$, currentDir$, geosPath$

Diese Systemvariablen enthalten die Namen wichtiger Pfade bzw. Verzeichnisse.

currentPath$ enthält den aktuellen Pfad z.B. "D:\GEOS\DOCUMENT\Bilder"
currentDir$ enthält das aktuelle Verzeichnis ohne Pfad z.B. "Bilder"
geosPath$ enthält das Geos-Hauptverzeichnis z.B. "D:\GEOS"

Beispiel:

```
PRINT "Geos befindet sich im Ordner ";geosPath$
```

SetCurrentPath

SetCurrentPath (Setze aktuellen Pfad) stellt das aktuelle Verzeichnis ein.

Syntax: **SetCurrentPath** **path$**

path$: Bezeichnung des einzustellenden Pfades. Es darf eine absolute oder eine relative Pfadangabe sein.

SetCurrentPath setzt die Systemvariable **fileError**, z.B. wenn das spezifizierte Laufwerk nicht existiert. Trat kein Fehler auf, enthält **fileError** den Wert 0.

Beispiele:

```
SetCurrentPath "Bücher"
SetCurrentPath "G:\\Daten\\Programme"
s$ = "\\daten" :     SetCurrentPath    s$ + "\\Programme"
```

Hinweis: Zur Abwärtskompatibilität mit älteren BASIC-Programmen kann man statt SetCurrentPath auch CHDIR ("Change Directory") schreiben.

PushDir, PopDir

GEOS verfügt über die einzigartige Fähigkeit, sich den aktuellen Pfad zu merken. Dafür wird eine sogenannter "Stack" (deutsch: Stapelspeicher), verwendet. Wie bei einem Papierstapel wird der Wert, der als letztes auf dem Stapel abgelegt wurde ("Push"-Operation) als erstes wieder vom Stapel entnommen ("Pop"-Operation). Zu jedem **PushDir** muss es daher ein dazugehöriges **PopDir** geben.

Syntax: **PushDir**
Das aktuelle Verzeichnis wird auf dem Stack abgelegt.

Syntax: **PopDir**
Das zuletzt auf dem Stack abgelegte Verzeichnis vom Stack geholt und als aktuelles Verzeichnisse wieder eingestellt.

PushDir und **PopDir** setzen die Variable **fileError** zurück (Null, kein Fehler).

Beispiel:

```
! Es sei "G:\Daten\Audio" das aktuelle Verzeichnis

PushDir                  ' Merken von "G:\Daten\Audio"
SetStandardPath    SP_TOP' Einstellen von, z.B. "C:\GEOS"
----
PopDir                   ' Wiederherstellen von "G:\Daten\Audio"
```

7.4 GEOS Standardpfade

SetStandardPath

SetStandardPath stellt eines der GEOS-Standardverzeichnisse als aktuelles Verzeichnis ein. Das funktioniert unabhängig davon, auf welchem Laufwerk und in welchen Verzeichnis GEOS installiert ist und auch dann, wenn eine länderspezifische GEOS-Version vorliegt, deren Verzeichnisse anders heißen.

Syntax: **SetStandardPath** **pfadKonstante**

pfadKonstante:
Eine der Standard-Pfadkonstanten. Siehe Tabelle.
Achtung! Ein ungültiger Wert für 'pfadKonstante' kann das System abstürzen lassen!

Die Variable **fileError** wird zurückgesetzt (Null, kein Fehler).

Beispiele:

```
SetStandardPath    SP_DOCUMENT
SetStandardPath    SP_TOP
```

R-BASIC kennt die folgenden Standard-Pfadkonstanten

Konstante	Wert	Eingestelltes Verzeichnis
SP_TOP	1	Geos Hauptverzeichnis
SP_WORLD	3	WORLD-Verzeichnis
SP_DOCUMENT	5	DOCUMENT-Verzeichnis
SP_SYSTEM	7	SYSTEM-Verzeichnis
SP_PRIV_DATA	9	PRIVDATA-Verzeichnis
SP_USER_DATA	19	USERDATA-Verzeichnis
SP_HELP	39	HELP-Verzeichnis (Hilfedateienverzeichnis)
SP_TEMPLATE	41	TEMPLATE-Verzeichnis (Vorlagenverzeichnis)
SP_DOS_ROOM	45	DOS-Room-Verzeichnis
SP_WASTE_BASKET	49	Papierkorb
SP_BACKUP	51	Sicherheitskopien-Verzeichnis

Im PC/GEOS SDK sind weitere Konstanten definiert.

ConstructPath$

ConstructPath$ erzeugt einen vollständigen Pfad aus einer Pfadkonstante und einem Unterverzeichnis. Es wird nicht geprüft, ob der so erzeugte Pfad tatsächlich existiert.

Syntax: **<path$>** = **ConstructPath$** (**pfadKonstante**, **tail$**)

pfadKonstante: Eine der Standard-Pfadkonstanten aus der Tabelle oben. Oder Null wenn der aktuelle Pfad anstelle eines Standardpfades verwendet werden soll.
tails$: Unterordner
<path$>: Variable vom Typ String oder besser String(200). GEOS-Pfade können bis zu 198 Zeichen lang werden.

Die Variable **fileError** wird gesetzt (Null, oder Fehlerwert).

Beispiele:

```
path$ = ConstructPath$ ( SP_DOCUMENT, "Bücher\\Karl May" )
path$ = ConstructPath$ ( SP_USER_DATA, "R-BASIC\\IMAGES" )

SetCurrentPath "C:\\GEOS\\DOCUMENT\\Bilder"
path$ = ConstructPath$ ( 0, "Urlaub" )
' --> liefert "C:\GEOS\DOCUMENT\Bilder\Urlaub"
```

GetStandardPath

GetStandardPath findet die zu einem Pfad gehörende Standardpfad-Konstante. Beschreibt der Pfad keinen Ordner im GEOS-System so liefert GetStandardPath den Wert Null. In diesem Fall wird die Variable fileError auf den Wert 3 (ERROR_PATH_NOT_FOUND) gesetzt.

Syntax: **<numVar>** = **GetStandardPath** (**"pfadstring"**)

"pfadstring": Bezeichnet den gewünschten Pfad. Es muss ein absoluter Pfad, einschließlich Laufwerksbuchstabe sein.
<numVar>: numerische Variable.

GetStandardPath findet prüft nicht, ob der Pfad wirklich existiert, sondern nur, ob er einen Standardpfad enthält. Die Variable **fileError** wird gesetzt (Null, oder Fehlerwert).

Beispiele. Wir nehmen an, GEOS befindet sich im Ordner "C:\GEOS". Der aktuelle Ordner sei "DOCUMENT\Bilder".

```
DIM stdPath

  stdPath = GetStandardPath ( "C:\\GEOS\\USERDATA\\R-BASIC" )
  ' -> liefert 19 (SP_USER_DATA)

  stdPath = GetStandardPath ( currentPath$ )
  ' -> liefert 5 (SP_DOCUMENT)

  stdPath = GetStandardPath ( "D:\\SOURCE\\PCGEOS" )
  ' -> liefert 0 (kein Standardpfad)
```

GetStandardPathTail$

GetStandardPathTail$ ist das Gegenstück zu GetStandardPath und findet den zum Standardpfad gehörenden Unterordner. Beschreibt der Pfad keine Ordner im GEOS-System gibt GetStandardPathTail$ den übergebenen Pfadstring komplett zurück. In diesem Fall wird die Variable fileError auf den Wert 3 (ERROR_PATH_NOT_FOUND) gesetzt.

Syntax: **<path$>** = **GetStandardPathTail$** (**"pfadstring"**)

"pfadstring": Bezeichnet den gewünschten Pfad. Es muss ein absoluter Pfad, einschließlich Laufwerksbuchstabe sein.
<path$>: Variable vom Typ String oder String(200). GEOS-Pfade können bis zu 198 Zeichen lang werden.

GetStandardPathTail$ findet prüft nicht, ob der Pfad wirklich existiert, sondern nur, ob er einen Standardpfad enthält. Die Variable **fileError** wird gesetzt (Null, oder Fehlerwert).

Beispiele. Wir nehmen an, GEOS befindet sich im Ordner "C:\GEOS". Der aktuelle Ordner sei "DOCUMENT\Bilder".

```
DIM path$ as String(200)

  path$ = GetStandardPathTail$ ( "C:\\GEOS\\USERDATA\\R-BASIC" )
  ' -> liefert "R-BASIC"

  path$ = GetStandardPathTail$ ( currentPath$ )
  ' -> liefert "Bilder"

  path$ = GetStandardPathTail$ (  "D:\\SOURCE\\PCGEOS" )
  ' -> liefert "D:\\SOURCE\\PCGEOS" (kein Standardpfad)
```

(Leerseite)

(Leerseite)

(Leerseite)