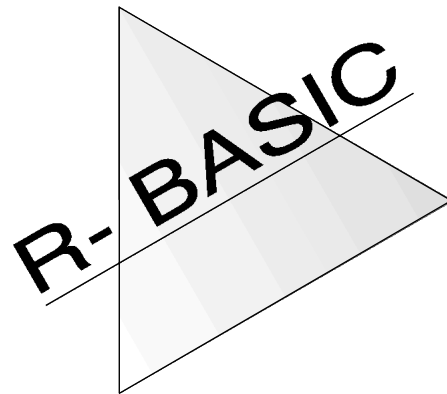


R-BASIC

Einfach unter PC/GEOS programmieren



Handbuch

Teil 3 - Spezielle Themen - Volume 2

Version 0.7 Beta

Inhaltsverzeichnis

Volume 1:

- 1 Formatierung von Zahlen
- 2 Verwendung von Schriften
- 3 Verwendung des Block-Grafik-Modus (Block-Font-Modus)
- 4 Arbeit mit Datum und Zeit
- 5 Der Datentyp HANDLE
- 6 Das Dateisystem
- 7 Arbeit mit Pfaden und Ordnern
- 8 Verwaltung von Dateien

Volume 2

- 9 Arbeit mit Dateien 9 - 1
 - 9.1 Überblick zur Dateiarbeit 9 - 1
 - 9.2 Datei-Attribute 9 - 3
 - 9.3 Anlegen, Öffnen und Schließen von Dateien 9 - 9
 - 9.4 Lesen und Schreiben (Byte-orientiert) 9 - 16
 - 9.5 Lesen und Schreiben (Text-orientiert) 9 - 21
 - 9.6 Sonstige Funktionen 9 - 24
- 10 Arbeit mit Laufwerken und Datenträgern 10 - 1
- 11 Portzugriffe 11 - 1
- 12 Speicherzugriff 12 - 1

R-BASIC Handbuch - Teil 3

Einfach unter PC/GEOS programmieren

Dieser dritte Teil des R-BASIC-Handbuchs beschäftigt sich mit speziellen Themengebieten, wie z.B. der Arbeit mit Dateien und welche Möglichkeiten bzw. Befehle Ihnen R-BASIC diesbezüglich bietet. Die Kapitel des dritten Teils sind weitgehend unabhängig voneinander geschrieben und Sie sollten sie lesen, wenn Sie das entsprechende Thema interessiert.

Verweise auf andere Kapitel beziehen sich, wenn nicht explizit anderes angegeben, immer auf den Teil 3 des Handbuchs.

Grundlegenden Befehle und Konzepte, die die R-BASIC Programmiersprache ausmachen finden Sie im Teil 1. Dort wird auch erklärt, wie man das R-BASIC Oberfläche benutzt und Sie erfahren vieles über Variablen, Schleifen, Verzweigungen, Unterprogramme und andere grundlegende Dinge.

Die Arbeit mit PC/GEOS Objekten und ihre Eigenschaften werden im zweiten Teil des Handbuchs besprochen.

9 Arbeit mit Dateien

9.1 Überblick zur Dateiarbeit

Für R-BASIC ist jede Datei zunächst eine einfache Abfolge von Bytes. Um auf die Daten in einer Datei zugreifen zu können, müssen Sie sie zuerst "öffnen" und nach Gebrauch wieder "schließen". Während die Datei geöffnet ist, erfolgt der Zugriff auf eine Datei über eine Variablen vom Typ **FILE**, die von **FileOpen** bzw. **FileCreate** geliefert wird. GEOS verwaltet außerdem einen "Dateizeiger", der die Position bestimmt, an der Daten gelesen oder geschrieben werden.

R-BASIC verfügt dabei über einige Möglichkeiten, die den meisten Programmiersprachen fehlen, wie z.B. das direkte Einfügen von Daten, ohne die darauf folgenden Daten zu überschreiben.

Hier finden Sie eine Übersicht über die Befehle zum Lesen aus und Schreiben von Daten in Dateien, die in diesem Kapitel behandelt werden.

Datei-Attribute (Abschnitt 9.2)

DOS-Attribute

FileGetAttrs, FileSetAttrs

Lesen und setzen die "Standard-Attribute" wie Archiv, schreibgeschützt usw.

FileGetTime, FileSetTime

Lesen und setzen das Datum der letzten Änderung.

FileSize

liefert die aktuelle Dateigröße.

GEOS-Attribute

Zusätzlich zu den DOS-Attributen haben GEOS-Dateien weitere Attribute.

R-BASIC unterstützt die folgenden Attribute:

Token	Dieses Attribut bestimmt das "Icon", das im GeoManager für diese Datei angezeigt wird. Befehle: FileSetToken, FileGetToken
Creator	Erzeuger. Dieses Attribut enthält das Token des Programms, das die Datei angelegt hat. Befehle: FileSetCreator, FileGetCreator Token und Creator werden in einer GeodeToken Struktur gespeichert.
CreationTime	Datum und Zeit, zu der die Datei angelegt wurde. Befehle: FileSetCreationTime, FileGetCreationTime
Usernotes	Die Benutzer-Notizen. Ein String mit bis zu 99 Zeichen. Befehle: FileSetUsernotes, FileGetUsernotes

Anlegen, Öffnen und Schließen von Dateien (Abschnitt 9.3)

FileOpen	Öffnet eine vorhandene Datei
FileCreate	Legt eine neu Datei an oder öffnet eine vorhandene
FileClose	Schließt eine Datei

Lesen und Schreiben von Daten - Byte-orientiert (Abschnitt 9.4)

FileRead, **FileWrite** und **FileInsert** sind Universal-Routinen. Sie arbeiten mit allen Dateien und allen Typen von Variablen, einschließlich Strings und Strukturen zusammen. Für die Arbeit mit Textdateien (lesen und schreiben von Text-Zeilen) gibt es zusätzlich dazu spezialisierte Routinen (Abschnitt 9.5).

FileRead	Liest eine bestimmte Anzahl von Bytes aus einer Datei.
FileWrite	Schreibt eine bestimmte Anzahl von Bytes in eine Datei, indem vorhandene Daten überschrieben werden. Bei Bedarf werden die Daten angehängt, d.h. dieDatei wird verlängert.
FileInsert	Fügt eine bestimmte Anzahl von Bytes in eine Datei ein, ohne das vorhandene Daten überschrieben werden. Die Datei wird dadurch automatisch verlängert.

Lesen und Schreiben von Daten - Text-orientiert (Abschnitt 9.5)

Diese Routinen sind auf Text-Zeilen, die durch ein **Zeilenendezeichen** abgeschlossen sind, spezialisiert. Das trifft für normale Text-Dateien zu. Zeilenendezeichen sind die ASCII-Codes 13 (Wagenrücklauf, Carriage retron, CR) bzw. 10 (Zeilenvorschub, LineFeed, LF) oder eine Kombination davon, üblicherweise die Folge 13, 10 (CRLF).

FileReadLine\$	Liest eine Textzeile aus einer Datei.
FileWriteLine	Schreibt eine Textzeile, indem vorhandene Daten überschrieben werden. Bei Bedarf wird die Textzeile angehängt, d.h. dieDatei wird verlängert.
FileInsertLine	Schiebt eine Textzeile in die Datei ein. Die Datei wird dadurch verlängert.
FileReplaceLine	Ersetzt eine Textzeile in einer Datei durch eine andere.

Sonstige Funktionen (Abschnitt 9.6)

FileGetPos	Liest den aktuellen Dateizeiger.
FileSetPos	Setzt den Dateizeiger.
FileResize	Ändert die Größe einer Datei, indem Daten eingefügt oder gelöscht werden.
FileTruncate	Schneidet die Datei ab.
FileCommit	Stellt sicher, dass eventuell vom System gepufferte Daten sofort, d.h. schon vor einem FileClose auf die Platte geschrieben werden.

9.2 Datei-Attribute

Neben dem Namen besitzt jede Datei eine Reihe von zusätzlichen Eigenschaften, die sogenannten "Attribute". Es gibt zwei Gruppen von Attributen, die "Standard-Attribute", einschließlich Dateigröße und dem Datum der letzten Änderung, die bereits aus alten DOS-Tagen stammen, und die "GEOS-Attribute", die nur für GEOS-Dateien vorhanden sind.

Die meisten der in diesem Kapitel besprochenen Funktionen können sowohl mit offenen Dateien (referenziert über eine FILE-Variable) als auch mit geschlossenen Dateien (referenziert über ihren Namen) umgehen.

Die Standard-Attribute

Die Standardattribute existieren für JEDE Datei und sind als einzelnen Bits in einem Byte definiert. Sie können mit **FileGetAttrs** gelesen und mit **FileSetAttrs** gesetzt werden. Die folgenden Attribute sind definiert, die Zahlen in der ersten Spalte sind die Bit-Nummer und der dazugehörige Wert.

Bit (Wert)	BASIC-Konstante	Bedeutung
Bit 0 (1)	FA_READ_ONLY	Read-Only. Die Datei ist schreibgeschützt.
Bit 1 (2)	FA_HIDDEN	Hidden. Die Datei ist versteckt.
Bit 2 (4)	FA_SYSTEM	System. Es ist eine wichtige System-Datei.
Bit 3 (8)	FA_VOLUME	Volume. Es ist keine Datei, sondern der Eintrag für die Datenträgerbezeichnung.
Bit 4 (16)	FA_SUBDIR	Subdir. Es ist keine Datei, sondern ein Verzeichnis (= Ordner).
Bit 5 (32)	FA_ARCHIVE	Archive. Die Datei wurde geändert. Dieses Bit wird bei jedem Schreibzugriff auf die Datei wieder gesetzt, so das Backup-Programm daran erkennen können, ob die Datei gesichert werden muss. Sie setzen dieses Bit nach dem Sicherungsprozess zurück.
Geos verwendet weiterhin:		
Bit 6 (64)	FA_LINK	Link. Dies ist kein Standard-Attribut. Ist dieses Bit gesetzt, handelt es sich nicht um eine echte Datei, sondern eine GEOS-internen Link auf eine Datei.

FileGetAttrs

Liefert die DOS-Standard-Attribute einer Datei. Die Attribute sind einzelnen Bits und können mit AND bzw. OR - Verknüpfungen abgefragt werden.

Format: x = **FileGetAttrs** (fileName\$)
oder: x = **FileGetAttrs** (fh)

fileName\$: Name der Datei. Pfadangaben im Namen sind zulässig.

fh: Variable (oder Funktion) vom Typ FILE. Bezeichnet die Datei.

Die Systemvariable **fileError** wird gesetzt oder gelöscht.

Beispiel:

```
DIM attrs AS word
attrs = FileGetAttrs ( "info.txt" )
IF attrs AND FA_READ_ONLY THEN Print "schreibgeschützt"
```

FileSetAttrs

Liefert die DOS-Standard-Attribute einer Datei. Die Attribute sind einzelnen Bits und können mit AND bzw. OR vernüpft werden.

Format: **FileSetAttrs** fileName\$, attrs

fileName\$: Name der Datei. Pfadangaben im Namen sind zulässig.

attrs: Neue Attribute

Die Systemvariable **fileError** wird gesetzt oder gelöscht. **FileSetAttrs** kann nicht mit einer offenen Datei verwendet werden.

Beispiele:

! Setzen von **FA_SYSTEM**, löschen aller anderen Attribute

```
FileSetAttrs "liste.txt" , FA_SYSTEM
```

! Setzen von **FA_SYSTEM** unter Beibehaltung der anderen Attribute

```
DIM attrs
```

```
attrs = FileGetAttrs ( "liste.txt" )
```

```
FileSetAttrs "liste.txt" , attrs OR FA_SYSTEM
```

FileGetTime

Liefert das Datum und die Uhrzeit der letzten Änderung einer Datei. Das Betriebssystem setzt diesen Wert jedes Mal, wenn der Dateiinhalt geändert wird.

Format: time = **FileGetTime** (fileName\$)
oder: time = **FileGetTime** (fh)

fileName\$: Name der Datei. Pfadangaben im Namen sind zulässig.
fh: Variable (oder Funktion) vom Typ FILE. Bezeichnet die Datei.
time: Eine Variable vom Typ **DateAndTime** (siehe Kapitel 4)

Die Systemvariable **fileError** wird gesetzt oder gelöscht.

Beispiel:

```
DIM time AS DateAndTime
time = FileGetTime "info.txt"
Print "Geändert am: "; FormatDate$(time);
Print "um: "; FormatTime$(time)
```

FileSetTime

Verändert das Datum und die Uhrzeit der letzten Änderung einer Datei.
Achtung! Sie überschreiben hiermit den Wert, den das Betriebssystem automatisch vergibt. Wird die Datei anschließend nochmals geändert, überschreibt das Betriebssystem den Wert erneut.

Format: **FileSetTime** fileName\$, time
oder: **FileSetTime** fh , time

fileName\$: Name der Datei. Pfadangaben im Namen sind zulässig.
fh: Variable (oder Funktion) vom Typ FILE. Bezeichnet die Datei.
time: Eine Variable vom Typ DateAndTime

Fehlerbedingung: Die Systemvariable **fileError** wird gesetzt oder gelöscht.

Beispiel:

```
DIM time AS DateAndTime
time = FileGetTime ( "info.txt" )
time.year = 2001
FileSetTime "info.txt", time
```

FileSize

Liefert die aktuelle Größe der Datei in Bytes. Der zurückgegebene Wert liegt im Bereich von 0 bis 4294967295, da Dateien maximal 4 GByte groß werden können.

Format: x = **FileSize** (fileName\$)

oder: x = **FileSize** (fh)

fileName\$: Name der Datei. Pfadangaben im Namen sind zulässig.

fh: Variable (oder Funktion) vom Typ FILE. Bezeichnet die Datei.

Die Systemvariable **fileError** wird gesetzt oder gelöscht.

GEOS-Attribute

Zusätzlich zu den Standard-Attributen haben GEOS-Dateien weitere Attribute. R-BASIC unterstützt die wichtigen Attribute: **Token** (Anzeige-Icon), **Creator** (Icon des zugehörigen Programms), **CreationTime** (Datum der Datei-Erstellung) und **UserNotes** (Benutzer-Notizen). Diese werden im Dateikopf, den ersten 256 Byte der Datei gespeichert. Bei Verzeichnissen befinden sich diese Attribute in der @dirname.000 - Datei.

CreationTime wird in einer **DateAndTime** Struktur (siehe Kapitel 4) gespeichert. **UserNotes** ist ein String mit bis zu 99 Zeichen.

Token und **Creator** werden in einer Struktur gespeichert, die **GeodeToken** heißt und folgendermaßen definiert ist. Das Bild dazu befindet sich in der TokenDB-Datei.

```
STRUCT GeodeToken
    manufid AS WORD 'Manufacturer ID (Hersteller-Identifikation)
    tokenChars AS STRING(4)
END STRUCT
```

Eine fehlerhafte Belegung der Werte für Token oder Creator kann die Arbeit mit der Datei unmöglich machen. Setzen Sie in diesem Fall einfach wieder den korrekten Wert.

Tritt ein Fehler auf, z.B. weil die Datei nicht gefunden wird oder weil das Attribut nicht unterstützt wird, (weil es eine DOS-Datei ist), wird die Systemvariable **fileError** gesetzt. Im Erfolgsfall wird die **fileError**-Variable zurückgesetzt (d.h. mit Null belegt)

Die Bedeutung der Parameter der folgenden Übersicht und Beispiele sind weiter unten zu finden.

FileGetToken, FileSetToken

Liest bzw setzt das "Token" der Datei.

Format: token = **FileGetToken** (fileName\$) ' Auslesen des Token
oder: token = **FileGetToken** (fh)

Format: **FileSetToken** fileName\$, token ' Setzen des Token
oder: **FileSetToken** fh, token

FileGetCreator, FileSetCreator

Liest (FileGetCreator) oder setzt (FileSetCreator) das Creator-Token.

Format: token = **FileGetCreator** (fileName\$) ' Auslesen des Creator-Token
oder: token = **FileGetCreator** (fh)

Format: **FileSetCreator** fileName\$, token ' Setzen des Creator-Token
oder: **FileSetCreator** fh , token

FileGetCreationTime, FileSetCreationTime

Liest oder verändert das Datum, an dem die Datei angelegt wurde.

Format: time = **FileGetCreationTime** (fileName\$) ' Auslesen der
Erstellungszeit
oder: time = **FileGetCreationTime** (fh)

Format: **FileSetCreationTime** fileName\$, time ' Setzen der Erstellungszeit
oder: **FileSetCreationTime** fh , time

FileGetUsernotes, FileSetUsernotes

Liest oder verändert die Benutzer-Notizen der Datei.

Format: notes\$ = **FileGetUsernotes** (fileName\$) ' Auslesen der
Benutzernotizen
oder: notes\$ = **FileGetUsernotes** (fh)

Format: **FileSetUsernotes** fileName\$, notes\$ ' Setzen der Benutzernotizen
oder: **FileSetUsernotes** fh, notes\$

Angaben zu den Parametern und Rückgabewerten:

token:	Eine Variable vom Typ "GeodeToken" Die Set- Routinen akzeptieren auch Funktionen, die eine GeodeToken-Struktur zurückgeben
fileName\$:	Name der Datei Stringausdrücke, auch mit Pfadangaben im Namen, sind zulässig.
fh:	Variable (oder Funktion) vom Typ FILE. Bezeichnet die Datei.
time:	Eine Variable vom Typ "DateAndTime" Die Set- Routinen akzeptieren auch Funktionen, die eine DateAndTime-Struktur zurückgeben
notes\$:	Ein String mit bis zu 99 Zeichen. Für FileSetUsernotes sind Stringausdrücke zulässig. Ist der String zu lang tritt ein Lautzeitfehler auf und das Programm wird beendet.

Beispiel:

```
DIM token as GeodeToken
DIM time as DateAndTime
SetStandardPath SP_SYSTEM

token = FileGetToken "geos.geo"
Print token.tokenchars; token.manufid      ' manufid = Manufacturer ID

token = FileGetCreator "geos.geo"
Print token.tokenchars; token.manufid

time = FileGetCreationTime "geos.geo"
Print FormatDate$(time); FormatTime$(time)
```

9.3 Anlegen, Öffnen und Schließen von Dateien

Um mit einer Datei arbeiten zu können, müssen Sie sie zuerst "öffnen" (was beim Anlegen automatisch geschieht) und nach Gebrauch wieder "schließen". Dabei legen Sie fest, ob Sie Daten in die Datei schreiben (**write**), aus ihr lesen (**read**) oder beides (read/write) wollen (Parameter 'accessFlags\$'). Read/Write ist der Standard. Da GEOS ein MultiThread-System ist, kann es passieren, dass andere Programme **gleichzeitig** auf diese Datei zugreifen wollen. Deswegen müssen Sie festlegen ob und wie andere Programme gleichzeitig auf Ihre Datei zugreifen dürfen (**read, write, read/write** bzw. **gar nicht**). Der Standard ist, keinerlei Zugriff zu erlauben (Parameter 'alienFlags\$'). Sie sollten davon nur abweichen, wenn es unbedingt erforderlich ist, da das System in diesem Fall viele Daten zwischenspeichern muss, was den Systemspeicher sehr belasten kann.

FileCreate

Legt eine Datei auf dem Datenträger an und öffnet sie. Existiert die Datei schon, kann sie auch verwendet (Daten bleiben erhalten) oder verworfen werden.

Format fh = **FileCreate** fileName\$ [, accessFlags\$ [, alienFlags\$]]

fh: Variable vom Typ FILE. fh enthält dann eine Referenz auf die Datei und wird für alle anderen Dateioperationen benötigt.

fileName\$: Name der Datei.

accessFlags\$: Zugriffs-Flags (optional): Zeichenkette, bestehend aus einer Kombination der Buchstaben "otn g rw x", die den Zugriff auf die Datei festlegen. Der Standard (accessFlags\$ nicht angegeben) ist "n rw": Neue Datei zum Lesen und Schreiben anlegen.

alienFlags\$: Fremdzugriffs-Flags (optional): Zeichenkette, bestehend aus einer Kombination der Buchstaben "rw", die den Zugriff auf die Datei festlegen. Der Standard (alienFlags\$ nicht angegeben) ist "": Keine Fremdzugriffe erlaubt.

Der Dateizeiger ist nach dem Anlegen einer Datei immer auf die Position 0 gesetzt, auch wenn sie schon Daten enthält. Wollen Sie Daten anhängen, müssen Sie ihn zunächst mit der Anweisung **FileSetPos** ans Datei-Ende setzen.

Die Systemvariable **fileError** wird gesetzt oder gelöscht. Beispiele und eine ausführliche Beschreibung der Flagzeichen finden Sie weiter unten.

Mögliche Fehlerbedingungen für FileCreate sind unter anderem:

- Die Datei existiert, aber es wurde 'n' (nur neu anlegen) angegeben.
- Die Datei existiert als DOS-Datei, aber es wurde 'g' (GEOS Daten-Datei anlegen) angegeben.
- Die Datei existiert und es wurde 'o' (open) bzw. 't' (truncate, abschneiden) angegeben, aber sie ist von einem anderen Programm geöffnet, und dieses verweigert den Zugriff.

FileOpen

Öffnet eine Datei. Die Datei muss schon auf dem Datenträger vorhanden sein.

Format: fh = **FileOpen** fileName\$ [, accessFlags\$ [, alienFlags\$]]

fh Variable vom Typ FILE. fh enthält dann eine Referenz auf die Datei und wird für alle anderen Dateioperationen benötigt.

fileName\$: Name der Datei.

accessFlags\$: Zugriffs-Flags, optional): Zeichenkette, bestehend aus einer Kombination der Buchstaben "rwx", die den Zugriff auf die Datei festlegen. Der Standard (accessFlags\$ nicht angegeben) ist "rw": Datei zum Lesen und Schreiben öffnen.

alienFlags\$: Fremdzugriffs-Flags, optional): Zeichenkette, bestehend aus einer Kombination der Buchstaben "rw", die den Zugriff auf die Datei festlegen. Der Standard (alienFlags\$ nicht angegeben) ist "": Leerstring, keine Fremdzugriffe erlaubt.

Der Dateizeiger ist nach dem Öffnen einer Datei immer auf die Position 0 gesetzt. Wollen Sie Daten anhängen, müssen Sie ihn zunächst mit der Anweisung **FileSetPos** ans Datei-Ende setzen.

Die Systemvariable **fileError** wird gesetzt oder gelöscht. Beispiele und eine ausführliche Beschreibung der Flagzeichen finden Sie weiter unten.

Mögliche Fehlerbedingungen für **FileOpen** sind unter anderem:

- Die Datei existiert nicht.
- Die Datei existiert, aber sie ist von einem anderen Programm geöffnet, und dieses verweigert den Zugriff.

Flagzeichen für FileCreate und FileOpen

Die Befehle **FileCreate** und **FileOpen** erwarten ein oder zwei Zeichenketten, in denen durch einzelnen Buchstaben symbolisiert ist, was Sie wollen. Das Setzen einzelner Bits oder, wie hier, Zeichen, wird üblicherweise als das Setzen von "**Flags**" (Flaggen) bezeichnet.

accessFlags\$ (Zugriffs-Flaggen-Zeichen) ist eine Zeichenkette, die bestimmt, wie die Datei angelegt oder geöffnet werden soll und ob auf die Datei lesend und/oder schreibend zugegriffen werden soll.

Mögliche **accessFlags** für **FileOpen**: **r, w, x** (x nur für Profis)

Mögliche **accessFlags** für **FileCreate**: **g, n, t, o, r, w, x** (x nur für Profis)

alienFlags\$ (Fremd-Zugriffs-Flaggen-Zeichen) ist eine Zeichenkette, welche die Zugriffsrechte für andere Programme bestimmt, während die Datei offen ist. Wird **alienFlags\$** nicht angegeben, so wird "" (leer, keine Zugriffsrechte) angenommen. Sie sollten davon nur abweichen, wenn es unbedingt erforderlich ist, da das System in diesem Fall viele Daten zwischenspeichern muss, was den Systemspeicher sehr belasten kann.

Mögliche **alienFlags** für **FileOpen** und **FileCreate**: **r, w**

Zugriffs-Flags (accessFlags\$) für FileCreate und FileOpen

r "read": Aus der Datei kann gelesen werden.

w "write" In die Datei kann geschrieben werden.

Wird weder **'r'** noch **'w'** angegeben, wird **'rw'** angenommen. Dies ist auch der Standard, wenn **accessFlags\$** nicht angegeben wird.

Zugriffs-Flags (**accessFlags\$**) nur für **FileCreate**

g "GEOS": Es wird eine GEOS-Daten-Datei angelegt. Der Dateiname muss den GEOS-Namenskonventionen (max 32 Zeichen) entsprechen. Wird **'g'** nicht angegeben, wird eine DOS-Datei angelegt. Der Name muss dann der Konvention 8.3 entsprechen.

n "neu": Nur Neuanlegen erlaubt. Es ist ein Fehler, wenn die Datei schon existiert, die Systemvariable **fileError** wird entsprechend gesetzt.

o "open": Existiert die Datei schon, wird sie normal geöffnet. Die vorhandenen Daten bleiben erhalten.

t "truncate" (= abschneiden): Existiert die Datei schon, wird sie abgeschnitten. Die vorhandenen Daten gehen verloren.

Wird weder **'n'**, **'o'** noch **'t'** angegeben, wird **'n'** angenommen. Dies ist auch der Standard, wenn "accessFlags\$ nicht angegeben wird.

Beispiele:

FileOpen "info.txt", "r"	Nur Lesezugriff zugelassen Kein Fremdzugriff, da keine alienFlags\$ angegeben.
FileCreate "info.txt", "orw"	Öffnen einer vorhandenen Datei oder Anlegen einer Neuen mit Schreib- und Lesezugriff. Eventuell vorhandene Daten bleiben erhalten. Kein Fremdzugriff, da keine alienFlags\$ angegeben.
FileCreate "info.txt", "gtw"	Öffnen einer vorhandenen GEOS-Daten-Datei oder Anlegen einer Neuen zum Schreibzugriff, ohne Lesezugriff. Eventuell vorhandene Daten werden gelöscht. Kein Fremdzugriff, da keine alienFlags\$ angegeben.

Fremd-Zugriffs-Flags (**alienFlags\$**) für FileCreate und FileOpen

- r** "read": Andere Programme können aus der Datei lesen
- w** "write" Andere Programme können in die Datei schreiben.

Wird weder 'r' noch 'w' angegeben, wird "" (keine Zugriffsrechte) angenommen. Dies ist auch der Standard, wenn **alienFlags\$** nicht angegeben wird.

Tipps:

- Für die Flags sind Groß- und Kleinbuchstaben erlaubt.
- Die Reihenfolge der Flag-Zeichen ist egal.
- String-Ausdrücke sind erlaubt.
- Überflüssige bzw. ungültige Zeichen werden ignoriert. Sie können den Flags-String so optisch strukturieren. Zu lange Zeichenketten (mehr als 32 Zeichen) können aber zu einem Laufzeitfehler führen.
- Verwenden Sie statt der Buchstaben keine Worte wie z.B. "write" statt "w". Sie würden im Beispiel die Datei zum Schreiben und Lesezugriff öffnen, da **write** die Buchstaben **w** und **r** enthält.

Sonstige **accessFlags**, nur für Profis:

- x** "eXtended": Erweiterter Zugriff, z.B. auf den Header einer GEOS-Datei. **Achtung!** Ungültige Änderungen im Header können die Arbeit mit der Datei unmöglich machen! GEOS kann die Datei in bestimmten Fällen auch nicht mehr löschen. Sie sollten hier ganz genau wissen, was Sie tun. Der Programmierer von R-BASIC übernimmt **keinerlei Haftung!**

Beispiele für FileOpen und FileCreate:

Die folgenden Beispiele setzen voraus, dass eine FILE-Variable folgendermaßen definiert ist:

```
DIM fh as FILE
```

Anlegen einer DOS-Datei zum Lesen und schreiben. Existiert die Datei schon, soll die Variable fileError gesetzt werden. Fremdprogramme sollen während dessen keinen Zugriff haben.

```
fh = FileCreate "Info.TXT"
```

Diese Anweisung ist identisch mit: FileCreate "Info.TXT", "rw", ""

Anlegen einer GEOS-Daten-Datei zum Lesen und schreiben. Es muss das Flag "g" angegeben werden, damit eine GEOS-Datei erzeugt wird. Existiert die Datei schon, soll die Variable fileError gesetzt werden. Fremdprogramme sollen während dessen Lese-Zugriff haben. Der Parameter "accessFlags\$" muss angegeben werden um "alienFlags\$" angeben zu können.

```
fh = FileCreate "Meine Daten" , "g", "r"
```

Anlegen einer GEOS-Daten-Datei nur zum Schreiben. Existiert die Datei schon, wird sie geöffnet und der Inhalt verworfen. Fremdprogramme sollen während dessen keinen Zugriff haben.

```
fh = FileCreate "Mein Daten-Logbuch" , "gtw"
```

Anlegen einer DOS-Datei zum Lesen und Schreiben. Existiert die Datei schon, wird sie geöffnet, der Inhalt bleibt erhalten. Fremdprogramme sollen während dessen keinen Zugriff haben.

```
fh = FileCreate "Data.dat" , "o"
```

Öffnen einer Datei zum Lesen und schreiben. Fremdprogramme sollen während dessen keinen Zugriff haben.

```
fh = FileOpen "Info.TXT"
```

Diese Anweisung ist identisch mit: FileOpen "Info.TXT", "rw", ""

Öffnen einer GEOS-Daten-Datei zum Lesen und schreiben. Beachten Sie, dass das Flag 'g' nicht angegeben wird, R-BASIC erkennt selbstständig, dass es sich um eine GEOS-Datei handelt. Fremdprogramme sollen während dessen Lese-Zugriff haben. Der Parameter "accessFlags\$" muss angegeben werden um "alienFlags\$" angeben zu können.

```
fh = FileOpen "Meine Daten" , "rw", "r"
```

Öffnen einer Datei nur zum Lesen. Fremdprogramme sollen während dessen keinen Zugriff haben.

```
fh = FileOpen "Daten.TXT" , "r"
```

Tipp:

Wenn Sie nicht sicher sein können, ob die Datei schon existiert, verwenden Sie `FileCreate` mit den `accessFlags` `o` (open) oder `t` (truncate). Um eventuell vorhandene Daten nicht zu verlieren, verwenden Sie das Flag `o`.

```
fh = FileCreate "Data.dat" , "o"
```

FileClose

Schließt eine offene Datei. Alle evt. noch im Hauptspeicher befindlichen Daten werden auf den Datenträger geschrieben. Nach dem Schließen haben andere Programme wieder den vollen Zugriff auf die Datei, die von **FileOpen** bzw. **FileCreate** gesetzten Restriktionen sind aufgehoben.

Format **FileClose** fh
fh: Variable (oder Funktion) vom Typ FILE. Bezeichnet die Datei.

Die Systemvariable **fileError** wird gesetzt oder gelöscht.

Beispiel:

```
DIM fh as FILE
fh = FileOpen "info.txt"
...
FileClose fh
```

NullFile

Liefert eine "leere" Datei-Variable zurück, dient also zum Löschen einer Datei-Variablen.

Format: dateiVariable = **NullFile**()
 Die Klammern sind erforderlich.
dateiVariable: Variable vom Typ FILE.

Nachdem die Datei geschlossen wurde (FileClose), sollten Sie der Dateivariablen mit Hilfe der Funktion NullFile() die Information "keine Datei" zuweisen.

```
DIM fh as FILE
....
....
FileClose fh
fh = NullFile ()
```

Sie können dann prüfen, ob eine Datei noch offen ist:

```
IF fh <> NullFile() THEN ...
```

Verwenden Sie im Falle eines Programmierfehlers eine schon geschlossene Datei nochmals, z.B. in der Reihenfolge

```
FileClose fh
x = FileRead ( fh )
```

gibt es einen BASIC Laufzeitfehler und das Programm wird beendet.

9.4 Lesen und Schreiben (Byte-orientiert)

FileRead, **FileWrite** und **FileInsert** sind Universal-Routinen. Sie arbeiten mit allen Dateien und allen Typen von Variablen zusammen, einschließlich Strings und Strukturen. Für die Arbeit mit Textdateien (lesen und schreiben von Text-Zeilen) gibt es zusätzlich darauf spezialisierte Routinen, siehe Kapitel 9.5.

FileRead

Liest eine bestimmte Anzahl von Bytes aus einer Datei. Der Dateizeiger wird hinter die gelesenen Daten gesetzt. Die Daten werden entsprechend dem Typ des Ausdrucks, in dem **FileRead** vorkommt, interpretiert.

Format: var = **FileRead** (fh , size [, signed])

- var: Variable von beliebigem Typ. Die gelesenen Daten werden in diese Variable kopiert.
- fh: Variable (oder Funktion) vom Typ FILE. Bezeichnet die Datei.
- size: Anzahl der zu lesenden Bytes. Bei numerischen Daten bestimmt size den Datentyp (Byte, Word, DWord, Integer, LONGINT, Real).
Zulässige (Grenz~) Werte für size: 1 <= size <= 16384 (16 kByte)
- signed: (optional): TRUE oder FALSE (default: FALSE).
Nur für numerische Daten der Größe 2 Byte oder 4 Byte: Gelesene Daten als vorzeichenlos (signed = FALSE oder signed nicht angegeben) oder vorzeichenbehaftet (signed = TRUE) ansehen.
-

Die Systemvariable **fileError** wird gesetzt oder gelöscht.

Beispiele:

y = **FileRead** (fh, 2) ' Lesen eines Word-Wertes (2 Byte)

y = **FileRead** (fh, 2, TRUE) ' Lesen eines Integer-Wertes (2 Byte)

text\$ = **FileRead** (fh, 100) ' 100 Byte lesen und als Text ansehen.

- ! Ein String endet, wenn eine binäre Null (ASCII-Code Null) gelesen wird.
- ! Len(stringVariable) kann daher kleiner als 'size' sein.
- ! Trotzdem werden 'size' Bytes gelesen.

DIM time as **DateAndTime**

time = **FileRead** (fh, SizeOf(time)) ' Lesen einer Struktur

FileWrite

Schreibt eine bestimmte Anzahl von Bytes in eine Datei, indem vorhandene Daten überschrieben werden. Bei Bedarf werden die Daten angehängt, d.h. die Datei wird verlängert. Der Dateizeiger wird hinter die geschriebenen Daten gesetzt.

Format: **FileWrite** fh , <expression> , size [, signed])

- fh: Variable (oder Funktion) vom Typ FILE. Bezeichnet die Datei.
<expression>: Ausdruck von beliebigem Typ. Das Ergebnis des Ausdrucks (z.B. eine Zahl, eine Struktur oder ein Text) wird in die Datei kopiert.
size: Anzahl der zu schreibenden Bytes. Bei numerischen Daten bestimmt size den Datentyp (Byte, Word, DWord, Integer, LONGINT, Real), in den die Zahl vor dem Schreiben konvertiert wird.
Zulässige (Grenz~) Werte für size: 1 <= size <= 16384 (16 kByte)
signed: (optional): TRUE oder FALSE (default: FALSE).
Nur für numerische Daten der Größe 2 Byte oder 4 Byte: Die Daten als vorzeichenlos (signed = FALSE oder signed nicht angegeben) oder vorzeichenbehaftet (signed = TRUE) schreiben.
-

Die Systemvariable **fileError** wird gesetzt oder gelöscht.

Beispiele:

FileWrite (fh, n, 2) ' Schreiben eines Word-Wertes
y = **FileWrite** (fh, 110, 2, TRUE) ' Schreiben eines Integer-Wertes

FileWrite (fh, text\$, 100) ' 100 Byte als Text schreiben.
! ist text\$ länger, wird der String abgeschnitten
! es wird keine Text-Ende-Kennung (Null) geschrieben
! ist text\$ kürzer, wird mit Nullen aufgefüllt

DIM time as **DateAndTime**

time = **FileWrite** (fh, time, SizeOf(time)) ' Schreiben einer Struktur

FileInsert

Fügt eine bestimmte Anzahl von Bytes in eine Datei ein, ohne das vorhandene Daten überschrieben werden. Die Datei wird dadurch automatisch verlängert. Der Dateizeiger wird hinter die geschriebenen Daten gesetzt.

Format: **FileInsert** fh , <expression> , size [, signed])
fh , <expression> , size [, signed]) siehe FileWrite.

Die Systemvariable **fileError** wird gesetzt oder gelöscht.

Beispiele: siehe **FileWrite**.

Spezielle Hinweise zum Speichern von File-, Handle- und Objekt-Variablen in Dateien

FileRead, **FileWrite** und **FileInsert** können mit File-, Handle und Objekt-Variablen bzw. Ausdrücken umgehen. Verwenden Sie als Datengröße den Wert 10 (= **SizeOf(File)**, **SizeOf(Handle)**, **SizeOf(Object)**).

Es ist jedoch im Normalfall nicht sinnvoll, diese Werte in einer Datei zu speichern, da die enthaltenen Werte nur zeitlich begrenzt gültig sind.

- **File-Variablen** sind nur solange gültig, wie die Datei geöffnet ist. Wird die Datei nach dem Schließen erneut geöffnet, ist der Inhalt der Dativariablen **NICHT** mit dem vom ersten Mal identisch.
- **Handle-Variablen** sind nur solange gültig, wie das von Ihnen bezeichnete Objekt bzw. die dahinter stehende Datenstruktur vorhanden ist. Beispielsweise gilt das von **FileFindFirst\$** gelieferte Handle nur so lange, bis **FileFindDone** gerufen wird. Ein erneutes **FileFindFirst\$** liefert eine anderes Handle, auch wenn es im gleichen Verzeichnis sucht.

Die **Ausnahme** sind Handles, die von der VMFiles-Library geliefert werden und sich auf Datenstrukturen in einer VM-Datei beziehen. Diese müssen üblicherweise in der VM-Datei selber gespeichert werden und sind so lange gültig, wie die Datenstrukturen selbst in der VM-Datei sind.

- **Objekt-Variablen** sind nur für die Laufzeit des Programms, maximal bis zum Ende der GEOS-Sitzung, gültig. Hier sind auch Abweichungen (kürzere Gültigkeitsdauern) möglich, z.B. wenn das zugehörige Objekt gelöscht oder im Objektbaum verschoben wird.

Spezielle Hinweise zum Speichern von numerischen Werten in Dateien

R-BASIC kennt 3 vorzeichenlose (BYTE, WORD und DWORD) und 3 vorzeichenbehaftete (INTEGER, LONGINT und REAL) numerische Datentypen.

Um Zahlen mit diesen Datentypen sowohl in Dateien schreiben als auch aus Ihnen lesen zu können, gelten folgende Konventionen:

Lesen von numerischen Werten:

- Der **size**-Parameter von **FileRead** bestimmt, wie viele Bytes gelesen und wie sie interpretiert werden (d.h. welchem Datentyp sie entsprechen). Zulässig sind die Werte

- 1 Lesen eines Byte
- 2 Lesen eines Word oder Integer
- 4 Lesen eines DWord oder LongInt
- 10 Lesen eines Real-Wertes

Andere Werte können zu unerwarteten Ergebnissen führen. Die ersten beiden gelesenen Bytes werden als Word (bzw. Integer) interpretiert, die restlichen Bytes werden verworfen.

- Der **signed**-Parameter bestimmt für die 2 und 4-Byte Datentypen, ob die Bytes als vorzeichenlose Zahl (**word** bzw. **dword**, signed=FALSE, default-Wert) oder als vorzeichenbehaftete Zahl interpretiert werden (**integer** bzw. **longint**, signed=TRUE, signed muss angegeben werden). Verwenden Sie am Besten den gleichen Wert für **signed**, den Sie auch beim Schreiben verwendet haben.
- Nachdem die Daten gelesen und interpretiert wurden, werden sie von FileRead in eine Real-Zahl konvertiert, so dass FileRead innerhalb von beliebigen Ausdrücken wie jede andere Funktion verwendet werden kann.

Beispiele:

```
DIM x, z as Real
```

```
DIM n as Word
```

```
n = FileRead ( fh, 2 )           ' Lesen eines Word
x = FileRead ( fh, 2, TRUE )     ' Lesen eines Integer-Wertes, aber speichern als Real
x = FileRead ( fh, 4, TRUE )     ' Lesen eines LONGINT
x = FileRead ( fh, 4 )           ' Lesen eines DWord
z = 2.7 * FileRead ( fh, 2 ) + FileRead ( fh, 10 )
```

Schreiben von numerischen Werten:

Die Parameter-Konventionen sind denen von **FileRead** analog.

- Der **size**-Parameter von **FileWrite** bzw. **FileInsert** wie viele Bytes geschrieben werden und in welchen Datentyp die Zahl vorher konvertiert werden soll. Zulässig sind die Werte:
 - 1 Schreiben eines Byte
 - 2 Schreiben eines Word oder Integer
 - 4 Schreiben eines DWord oder LongInt
 - 10 Schreiben eines Real-WertesBei ungültigen Werten wird zunächst ein Word (bzw. Integer) geschrieben und der Rest mit Nullen aufgefüllt.
- Der **signed**-Parameter bestimmt für die 2 und 4-Byte Datentypen, ob die Bytes als vorzeichenlose Zahl (**word** bzw. **dword**, signed=FALSE, default-Wert) oder als vorzeichenbehaftete Zahl geschrieben werden (**integer** bzw. **longint**, signed=TRUE, signed muss angegeben werden).

Dazu wird der von <expression> gelieferte (REAL~) Wert zunächst in den entsprechenden Datentyp konvertiert und dann in die Datei geschrieben.

- Der Unterschied zwischen vorzeichenlosen und vorzeichenbehafteten Werten ist, dass entsprechend den R-BASIC Konventionen bei vorzeichenbehafteten Werten (**signed** = TRUE) bei einer Zahlenbereichsüberschreitung (z.B. 100 000 für Integer) eine Begrenzung auf den größten bzw. kleinsten Wert erfolgt, während bei vorzeichenlosen Werten eine Modulo-Operation erfolgt, d.h. man fängt bei zu großen oder zu kleinen Werten einfach wieder von vorne an.

Beispiele:

```
DIM x as Real
DIM n as Word
```

```
FileWrite ( fh, n, 2 )           ' Schreiben eines Word
FileWrite ( fh, x, 2, TRUE )    ' x runden und Schreiben als Integer-Wert
FileWrite ( fh, x, 4, TRUE )    ' Schreiben eines LONGINT, x vorher runden
FileWrite ( fh, n, 4 )         ' Schreiben eines DWord, n vorher in DWord konvertieren
```

9.5 Lesen und Schreiben (Text-orientiert)

R-BASIC verfügt über Spezialbefehle zum Lesen und Schreiben von Zeilen aus Textdateien. Diese Befehle arbeiten nur mit String-Variablen zusammen und werden in diesem Abschnitt erklärt. Universelle Lese- und Schreibbefehle finden Sie im Abschnitt 9.4.

FileReadLine\$, FileWriteLine, FileInsertLine und **FileReplaceLine** sind auf Text-Zeilen, die durch ein Zeilenendezeichen abgeschlossen sind, spezialisiert. Das trifft für normale Text-Dateien zu. Zeilenendezeichen sind die ASCII-Codes 13 (Wagenrücklauf, Carriage return, CR) bzw. 10 (Zeilenvorschub, LineFeed, LF) oder eine Kombination davon, üblicherweise die Folge 13, 10 (CRLF).

Hinweis: Ein Text-Editor fügt häufig zur Gewährleistung der Lesbarkeit von Texten automatische (nur am Bildschirm vorhandene) Zeilenumbrüche ein. Die R-BASIC "Text-Zeilen" erscheinen daher als "Absätze" in einem Text-Editor.

FileReadLine\$

Liest eine Textzeile aus einer Datei. Es werden maximal 1024 Zeichen gelesen (BASIC-Begrenzung für Strings). Ist die Zeile länger (d.h. es wurde keine Zeilenendekennung gelesen), so wird fileError auf -11 (LINE_TO_LONG) gesetzt. Ihr Programm kann dann entsprechend reagieren.

Format: z\$ = **FileReadLine\$** (fh [, mode])

- fh: Variable (oder Funktion) vom Typ FILE. Bezeichnet die Datei.
mode: (optional): Behandlung der Zeilen-Ende-Zeichen
- 0: (default) Zeilen-Ende-Zeichen abschneiden
 - 1: Zeilen-Ende-Zeichen durch CR (ASCII-Code 13) ersetzen. Dieser Code wird von GEOS-Text-Objekten als Zeilenendezeichen verwendet.
 - 2: Zeilen-Ende-Zeichen entfernen und in jedem Fall ein CR-Zeichen (Code 13) anhängen, auch wenn kein Zeilenendezeichen vorhanden war (z.B. am Ende einer Datei)
 - 3: Text nicht ändern, Zeilenendezeichen bleiben erhalten.
-

Die Systemvariable **fileError** wird gesetzt oder gelöscht.

Hinweise:

- Am Dateiende wird eine fehlende Zeilenendekennung akzeptiert und fileError auf Null (Kein Fehler) gesetzt.
- Passt der gelesene String nicht in die Variable auf der linken Seite der Zuweisung wird ein Laufzeitfehler erzeugt und das Programm beendet.
- Lesen über das Dateiende hinaus setzt fileError auf 128 (SHORT_READ_WRITE), siehe Beispiel 2

Beispiele:

```
z$ = FileReadLine$ (fh)           ' eine Zeile lesen

WHILE   fileError = 0
          z$ = FileReadLine$ (fh)       ' eine Zeile lesen
          IF fileError <> 0 THEN Print z$ ' und ausgeben
          WEND

z$ = FileReadLine$ (fh, 3)         ' eine Zeile incl. CR/LF lesen
```

FileWriteLine

Schreibt eine Textzeile, indem vorhandene Daten überschrieben werden. Bei Bedarf wird die Textzeile angehängt, d.h. die Datei wird verlängert.

Format **FileWriteLine** fh, zeile\$ [, mode])

fh: Variable (oder Funktion) vom Typ FILE. Bezeichnet die Datei.
zeile\$: zu schreibender Text
mode: (optional): Behandlung der Zeilen-Ende-Zeichen
 0: (default) Zeilen-Ende-Zeichen-Folge CRLF (13, 10) anhängen
 1: CR-Codes (ASCII-Code 13) durch CRLF (Folge 13, 10) ersetzen.
 Dadurch werden Texte, die von GEOS-Text-Objekten kommen, zur
 Verwendung in DOS-Dateien angepasst.
 2: CR-Codes durch CRLF ersetzen, wie mode 1. Unterschied: War
 am Textende kein CR-Code, so wird ein zusätzliches CRLF
 angehängt.
 3: Text nicht ändern, Zeilenendezeichen bleiben erhalten.

Die Systemvariable **fileError** wird gesetzt oder gelöscht.

Beispiele:

```
FileWriteLine fh, "Hallo Welt"           ' CRLF automatisch anhängen

FileWriteLine fh, "Hallo Welt\r", 2      ' '\r' (CR) durch CRLF ersetzen
FileWriteLine fh, "\r\r\r", 2           ' 3 Leerzeilen
FileWriteLine fh, "Text geschrieben von R-BASIC", 2
FileWriteLine fh, "Letzte Zeile", 3     ' kein CRLF anhängen
```

FileInsertLine

Schiebt eine Textzeile in die Datei ein. Die Datei wird dadurch verlängert.

Format **FileInsertLine** fh, zeile\$ [, mode])
 fh, zeile\$ [, mode]): siehe **FileWriteLine**

Die Systemvariable **fileError** wird gesetzt oder gelöscht.

FileReplaceLine

Ersetzt eine Textzeile in einer Datei durch eine andere. Die ersetzte Zeile darf maximal 1024 Zeichen lang sein (R-BASIC Begrenzung für Strings).

Format **FileReplaceLine** fh, zeile\$ [, mode])
fh, zeile\$ [, mode]): siehe **FileWriteLine**

Die Systemvariable **fileError** wird gesetzt oder gelöscht.

9.6 Sonstige Funktionen

Positionieren des Dateizeigers

Während die Datei geöffnet ist, gibt es einen "**Dateizeiger**", der die Position bestimmt, an der Daten gelesen oder geschrieben werden. Der Zugriff auf den Dateizeiger einer Datei erfolgt über eine Variablen vom Typ FILE, die von **FileOpen** bzw. **FileCreate** geliefert wird.

FileGetPos

Liest die aktuelle Position des Dateizeigers aus. Der zurückgegebene Wert liegt im Bereich von 0 (Dateianfang) bis 4294967295, da Dateien maximal 4 GByte groß werden können.

Format: x = **FileGetPos** (fh)

fh: Variable (oder Funktion) vom Typ FILE. Bezeichnet die Datei.

Die Systemvariable **fileError** wird gesetzt oder gelöscht.

Beispiel:

```
' Herausfinden, ob man schon am Dateiende ist  
IF FileSize(fh) = FileGetPos(fh) THEN Print "Dateiende erreicht"
```

FileSetPos

Setzt den Dateizeiger an eine bestimmte Position. Die folgenden File~ Operationen lesen bzw. schreiben ab dieser neuen Position.

Format: **FileSetPos** fh, position [, fromEnd]

fh: Datei-Variable, bestimmt die betroffene Datei.

position: neue Dateiposition

fromEnd: bestimmt den Positionierungs-Modus

- nicht angegeben oder Null: Ab Dateianfang
- ungleich Null: Ab Dateiende

Achtung! "position" muss hier **negativ** sein, damit Sie eine Position vor dem Dateiende anwählen. Ist "position" positiv, ist das Ergebnis unbestimmt. Manchmal wird die Datei verlängert, manchmal nicht.

Die Systemvariable **fileError** wird gesetzt oder gelöscht.

Beispiele:

Setzen des Dateizeigers an den Dateianfang

```
FileSetPos fh, 0
```

Setzen des Dateizeigers ans Dateieende, so dass FileWrite Daten an die Datei anhängen kann:

```
FileSetPos fh, 0, TRUE
```

Setzen des Dateizeigers 100 Byte vor das Dateieende. Der Parameter "position" ist hierzu negativ.

```
FileSetPos fh, -100, TRUE
```

Verschieben des Dateizeiger um 10 Byte nach hinten.

```
FileSetPos fh, FileGetPos(fh) + 10
```

Setzen des Dateizeigers hinter das 2. Byte (ab Dateianfang). Das erste Byte hat die Position 0, das zweite die Position 1

```
FileSetPos fh, 2
```

Tipps & Tricks:

- Benutzen Sie `FileSize`, um die aktuelle Größe der Datei zu ermitteln.
- Setzen Sie den Dateizeiger hinter das Dateieende (auf einen Wert, der größer ist, als der von `FileSize` gelieferte), so ist das Ergebnis unbestimmt.
Anmerkung: In einigen Fällen wird die Datei verlängert (und die neu angehängten Bytes mit Nullen initialisiert), in anderen Fällen wurde die Dateilänge nicht geändert.

Weitere Datei-Operationen

FileResize

Einfügen oder Löschen von Bytes an der aktuellen Position einer Datei. Die dahinter folgenden Daten werden automatisch verschoben. Eingefügte Bytes werden mit Null initialisiert. Der Dateizeiger wird hinter die eingefügten Bytes gesetzt. **FileResize** eignet sich auch, um Null-Bytes an eine Datei anzufügen.

Format: **FileResize** fh, bytesToDelete, bytesToInsert

fh:	Variable (oder Funktion) vom Typ FILE. Bezeichnet die Datei.
bytesToDelete:	Anzahl zu löschender Bytes. Maximal 2 GByte.
bytesToInsert:	Anzahl einzufügender Bytes. Maximal 2 GByte.

Die Systemvariable **fileError** wird gesetzt oder gelöscht.

Beispiele:

FileResize fh, 200, 0	' 200 Bytes löschen. Der Dateizeiger bleibt an der aktuellen Position.
FileResize fh, 0, 200	' 200 Bytes einfügen, der Dateizeiger steht hinter den eingefügten Bytes.
FileResize fh, 100, 200	' 100 Byte löschen und durch 200 Null-Bytes ersetzen. Die Datei wird um 100 Byte verlängert, der Dateizeiger wird hinter die 200 Null-Bytes gesetzt.
FileResize fh, 100, 40	' 100 Byte löschen und durch 40 Null-Bytes ersetzen. Die Datei wird um 60 Byte kürzer, der Dateizeiger wird hinter die 40 Null-Bytes gesetzt.
FilePos fh, 0, TRUE	' Dateizeiger ans Dateieende
FileResize fh, 0, 800	' 800 Null-Bytes anhängen

FileTruncate

FileTruncate schneidet die Datei an der Position des aktuellen Dateizeigers ab.

Format: **FileTruncate** fh
fh: Variable (oder Funktion) vom Typ FILE. Bezeichnet die Datei.

Die Systemvariable **fileError** wird gesetzt oder gelöscht.

Beispiel:

! Einkürzen einer Datei auf die Länge Null, d.h. alle Daten löschen.

FileSetPos fh, 0
FileTruncate fh

FileCommit

Bewirkt, dass alle in Daten der Datei unverzüglich auf die Platte geschrieben werden. Das GEOS-System hält aus Performance-Gründen viele Daten oft bis zum Schließen der Datei im Speicher, d.h. die Daten kommen manchmal erst beim **FileClose** wirklich auf der Platte an. **FileCommit** ist sinnvoll, wenn mehrere Programme gleichzeitig auf die gleiche Datei zugreifen und man kann damit einem Datenverlust im Falle eines Systemabsturzes vorbeugen.

Format: **FileCommit** fh
fh: Variable (oder Funktion) vom Typ FILE. Bezeichnet die Datei.

Die Systemvariable **fileError** wird gesetzt oder gelöscht.

10 Arbeit mit Laufwerken und Datenträgern

Mit **DiskWriteable**, **DiskSpace**, **DiskExist** und **DriveInfo** erhalten Sie Informationen über Datenträger oder Laufwerke. Diese Befehle setzen die Systemvariable **fileError** (z.B. wenn das Laufwerk nicht existiert) oder löschen Sie, wenn kein Fehler auftrat.

DiskWriteable

Prüft, ob sich ein beschreibbarer Datenträger im Laufwerk befindet. DiskWriteable liefert "wahr" (TRUE, -1) wenn sich ein beschreibbarer Datenträger im Laufwerk befindet. Existiert das Laufwerk nicht, befindet sich kein, oder kein formatierter Datenträger im Laufwerk, liefert DiskWriteable "falsch" (FALSE, 0).

Format: x = **DiskWriteable** (lw\$)

lw\$: Laufwerksbezeichnung, z.B. "A:" oder "D:"

Beispiel:

```
IF DiskWriteable("a:") = 0 THEN Print "Die Diskette ist schreibgeschützt."
```

DiskSpace

Prüft den auf einem Datenträger verfügbaren Platz.

Format: x = **DiskSpace** (lw\$ [, all])

lw\$: Laufwerksbezeichnung, z.B. "A:" oder "D:"

all: (optional) Wenn angegeben und ungleich Null, liefert DiskSpace den insgesamt auf dem Datenträger vorhandenen Platz.
Wenn nicht angegeben oder gleich Null, liefert DiskSpace den freien Speicherplatz auf dem Datenträger.

Wenn mehr als 2 GB verfügbar bzw. vorhanden sind, liefert DiskSpace immer den Maximalwert von 2 147 418 112 Byte zurück.

Beispiel:

```
Print "Speicherstatus von Laufwerk C:"  
Print DiskSpace ("C:", TRUE) " Bytes insgesamt"  
Print DiskSpace ("C:") " Bytes verfügbar"
```

DiskExist

Prüft, ob sich ein formatierter Datenträger im Laufwerk befindet. Wenn Sie wissen wollen, ob ein bestimmtes Laufwerk existiert, verwenden Sie **DriveInfo** (unten). DiskExist liefert "wahr" (TRUE, -1), wenn ein formatierter Datenträger im Laufwerk ist. Das gilt auch für Festplatten. Befindet sich kein oder nur ein unformatierter Datenträger im Laufwerk, liefert DiskExist "falsch" (FALSE, 0).

Format: x = **DiskExist** (lw\$)

 lw\$: Laufwerksbezeichnung, z.B. "A:" oder "D:"

Hinweis: DiskExist setzt die Systemvariable **fileError** auf 0, wenn das Laufwerk existiert aber kein Datenträger enthalten ist.

Beispiel:

```
IF DiskExist("a:") = 0 THEN Print "Legen Sie eine formatierte Diskette ein!"
```

DriveInfo

Liefert ausführliche Informationen über das angegebene Laufwerk. DriveInfo gibt Null zurück, wenn das Laufwerk nicht existiert, andernfalls einen Wert ungleich Null.

Format: x = **DriveInfo** (lw\$)

 lw\$: Laufwerksbezeichnung, z.B. "A:" oder "D:"

Beispiel:

```
IF DriveInfo("H:") = 0 THEN Print "Laufwerk H: existiert nicht"
```

Existiert das Laufwerk, enthält der Rückgabewert vielfältige Informationen über das Laufwerk, wobei jedes einzelne Bit eine Bedeutung hat. Die Auswertung dieser Daten ist etwas für Experten. Kenntnisse im Umgang mit Bits und logischen Verknüpfungen sind hilfreich. R-BASIC unterstützt die Arbeit mit den wichtigsten Eigenschaften durch ein paar vordefinierte Konstanten.

Wert	R-BASIC Konstante	Bedeutung
15	DI_TYPE_MASK	Maske zum Herausfiltern des Laufwerkstyp
2	DI_FIXED	Laufwerkstyp: Festplatte
4	DI_CD_ROM	Laufwerkstyp: CD-ROM-Laufwerk
64	DI_REMOVABLE	Datenträger ist wechselbar (CD, Diskette)
2048	DI_READ_ONLY	"Nur-Lesen" - Datenträger

Tabelle 10.1 R-BASIC Konstanten für DriveInfo. Eine komplette Beschreibung finden Sie auf der nächsten Seite.

Verwenden Sie die Konstanten wie folgt:

Beispiel 1:

```
DIM bitfeld as word
```

```
bitfeld = DriveInfo ("D:")
```

```
IF bitfeld = 0 THEN Print "Laufwerk D: existiert nicht"
```

```
! ## Abfragen für Experten
```

```
IF bitfeld AND DI_REMOVABLE THEN
    Print "Datenträger in D: ist entnehmbar"
END IF
```

```
IF (bitfeld AND DI_TYPE_MASK) = DI_CD_ROM THEN
    Print "Laufwerk D: ist ein CD-Laufwerk"
END IF
```

Beispiel 2:

```
DIM info, type as word
```

```
info = DriveInfo ("D:")
```

```
! ## Herausfinden des Laufwerkstypes
```

```
type = info AND DI_TYPE_MASK ' Bits 4 .. 15 Null setzen
```

```
IF type AND DI_FIXED THEN Print "Festplatte"
```

```
IF type AND DI_CD_ROM THEN Print "CD-ROM Laufwerk"
```

```
! ## Entnehmbar und beschreibbar
```

```
IF info AND DI_REMOVABLE THEN Print "Entnehmbarer Datenträger"
```

```
IF info AND DI_READ_ONLY THEN Print "Nur-Lesen Datenträger"
```

Bedeutung der Bits im Rückgabewert von DriveInfo

DriveInfo liefert eine 16-Bit Wert, der ausführliche Informationen über das abgefragte Laufwerk enthält. Hier finden Sie eine komplette Liste der Bits. Kenntnisse im Umgang mit Bits und logischen Verknüpfungen sind hilfreich.

Hinweis: Die meisten Infos sind der SDK-Dokumentation ungeprüft entnommen. Für eventuelle Fehler in der SDK-Dokumentation kann R-BASIC nichts.

Bit-Nummer	R-BASIC Konstante	Bedeutung
0 - 3	DI_TYPE_MASK = 15	Laufwerkstyp 0 : 5,25" Diskette 1 : 3,5" Diskette
	DI_FIXED = 2	2 : Fixed (Festplatte)
	DI_CD_ROM = 4	3 : RAM-Laufwerk 4 : CD-ROM-Laufwerk 5 : 8" Diskette 16: unbekannter Typ
4		(unbenutzt / reserviert)
5		Netzwerk-Laufwerk
6	DI_REMOVABLE = 64	Datenträger ist wechselbar (CD, Diskette)
7		Laufwerk ist physisch vorhanden
8		"besetzt" - Laufwerk wird benutzt
9		"alias" - Laufwerk ist ein Alias für einen Pfad auf einem anderen Laufwerk
10		Datenträger ist formatierbar
11	DI_READ_ONLY = 2048	"Nur-Lesen" - Datenträger, z.B. CD
12		Laufwerk ist nicht über das Netzwerk sichtbar.
13		(unbenutzt / reserviert)
14		(unbenutzt / reserviert)
15		(unbenutzt / reserviert)

Tabelle 10.2: Bedeutung der Bits im Rückgabewert von **DriveInfo**.

11. Portzugriffe

Achtung! Die Befehle greifen direkt auf die Hardware des Computers zu!

INP

Die Funktion INP (= Input) greift auf die Hardware des Computers zu und liest ein Byte von einem I/O-Port. Sie müssen sich mit der Hardware des Computers und den Portadressen sowie deren Bedeutung auskennen, um diesen Befehl nutzen zu können.

Format: y = INP (port)

Parameter: port: Port-Adresse, von der gelesen werden soll

OUT

Der Befehl OUT (= Output) greift auf die Hardware des Computers zu und schreibt ein Byte in einem I/O-Port. Sie müssen sich mit der Hardware des Computers und den Portadressen sowie deren Bedeutung auskennen, um diesen Befehl nutzen zu können.

Format: OUT port, wert

Parameter: port: Port-Adresse, auf die ausgegeben werden soll

 wert: auszugebender Wert

Achtung: Eine Ausgabe ungültiger Werte auf bestimmte I/O-Ports des Computers könnte die Funktion des Computers schwer stören oder unmöglich machen. Der Programmierer von R-BASIC übernimmt keinerlei Haftung für Schäden, die auf eine fehlerhafte Verwendung der Befehle INP und OUT zurückgehen!

WAIT

Die Funktion WAIT (= Warte) greift auf die Hardware des Computers zu und wartet bis ein bestimmtes Bitmuster an einem I/O-Port anliegt. Sie müssen sich mit der Hardware des Computers und den Portadressen sowie deren Bedeutung auskennen, um diesen Befehl nutzen zu können.

Format: WAIT port, mask [, xBits [, mode]]

Parameter: port: Port-Adresse, von der gelesen werden soll

 mask: Maske, welche Bits abgefragt werden sollen

 xBits: Welche Bits davon Null sein müssen.

 xBits ist optional. Vorgabewert ist Null.

 mode: mode = 0: WAIT wartet **BIS** das gesuchte Bitmuster
 erscheint (Vorgabewert, wenn mode nicht angegeben).

 mode = 1: WAIT wartet **SOLANGE** das gesuchte Bitmuster
 am Port anliegt.

 Soll mode angegeben werden, so ist auch xBits anzugeben.

Funktion: WAIT wartet, bis die bitweise logische Verknüpfung des Bitmusters am abgefragten Port mit mask und (falls angegeben) xBits die geforderte Bedingung (mode) erfüllt.
"mask" bestimmt, welche Bits berücksichtigt werden sollen. Bits, die in mask nicht gesetzt sind, können beliebige Werte haben.
"xBits" bestimmt, welche Bits Null sein müssen. Bits, die in xBits nicht gesetzt sind, müssen Eins sein, damit die Bedingung erfüllt ist.

Intern werden "mask" und "xBits" (wenn angegeben) logisch XOR verknüpft. Der von "port" gelesene Wert mit "mask" logisch UND verknüpft. Die genauen Formeln lauten:

$$\begin{aligned} \text{erwartung} &= (\text{mask XOR xBits}) \text{ AND mask} \\ \text{gelesen} &= \text{INP}(\text{port}) \text{ AND mask} \end{aligned}$$

mode = 0 wartet, bis das gelesene Bitmuster erscheint:
Warte bis erwartung = gelesen

mode = 1 wartet, solange das gelesene Bitmuster korrekt ist:
Warte solange erwartung = gelesen

Beispiele: x ist ein beliebiges Bit (0 oder 1)

1 ist binär 0001, 2 ist binär 0010, 3 ist binär 0011

Warten auf ein bestimmtes Bitmuster an Port p:

wait p, 2 ' warten auf xx1x an Port p

wait p, 2, 2 ' warten auf xx0x

wait p, 3, 1 ' warten auf xx10

Warten solange ein bestimmtes Bitmuster an Port p anliegt:

wait p, 2, 0, 1 ' warten solange xx1x an Port p

wait p, 2, 2, 1 ' warten solange xx0x anliegt

wait p, 3, 3, 1 ' warten solange xx00 anliegt

' WAIT setzt fort, wenn xx01, xx10 oder xx11 erscheint.

12. Speicherzugriff

Gelegentlich benötigt man einfach eine bestimmte Menge Speicher, die man nach den eigenen Vorstellungen organisieren kann. R-BASIC ermöglicht zwar keinen direkten Zugriff auf den Hauptspeicher des Computers (das wäre zu unsicher), stellt dem Programmierer aber 64 kByte "geschützten" Speicher zur Verfügung, der aus Sicht des Programms als 64 kByte fortlaufender Speicher über die Adressen 0 bis 65635 (hexadezimal &h0 bis &hFFFF) angesprochen werden kann. Der Zugriff auf diesen Speicher erfolgt über schreibend über diverse **Poke**- und lesend über die passenden **Peek**-Befehle (siehe Tabelle).

Befehl	Wirkung
POKE <adr>, wert	Schreibt ein Byte
DOKE <adr>, wert	Schreibt einen Integer- oder Word-Wert (2 Byte)
POKE\$ <adr>, <string>	Schreibt eine String. Am Ende des Strings wird eine binäre Null als Ende-Kennung geschrieben. Es werden also LEN(<string>)+1 Bytes geschrieben.
SPOKE<adr>, <struktur>	Schreibt eine Struktur. Die Anzahl der geschriebenen Bytes hängt von der Struktur ab und beträgt sizeof(<struktur>) Bytes.
<nVar> = PEEK (<adr>)	Liest ein Byte
<nVar> = DEEK (<adr>)	Liest einen Word oder Integer-Wert (2 Byte).
<sVar\$> = PEEK\$ (<adr>)	Liest einen String. Das Stringende ist durch eine binäre Null gekennzeichnet. Es werden maximal 1024 Zeichen gelesen.
<stVar> = SPEEK (<adr>)	Liest eine Struktur. Die Anzahl der gelesenen Bytes hängt von der Struktur ab und beträgt sizeof(<stVar>) Bytes.
VPOKE <adr>, wert	Kompatibilitätsbefehl für KC-85 Kompatibilität. Hat in R-BASIC die gleiche Wirkung wie POKE.
<nVar> = VPEEK (<adr>)	Kompatibilitätsbefehl für KC-85 Kompatibilität. Hat in R-BASIC die gleiche Wirkung wie PEEK.

<adr> numerischer Ausdruck, der einen Wert von 0 bis 65635 liefert. Liegt der Wert außerhalb dieses Bereichs, wird MOD 65636 gerechnet, dh. es wird einfach wieder von vorne begonnen.

<string> Ein Stringausdruck.

<struktur> Eine Strukturvariable oder eine Funktion, die eine Struktur liefert. Es sind sowohl R-BASIC Strukturen als auch selbst definierte Strukturen zulässig.

<nVar> numerische Variable.

<sVar\$> String-Variable.

<stVar> Struktur-Variable.

Hinweise:

- R-BASIC verwaltet den Speicher in Blöcken zu 8 kByte, die erst angefordert werden, wenn sie benötigt werden.
- Wollen Sie andere Datentypen (FILE, HANDLE, DWORD etc.) schreiben, müssen Sie sie in einer Struktur kapseln.
- R-BASIC kontrolliert nicht, ob die gelesenen Daten gültig sind, d.h. zum gelesenen Datentyp passen.