Dateien (allgemein)

- Die Sicherung d. Programms obliegt d. IDE;
  die Sicherung d. Ergebnisse obliegt d. Programm(ierer|In).
- Ergebnisse können in Dateien gespeichert werden – d.h. in Speicherbereichen mit individuellem Anfang, Ende sowie Namen und (da hierarchisch organisiert:) Pfad.
- „Öffnen“ einer Datei: (Unumgängliche) Mitteilung d. Programms an das System, daß es Daten mit einer Datei austauschen will.
- Durch das Öffnen wird (u.a.)
  - die Datei ermittelt u. ggf. (falls zum Schreiben geöffnet und nicht existent) generiert (Pfad muß existieren)
  - der Datei eine programm-interne Nummer zugewiesen,
  - Zugriffsart und -modus für den Datenaustausch festgelegt,
  - ein E/A-Puffer reserviert
- „Schließen“ einer Datei hebt die Kommunikation mit ihr auf und gibt die reservierten Ressourcen wieder frei.

Dateien (allgemein)

VB-Anweisung:

```
Open PFADNAME For MODUS [Access ZUGRIFF] [SPERRE]
          As [#]DATEINR [Len=SATZLÄNGE]
```

Bedeutung der Parameter:

`PFADNAME`	Dateiname (mit / ohne Pfad)
`MODUS`	eines von: **`Append`**, **`Binary`**, **`Input`**, **`Output`** oder **`Random`**.
`ZUGRIFF`	Festlegung der Operationen an der geöffneten Datei: **`Read`**, **`Write`** oder **`Read Write`**
`SPERRE`	Festlegung der zulässigen Operationen für andere Prozesse: **`Shared`**, **`Lock Read`**, **`Lock Write`** oder **`Lock Read Write`**
`DATEINR`	Dateinummer 1 bis 511
`SATZLÄNGE`	≤ 32.767 (Bytes): bei sequentiellem Zugriff: Anzahl gepufferter Zeichen, bei wahlfreiem Zugriff (**`Random`**) Datensatzlänge

Dateien (allgemein)

Schließen von Dateien mit Dateinummern `DATEINR1`, ...:

```
Close [[#]DATEINR1][, [#]DATEINR2]
```

Bemerkungen:

- Sind keine Dateinummern angegeben, so werden alle geöffneten Dateien geschlossen.
- Zur Ausführung von **`Close`** gehört auch, daß evtl. noch vorhandener Pufferinhalt in die Datei geschrieben wird (falls im Modus **`Output`** oder **`Append`** geöffnet), bevor der gesamte ihr zugeordnete Pufferspeicher freigegeben wird.
- ... für sequentielle, wahlfreie und binäre Dateien:

Dateien (allgemein)

Anmerkungen:

- „System“ ist im hiesigen Kontext (vereinfacht) jener Teil des BSs, der für die Ausführung von Programmen zuständig ist.

- Ein „Puffer“ ist hier ein Speicher, der zwischen Programm und Dateisystem geschaltet wird; er hilft (u.a.), Geschwindigkeitsunterschiede auszugleichen (z.B. RAM vs. Diskette) und allgemein die System-Verwaltung zu erleichtern (Schutz vor Datenverlust bei Programm-Ende etc.)

- Die Bedeutung der Einstellung: „**`Lock ...`** “ läßt sich leicht behalten als: „Andere abhalten von...“

- Die redundante Verwendung der `Open`-Parameter ist zulässig (obwohl kaum sinnvoll), sofern sie keine Widersprüche enthält; so führt:

  ```
  Open "ABC.txt" For Output Access Read Write As #1
  ```

  bereits beim Editieren zu Fehlermeldungen (wg. `Read`).

Dateien (sequentiell)

Prof. Dr. Aris Christidis

Schreiben in (bzw.: anhängen am Ende von) Datei für sequentiellen Zugriff (sequentielle Datei – engl.: *sequential access / seq. file; text file*):

```
Open PFADNAME For {Output|Append} [SPERRE] As [#]DATEINR

Print #DATEINR, [AUSGABE1] [[,|;]AUSGABE2][,...][,|;]
    'überträgt Daten unverändert u. länderspezifisch (Dezimalkomma etc.)

Write [#]DATEINR, [AUSDRUCK1][[,|;]AUSDRUCK2][...][,|;]
    'setzt: doppelte Anführungsstriche (") vor und hinter Zeichenketten,
    'Zeilenwechsel (<CR> Chr(13) & <LF> Chr(10)) nach letztem Ausdruck,
    'Kommata zwischen Ausdrücke; länderunspezifisch
```

Parameter:

- `AUSGABEx:[{Spc(n)|Tab[(n)]}][AUSDRUCKx][...] [{[;|,]}]`
- `n`: Anzahl von Leerzeichen bzw. Tabulatoren (=14 Leerz.) in Zieldatei
- `AUSDRUCKx`: Numerischer Ausdruck (Variablen u./o. Konstanten) o. Zeichenkette in Gänsefüßchen (") - falls abwesend: Leerzeile in Zieldatei
- Semikola (oder Leerzeichen) als Trennzeichen ⇨ lückenlose Ausgabe (Kommata bei **`Print`** ⇨ Tabulatoren); Semikola (o. Kommata) als Abschlußzeichen unterdrücken Zeilenwechsel in Zieldatei

Dateien (sequentiell)

Prof. Dr. Aris Christidis

Lesen aus sequentiellen Dateien:

```
Open PFADNAME For Input [SPERRE] As [#]DATEINR
```

```
Input(n, [#]DATEINR) 'Input-Funktion
```

'liest n Zeichen aus geöffneter Datei (meist geschrieben mit **Print** o. **Put**)

'gibt zurück jedes gelesene Zeichen (Leerzeichen, Gänsefüßchen, **<CR>**…)

```
Input [#]DATEINR, VAR1 [,VAR2,...] 'Input-Anweisung
```

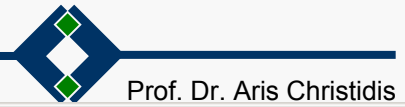
'liest Werte aus geöffneter Datei (meist geschrieben mit **Write**) u. weist sie Variablen zu (nicht: Feldern, Objekten, Strukturen, nur deren Variablen)

'nicht übertragen: Kommata, Leerzeichen, Gänsefüßchen, Zeilenwechsel

Typische Anwendungen:

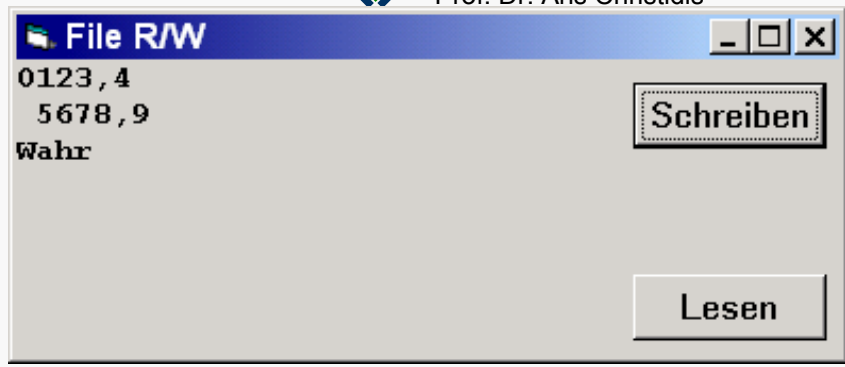
- Schreiben mit **Print**, lesen mit **Input**-Fkt., Nutzung durch Menschen
- Schreiben mit **Write**, lesen mit **Input**-Anw., Nutzung durch Programm

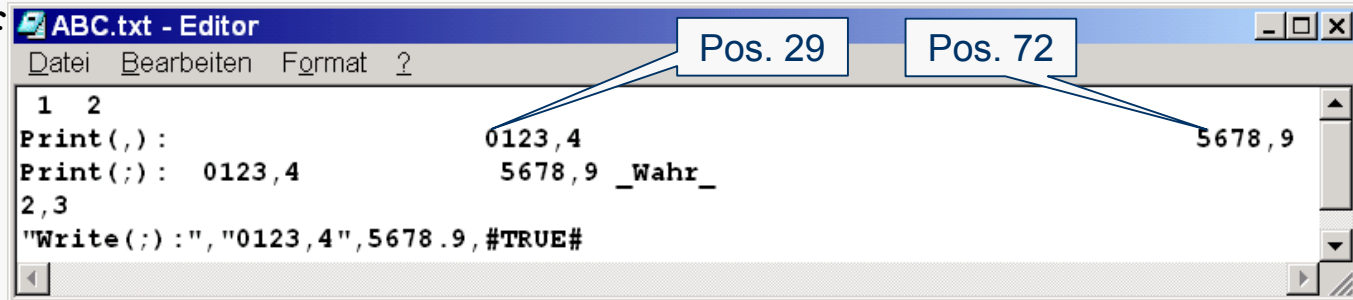
Dateien (sequentiell)



Prof. Dr. Aris Christidis

```
Option Explicit
Dim charVar$, singVar!
Dim boolVar As Boolean

Private Sub cmdOut_Click()
charVar = "0123,4": singVar = 5678.9
boolVar = -1: Cls
Print charVar: Print singVar: Print boolVar

Open "ABC.txt" For Output As #1
Print #1, 1; 2
Print #1, "Print(,):", Spc(2), charVar, Tab, singVar
Print #1, "Print(;):"; Spc(2); charVar; Tab; singVar;
Print #1, "_"; boolVar; "_"

Write #1, 2; 3 'Trennung mit Komma hat identische Wirkung:
Write #1, "Write(;):"; charVar; singVar;
Write #1, boolVar

Close #1
End Sub
```

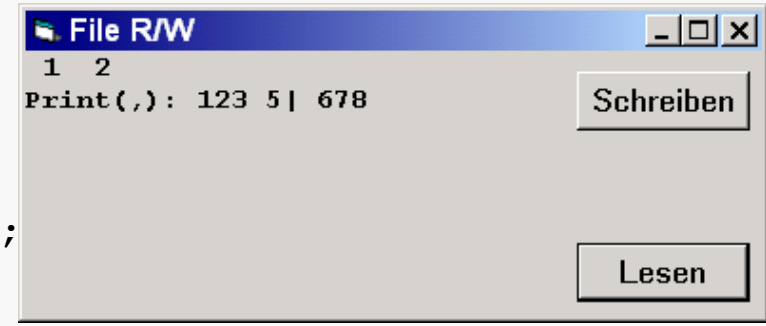


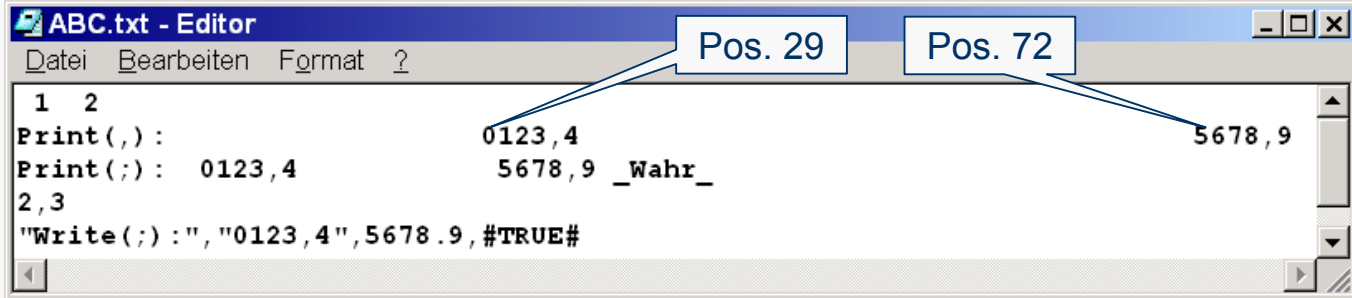


Dateien (sequentiell)

Prof. Dr. Aris Christidis

```
Private Sub cmdIn_Click()
Dim dumInt%, dumSing!, dumStr$: Cls
Open "ABC.txt" For Input As #1
'Lesen 1 u. 2 (inkl.Print-LZ), CR, LF:
dumInt = Val(Input(2, #1)): Print dumInt;
dumInt = Val(Input(6, #1)): Print dumInt
'Lesen bis vor die 0, ausgeben "Print(,):":
dumStr = Input(28, #1): Print Left$(dumStr, 9);
'Lesen als Zahl: wg. Komma=123; Ausgabe(mit LZ):
dumSing = Val(Input(6, #1)): Print dumSing;
Do: dumStr = Input(1, #1) 'Fußsteurg wg.Anfangswert
Loop While dumStr = " "   'Nach Lesen feststellbar!
Print dumStr; "|";    'verpasste "5" u. Trennstrich
'statt "5" LZ dahinter;wg.Komma =678;Ausg.:LZ davor:
singVar = Val(Input(6, #1)): Print singVar;
```

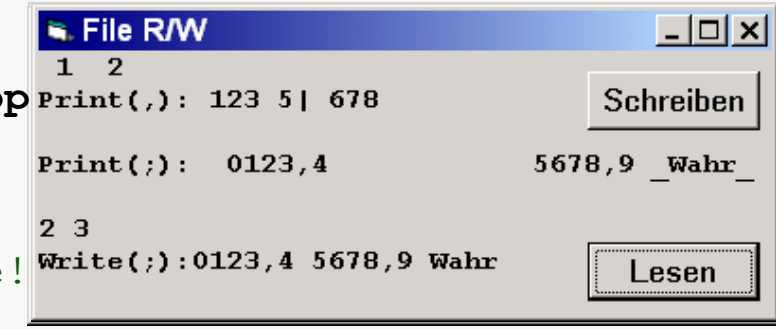


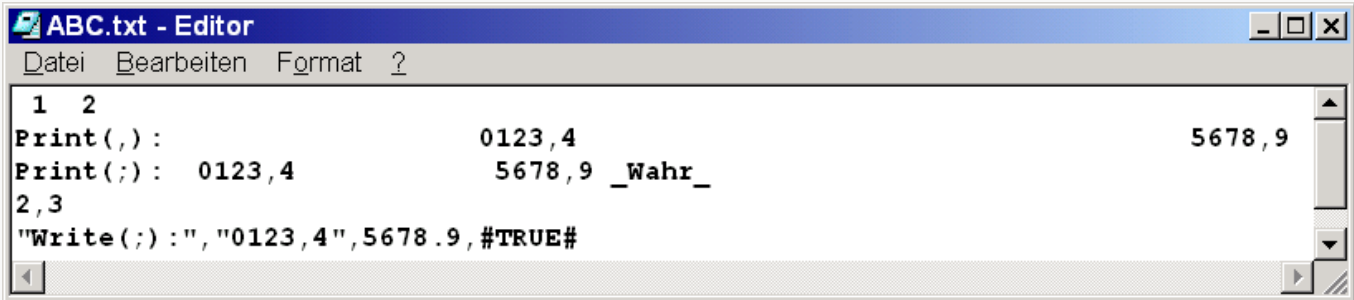


Dateien (sequentiell)

Prof. Dr. Aris Christidis

```
'Print Chr(13) = Print Chr(10)) = Zeilenwechsel:
Do Until dumStr = "_" 'Erst danach klar
dumStr = Input(1, #1): Print dumStr;: Loop
boolVar = Input(4, #1): Print boolVar;
dumStr = Input(3, #1)
Print dumStr; '"_",CR/LF zusammen:1 Zeile!

Print 'Leerzeile: --- Write / Input #:---
Input #1, dumStr:  Print dumStr;  '2 als Text
Input #1, dumInt:  Print dumInt   '3 als Zahl
Input #1, dumStr:  Print dumStr;  '"Write(,):"
Input #1, charVar: Print charVar; '"0123,4"
Input #1, singVar: Print singVar; '5678.9
Input #1, boolVar: Print boolVar  '#TRUE#
Do While Not EOF(1) 'End Of File als Ereignis
Input #1, dumStr
Print dumStr; 'Na?
Loop: Close #1
End Sub
```





Dateien (sequentiell)

Bemerkungen zu sequentiellen Dateien:

- Öffnen einer Datei im **`Output`**-Modus stellt immer eine leere Datei bereit; existierende Dateien werden dadurch gelöscht.
- Für einen Wechsel zwischen Schreiben und Lesen (und umgekehrt) muß zuvor die Datei geschlossen werden.
- Für selektive Änderungen in einer Datei (Teillöschung, Zusätze, Umordnung) wird eine neue Datei bel. Namens mit dem gewünschten Inhalt erstellt, dann das Original gelöscht und schließlich die neue Datei auf den alten Namen umbenannt.
- Vorteile sequentieller Dateien:
  - verhältnismäßig effiziente Speicherplatz-Nutzung
  - relativ leichte Programmierung
- Nachteile sequentieller Dateien:
  - Schwierigkeit, konkrete Inhalte der Datei abzufragen
  - Unmöglichkeit, Teile der Datei direkt zu verändern

Dateien (sequentiell)

Anmerkungen:

- Der **`Open`**-Anweisung kann auch eine Zugriffsart (**`ACCESS`**) zugefügt werden; diese darf allerdings nur **`Read`** (bei **`Input`**) bzw. **`Write`** (bei **`Output`**/**`Append`**) sein.
- Mit „länder(un)spezifisch“ ist die Frage gemeint, ob die Speicherung der Daten abhängig von den „Ländereinstellungen“ ist; darunter fallen z.B. die Formate für Datum, Währung, Wahrheitswerte (True/False bzw. Wahr/Falsch), Dezimal-Trennzeichen (Punkt bzw. Komma).
- **`Print`** (auf Dateien wie auf Fenster) gibt vor und hinter Zahlen jeweils ein Leerzeichen aus (Gegenmittel: **`Format()`**) und setzt bei Trennung mit Komma die nächste Ausgabe an die nächste Tabulator-Position – d.h., **`Tab`** führt zur Ausgabe von mind. 14 Leerzeichen (=1 Tab)
- **`<CR>`** steht für „Carriage Return“ (Wagenrücklauf), **`<LF>`** für „Line Feed“ (Zeilenvorschub)
- Die Funktionen **Spc()** und **Tab** werden i.a. auch von der **Write**-Anweisung angenommen; aber ihre Behandlung ist „gewöhnungsbedürftig“:
  - **Spc(**n**),** fügt n Leerzeichen in der Datei ein, die durch ein Komma abgeschlossen werden (d.h., ein neuer Eintrag entsteht); **Spc(**n**);** verschiebt dagegen lediglich die weiteren Einträge um n Leerzeichen.
  - **Tab,** wird als ein Komma umgesetzt: in die Datei werden 2 Kommata (direkt hintereinander) geschrieben; **Tab;** ergibt gewissermaßen eine Art „Null-Eintrag“: ein (weiteres) Komma wird vor den darauffolgenden Eintrag gesetzt.
- Die **`Input`**-Funktion liest genaugenommen n Unicode-Zeichen (2n Bytes); bei Bedarf kann die Funktion **`InputB`** verwendet werden (liest Bytes).
- Die **`Input`**-Anweisung liest praktisch von einem Separator (Komma) zum nächsten und „versucht“, das Gelesene im bereitgestellten (Variablen-)Speicher „unterzubringen“.

Dateien (wahlfrei / binär)

Prof. Dr. Aris Christidis

Dateien für wahlfreien Zugriff (wahlfreie Dateien – engl.: *random-access files* – auch: *direct-access files*, *relative files*):

```
Open PFADNAME For Random [Access ZUGRIFF] [SPERRE] As
     [#]DATEINR Len=SATZLÄNGE
```

```
Put [#]DATEINR, [SATZNR], VAR
```

- 'schreibt Daten byteweise (wie in `VAR` gespeichert) ab Datensatz-Anfang
- 'übersprungene Datensatz-Nummern werden ggf. generiert
- 'Fehlermeldung, falls `VAR`-Datenmenge größer als Satzlänge
- 'Strukturen/ Felder/ Strings/ Variablen fester Länge: nur Daten-Übertragg.
- 'vor Strings variabler Länge: 2-Byte-Deskriptor (Länge);
- 'vor Variant: 2-Byte-Deskriptor (Typ) – falls als String: zzgl. 2 Byte (Länge);
- 'vor dyn. Feldern: 2-Byte-Deskriptor + (Anzahl Dimensionen)*8Byte

```
Get [#]DATEINR, [SATZNR], VAR
```

- 'liest Daten byteweise (erwartet ggf. Deskriptoren)

Parameter:

`ZUGRIFF`	**`Read`**, **`Write`** oder **`Read Write`** (default)
`SATZNR`	Datensatz-Nummer (*record number*); bei Auslassung: nächster
`VAR`	Variable (auch benutzerdefiniert – kein Objekt!)

Dateien (wahlfrei / binär)

Prof. Dr. Aris Christidis

Dateien für binären Zugriff (binäre Dateien – engl.: *binary files*):

```
Open PFADNAME For Binary [Access ZUGRIFF] [SPERRE] As
     [#]DATEINR
```

```
Put [#]DATEINR, [BYTENR], VAR
```

- schreibt Daten byteweise (wie in `VAR` gespeichert) ab `BYTENR`
- Daten-Übertragung ohne Deskriptoren (**`Open`**-gesteuert)

```
Get [#]DATEINR, [BYTENR], VAR
```

- liest Daten byteweise (erwartet keine Deskriptoren)

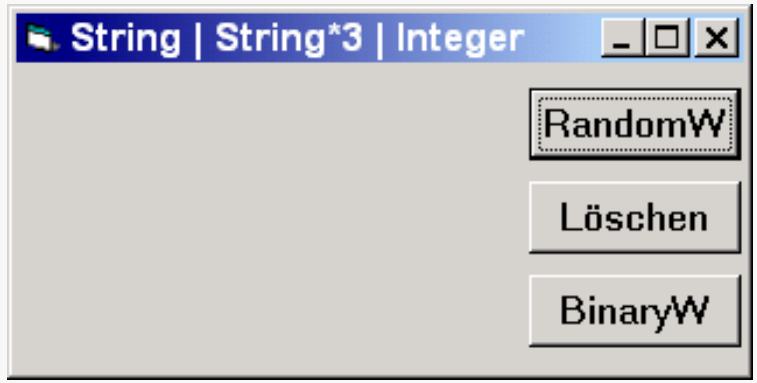
Parameter:

`ZUGRIFF`	**`Read`**, **`Write`** oder **`Read Write`** (default)
`BYTENR`	Byte-Nummer; bei Auslassung: nächstes
`VAR`	Variable (auch benutzerdefiniert – kein Objekt!)

Dateien (wahlfrei / binär)

Prof. Dr. Aris Christidis

Beispiel:

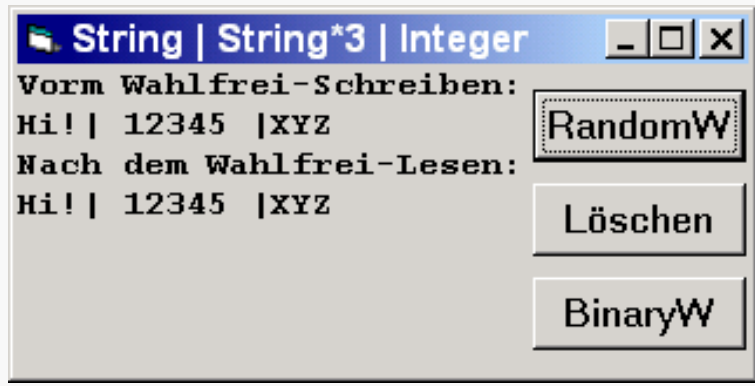



```
Option Explicit
Const R = 0, W = 1
Private Type Datensatz
  inteVar As Integer     '2Byte
  cFixVar As String * 3 '3Byte
End Type

Dim Satz As Datensatz
Dim charVar$ '2 Byte(Länge)+Inhalt
Dim recNr%, recLen%, rndStat%, binStat%
Dim rndCap$(1), binCap$(1)

Private Sub Form_Load()
rndStat = W: binStat = W
rndCap$(R) = "RandomR": rndCap$(W) = "RandomW"
binCap$(R) = "BinaryR": binCap$(W) = "BinaryW"
End Sub

Private Sub cmdDel_Click()
Open "ABC.txt" For Output As #1: Close #1
End Sub
```

Dateien (wahlfrei / binär)

Prof. Dr. Aris Christidis

```
Private Sub cmdRnd_Click()
recLen = Len(Satz): recNr = 3
Open "ABC.txt" For Random As #1 Len = recLen

Select Case rndStat
Case W: Cls: charVar = "Hi!"
    Satz.inteVar = 12345: Satz.cFixVar = "XYZ"
    Print "Vorm Wahlfrei-Schreiben:„
    Print charVar; "|"; Satz.inteVar; "|"; Satz.cFixVar
    Put #1, , charVar
    Put #1, recNr, Satz

Case R: charVar = ""
    Satz.inteVar = 0: Satz.cFixVar = ""
    Get #1, recNr, Satz
    Get #1, 1, charVar
    Print "Nach dem Wahlfrei-Lesen:"
    Print charVar; "|"; Satz.inteVar; "|"; Satz.cFixVar
End Select

rndStat = 1 - rndStat: cmdRnd.Caption = rndCap(rndStat): Close #1
End Sub
```



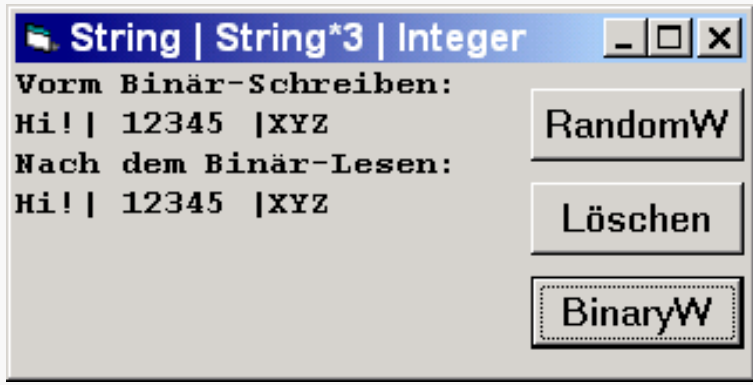
Dateien (wahlfrei / binär)

Prof. Dr. Aris Christidis


```
Private Sub cmdBin_Click()
Open "ABC.txt" For Binary As #1

Select Case binStat
Case W: Cls: charVar = "Hi!"
    Satz.inteVar = 12345: Satz.cFixVar = "XYZ"
    Print "Vorm Binär-Schreiben:"
    Print charVar; "|"; Satz.inteVar; "|"; Satz.cFixVar
    Put #1, 1, charVar
    Put #1, 4, Satz
```

(statt Deskriptor)

```
Case R: charVar = "    "
    Satz.inteVar = 0: Satz.cFixVar = ""
    Get #1, 1, charVar
    Get #1, , Satz
    Print "Nach dem Binär-Lesen:"
    Print charVar; "|"; Satz.inteVar; "|"; Satz.cFixVar
End Select

binStat = 1 - binStat: cmdBin.Caption = binCap(binStat): Close #1
End Sub
```



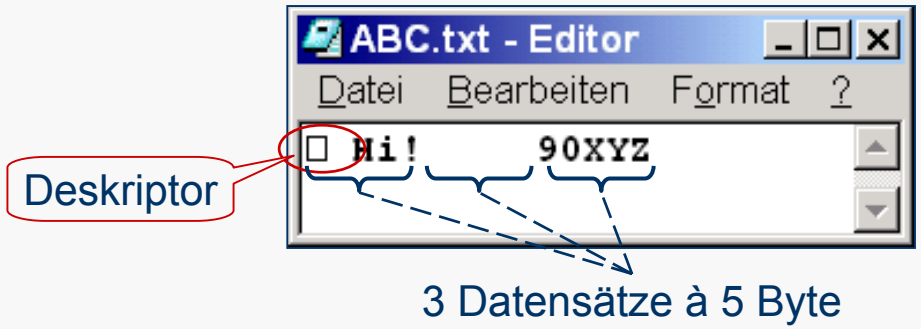
Dateien (wahlfrei / binär)

Prof. Dr. Aris Christidis

Fußnote: Im Editor werden oft Speicherungsdetails sichtbar.

binäre Datei:

wahlfreie Datei:



Die Zahl 12345 (**Integer**: 2 Byte) wird als 90 (**ASCII**) abgelegt:

12345_{10} = 0011 0000 0011 1001$_2$ = 48*2^8 + 57*2^0

Chr$(0011 0000$_2$) = Chr$(48_{10}) = "0"

Chr$(0011 1001$_2$) = Chr$(57_{10}) = "9"

Intel wendet den sog. „Little Endian“ an: zuerst (an niederwertigen Adressen) werden niederwertige („little“) Bytes gespeichert. Das beschleunigt Operationen an den weniger signifikanten Stellen (vgl.: Next i). Im Byte-Inneren besteht weiterhin „Big Endian“.

Dateien (wahlfrei / binär)

Bemerkungen zu wahlfreien u. binären Dateien :

- Wahlfreie und binäre Dateien sind Realisierungen unterschiedlicher Prinzipien (vgl. Reklame auf rechteckigen Werbetafeln oder als lineare Laufschrift).
- Binäre Dateien sind im Vergleich zu wahlfreien universeller, platzsparender, aber aufwendiger zu programmieren.
- Beide Datei-Arten erlauben Lesen und Schreiben ohne vorheriges Schließen. Das ermöglicht auch Teil-Löschung durch Umordnen des Datei-Inhalts.
- Die beabsichtigte Anwendung sollte über die Art der verwendeten Dateien entscheiden: Gefordertes Durchgehen des gesamten Datei-Inhalts (oft: Textverarbeitg.) spricht für sequentielle Dateien; selektive Veränderung erfordert wahlfreien o. binären Zugriff (oft: Bildverarbeitg).

Dateien (allgemein)

Prof. Dr. Aris Christidis

Nützliche Funktionen im Umgang mit Dateien:

`FreeFile``[(`{**`0`**|**`1`**}`**`)`**`]`
gibt die nächste verfügbare Dateinummer zurück:
`0` beschränkt Suche auf Werte 1-255, **`1`** auf 256-511.

`Kill` `PFADNAME`
löscht eine (nicht geöffnete) Datei auf der Festplatte.

`Name` `PFADNAME1` **`As`** `PFADNAME2`
benennt eine (nicht geöffnete) Datei um.

`Loc(``DATEINR`**`)`**
liefert Nr. des zuletzt beschriebenen oder gelesenen Datensatzes / Byte einer mit **`Random`** / **`Binary`** geöffneten Datei.

`Seek(``DATEINR`**`)`**
liefert Nr. des nächsten zu beschreibenden o. zu lesenden Datensatzes / Byte einer mit **`Random`** / **`Binary`** geöffneten Datei.

`LOF(``DATEINR`**`)`**
gibt die Länge einer geöffneten Datei in Byte zurück.