...

Dateien und ...

The Holy Writt by kosti

Hier werde ich mich nach einigen grundsätzlichen Ausführungen zum Thema mit folgenden Themen beschäftigen:

- Zufriff auf Dateien
  - Textdateien
    - Wie öffnet man Textdateien zum lesen ?
    - Wie kann ich Textdateien lesen?
      - Eine Textdatei Zeilenweise lesen ?
      - Bestimmte Anzahl von Zeichen lesen.
      - Variablen einlesen
    - Wie öffne ich Textdateien zum schreiben/speichern ?
    - Wie speichert man Daten in Textdateien ?
      - Zeilenweise speichern in Textdateien
      - Variablen speichern
  - Typisierte Dateien
    - Wie kann ich eine typisierte Datei öffnen ?
    - Was ist der Datensatzzeiger und was kann man damit machen ?
    - Wie schreibe ich Datensätze in eine typisierte Datei ?
    - Wie kann ich Datensätze aus typisierten Dateien lesen ?
  - Binärdateien
    - Wie öffne ich Binärdateien ?
    - Was hat es hierbei mit dem Satzzeiger an sich ?
    - Wie kann ich Daten aus eine Binärdatei lesen ?
    - Wie kann ich in eine Binärdatei schreiben ?

Auser den drei Dateiarten gibt es noch eine ganze menge andrer Dateien, die sollen jedoch in diesem Toturial nicht behandelt werden.

 Wem das Online-Lesen zur teuer ist der kann sich hier die Seite als Gezipte-Pdf-Datei runterladen 485 KB

Zuerst eine Auflistung einige Dateifunktionen (Befehle) und dessen kurze Beschreibung:

Befehl:	Beschreibung:
FreeFile	liefert eine freie Dateinummer. Eine Dateinummer benötigt man beim öffnen eine Datei um den Datenkanal später eindeutig identifizieren können. Jede Dateioperation auf die Datei bezieht siech dann auf die Dateinummer.
Open	öffnet eine Datei und ordnet ihr eine Dateinummer zu.
Print	schreibt in eine Textdatei
Write	schreibt in eine Textdatei. Im Gegensatz zur Print-Anweisung fügt die Write-Anweisung beim Schreiben in die Datei Kommas zwischen Elementen und Anführungszeichen eingeschlossener Zeichenfolgen ein.
Input	liest aus eine Textdatei.
Line Input	liest eine Zeile aus eine Textdatei
Get	liest Random oder Binärdateien
Put	schreibt in Random oder Binärdateien
Seek	setzt die Position des Datenzeigers
EOF	signalisiert das Dateiende. EOF gibt True zurück wenn sich der Datenzeiger am Dateiende befindet.
Loc	gibt die aktuelle Position des Datenzeigers.
LOF	gibt die Größe eine geöffneten Datei in Bytes zurück.
Reset	schließt alle !!! geöffneten Dateien.

Close	schließt eine oder mehrere geöffnete Dateien
CurDir	liefert das aktuelle Verzeichnis-Pfad.
ChDir	wechselt das Verzeichnis.
ChDrive	wechselt das Laufwerk.
MkDir	erstellt ein neues Verzeichnis.
RmDir	löscht ein Verzeichnis.
Kill	löscht eine Datei (VORSICHT !!!)
FileCopy	kopiert eine Datei.
Name	benennt eine Datei oder ein Verzeichnis um.
FileAttr	gibt den Zugriffsmodus für eine mit der Open-Anweisung geöffnete Dateien zurück. Möglich sind: - Input 1 - Output 2 - Random 4 - Append 8 - Binary 32
FileDateTime	gibt den Tag und die Uhrzeit der Erstellung bzw. der letzten Änderung der Datei oder des Verzeichnisses.
GetAttr	liefert die Attribute einer Datei oder eines Verzeichnisses. Möglich sind: - vbNormal 0 - Normal - vbReadOnly 1 - Schreibgeschützt - vbHidden 2 - Versteckt - vbSystem 4 - Systemdatei - vbDirectory 16 - Verzeichnis oder Ordner - vbArchive 32 - Datei wurde seit dem letzten Sichern geändert
SetAttr	Legt die Attribute für eine Datei fest (sehe GetAttr)
FileLen	liefert die Dateigröße einer ungeöffneten Datei in Bytes.

nach Oben

Der Zugriff auf Dateien

Da Windows ein Multitasking-Betriebssystem ist muss man vor dem öffnen eine Datei darüber nachdenken wie man eine Datei öffnen und welche Rechte andere Programme an der geöffneten Datei haben sollen. Man kann beim öffne eine Datei das Verhalten der Datei gegenüber anderen Anwendungen festlegen. Dabei gibt es folgende Möglichkeiten:

Modus	Beschreibung
Shared	Jeder andere Prozess hat Schreib und Leserechte für diese Datei
Lock Read	Die Datei kann durch andere Anwendungen nicht gelesen werden.
Lock Write	Andere Anwendungen haben kein Schreibrecht in die Datei
Lock Read Write	Andere Anwendungen haben keine Schreib und keine Leserechte für die Datei. (Default)

Voraussetzung für die Zugriffsmodi ist das kein anderes Programm für die Datei ein Zugriffsmodi gewählt hat. In diesem Zusammenhang ist es wichtig mögliche Fehler beim Öffnen, Lesen, Schreiben abzufangen und darauf entsprechend reagieren. Ein einfaches On Error Resume Next reicht hier nicht aus !!!

Hier noch einige Beispiele:

```
Open "dateiname" For Input Lock Read  As #1          'Andere Prozesse können die Datei nicht lesen
Open "dateiname" For Input Lock Read Write  As #1    'Andere Prozesse können die Datei nicht lesen und nicht schreiben
```

Optional zur den Zugriffsmodi für andere Programme kann man auch festlegen was das eigene Programm mit der Datei machen kann. Hier gibt es auch wiederum einige Optionen die ich hier aufführe:

Zugriff	Beschreibung
Read	Nur Leserechte
Write	Nur Schreibrechte
Read Write	Sowohl Lese als auch Schreibrechte (Default)

Die Zugriffsrechte des eigenen Programms sind nur sinnvoll in Bezug auf Typisierte und Binären-Dateien und werden mit dem Parameter Access festgelegt.

Hier noch einige Beispiele dazu:

```
Open "dateiname" For Binary Access Write As #1          ' Das Programm kann nur schreiben in die Datei
Open "dateiname" For Binary Access Read As #1           ' Nur lessen erlaubt
Open "dateiname" For Binary Access Read Lock Write As #1    ' Das Programm kann nur lesen, andere Programme nur schreiben
```

nach Oben

Textdateien

Textdateien sind Zeichen und Zeilenweise orientiert. Jeder Zeile wird durch die ASCII-Zeichen 13 und 10 abgeschlossen. Die Länge der Zeile wird allein durch die Position der beiden ASCII-Zeichen bestimmt. Man kann sich das so vorstellen:

```
Das ist ein Text in eine Textdatei|10|13
Die Zeilenlänge wird durch das CR bestimmt|10|13
Es können auch ganz lange Zeilen seinablablablabla|10|13
```

Textdateien können entweder zum lesen oder zum schreiben geöffnet werden. Das kombinieren der beiden Modi ist bei Textdateien nicht möglich (es sei den man schließt die Datei bevor in einem anderem Modus darauf zugegriffen wird)

nach Oben

Wie öffnet man Textdateien zum lesen ?

Um auf eine Textdatei zugreifen können muss diese erst geöffnet werden. Dies Gehschiet mit dem Befehl Open (Parameter in Klammern sind optional)

```
Open Pfadname For Modus [Access Zugriff] [Sperre] As [#]Dateinummer [Len=Satzlänge]
```

Die Syntax der Open-Anweisung besteht aus folgenden Teilen:

Teil	Beschreibung
Pfadname	Pfad und Dateiname der zu öffnenden Datei
Modus	Legt den Zugriffsmodus für die Datei fest. Möglich sind: Append - Schreibzugriff, Daten werden an die Datei angehängt. Binary - Zugriff auf die Datei im Binärmodus, dazu jedoch später mehr Input - Lesezugriff, Datei kann nur gelesen werden. Output - Schreibzugriff, Daten können nur geschrieben werden, eine vorhandene Datei würde überschrieben
Zugriff	Legt fest was das eigene Programm mit der Datei machen kann. (sehe

	oben)
Sperre	Legt fest wie und ob andere Programme mit der Datei machen können. (sehe oben)
Dateinummer	Eine gültige Dateinummer im Bereich von 1 bis 511 (einschließlich). Mit der FreeFile-Funktion erhält man die nächste verfügbare Dateinummer. Unter dieser Nummer erfolgen dann alle weitere Zugrigge des Programms auf die Datei.
Satzlänge	Zahl kleiner oder gleich 32.767 (Bytes). Bei Dateien mit wahlfreiem Zugriff ist dies die Datensatzlänge, bei sequentiellen Dateien die Anzahl der gepufferten Zeichen

Die erforderliche Modusangabe ist einer der größten Nachteile eine Textdatei.

Beispiele für Öffnen der AUTOEXEC.BAT zum lesen:
```
Open "AUTOEXEC.BAT" For Input As #1
```

Beispiel zum öffnen eine Textdatei zum schreiben:
```
Open "C:\Test.txt" For Output As #1
```

Beispiel um Daten an eine bestehende Textdatei anzuhängen:
```
Open "C:\Test.txt" For Append As #1
```

Wenn die Open-Anweisung von Misserfolg war, tritt im allgemeinen ein Laufzeitfehler 53 auf. Diesen kann man mit der On Error Goto-Anweisung jedoch abfangen:

```
Private Sub Egal()
    On Error Goto Fehler
    Dim Datei as String
    Dim Fnr as Long
    Datei  = "C:\Test.txt"
    Fnr = FreeFile
    Open Datei For Input As Fnr
    ...
    Close Fnr
    Exit Sub
Fehler:
    MsgBox "Es trat ein Fehler beim Öffnen der Datei !", 16,"Problem"
    Exit Sub
    Resume Next
End Sub
```

nach Oben

Wie kann ich Textdateien lesen?

Bei Lesen eine Textdatei muss man zwischen drei verschiedenen Mögöligkeiten unterscheiden.

1. Zeilenweise Einlesen
2. Bestimmte Anzahl von Zeichen lesen
3. Variablen einlesen.

nach Oben

Eine Textdatei Zeilenweise lesen ?

Zeilenweise lesen bedeutet das, mit dem Line Input-Befehl, aus eine Datei alle Zeichen bis zum Zeilenende (wir erinnern uns ASCII 10 und 13) gelesen und in eine Variable gespeichert werden. Die jeweils nächste Zeile wird dann mit dem nächstem Line Input-Befehl gelesen. *Die Zeilenend-Makierungen werden dabei nicht in der Variable gespeichert !*

Das Zeilenweise lesen eine Textdatei erreicht man mit dem Befehl:

Line Input *#Dateinummer, Variablenname*

Parameter	Beschreibung
Dateinummer	Eine beliebige gültige Dateinummer.
Variabelenname	Ein gültiger Variabelenname vom Typ Variant oder String.

Mit folgendem Code kann man beispielweise ein Listenfeld mit Einträgen füllen die in eine Textdatei gespeichert sind:

```
While Not EOF(dateinr)
    Line Input #dateinr , variable
    List1.AddItem variable
Wend
```

Hierbei werden solange Einträge in ein Listfeld hinzugefügt bis das Dateiende erreicht wird (While Not EOF)
Bei Line Input muss man den Umweg über eine Variable machen um den Text eine Eigenschaft eines Steuerelementes übergeben. Bei:

```
Line Input #dateinr , Text1.Text
```

Bekommt man ein Laufzeitfehler !!! Einige weitere Beispiele:

```
Dim Zeiger As Long
Dim Name() as String
Dim Adresse() As String
Dim Tel() As String
Dim temp
Dim temp1
...
Open "C:\Adressen.txt" For Input As 1
    Rdim Preserve Name(0)
    Rdim Preserve Adresse(0)
    Rdim Preserve Tel(0)
    While Not EOF(1)
        Line Input #1, Name(Ubound(Name))
        Line Input #1, Adresse(Ubound
(Adresse))
        Line Input #1, Tel(Ubound(Tel))
        Rdim Preserve Name(Ubound(Name)+1)
        Rim Preserve Adresse(Ubound(Adresse)
+1)
        Rdim Preserve Tel(Ubound(Tel)+1)
    Wend
Close 1
```

Dieses kleine Progi liest aus eine Textdatei Name, Adresse und die Tel.Nr und speichert es in die drei Arrays. Nächstes Beispiel:

```
Dim Monat(12) As String
Dim Wochentag(7) As String
Dim temp
...
Open "C:\Test.txt" For Input As 1
    For temp = 1 to 7
        Line Input #1, Wochentag(temp)
    Next temp
    For temp = 1 to 12
        Line Input #1, Monat(temp)
    Next temp
Close 1
```

Ich glaube dieser Code erklärt sich von selbst.

nach Oben

Bestimmte Anzahl von Zeichen lesen.

Diese Möglichkeit des Dateilesens ermöglicht den gesamten Inhalt der Textdatei, also inklusive der Zeilenend-Zeichen, zu lesen. Das kann man sich zunutze machen wenn man ein Multiline-Textfeld mit dem Inhalt eine Textdatei füllen will. Um eine bestimmte Anzahl von Zeichen aus eine Textdatei zu lesen verwendet man folgenden Befehl:

```
Variable = Input(Zahl, [#]Dateinummer)
```

Parameter	Beschreibung
Zahl	Ein beliebiger gültiger numerischer Ausdruck, der die Zahl der zurückzugebenden Zeichen angibt.
Dateinummer	Eine beliebige gültige Dateinummer
Variable	Enthält die Zurückgegebene Zeichen aus der Datei

Zur Input gibt es einfach nicht viel zusagen. Es liest einfach die angegebene Anzahl von Zeichen aus der Datei und übergebt sie der Variable. Es sollte vielleicht beachtet werden das man bei dem munterem lesen nicht über das Dateiende hinauskommt, sonst gibt es eine Fehlermeldung. Einige Beispiele sollten das ganze verständlicher machen.

Beispiel:

```
Dim temp as String
Open dateiname For Input As dateinummer
    Temp = Input(LOF(dateinummer) ,
dateinummer)
Close dateinummer
Text1.Text = temp
```

Dieser Code liest die ganze Textdatei (LOF(dateiname) gibt die Dateilänge zurück) in die temp-Variable und dann wird der Text in eine TextBox angezeigt. Hierbei sollte beachtet werden das die MultiLine-Eigenschaft der TextBox auf True gesetzt ist sonst werden die Zelienendzeichen als senkrechte Striche in der TextBox angezeigt und es erfolgt kein Zeilenumbruch.

Beispiel:

```
Dim temp as String
Open dateiname For Input As dateinummer
While Not EOF(dateinummer)
    Temp = Input( 1 , dateinummer)
    List1.AddItem temp
Wend
Close dateinummer
```

Hier wird die Textdatei ein Zeichen nach dem anderem solange gelesen bis das Dateiende erreicht wird. Die gelesener Zeichen werden in eine ListBox angezeigt.

Beispiel:

```
Dim temp as String
Open dateiname For Input As dateinummer
    Temp = Input( 100  , dateinummer)
Close dateinummer
Text1.Text = temp
```

Hier werden die ersten 100 Zeichen der Textdatei gelesen und in eine Textbox angezeigt. Sollte die Textdatei keine 100 Zeichen haben kommt es zur eine Fehlermeldung.

nach Oben

Variablen einlesen

An dieser Stelle möchte ich auf eine Besonderheit mit der Deutschen Win-Version eingehen. Variablen werden nämlich mit einem Komma voneinander in der Datei getrennt, gleichzeitig werden aber Gleitkommawerte mit einen Komma getrennt. Wundert euch also nicht wenn nach dem lesen von Gleitkommawerten die Zahlen nicht übereinstimmen. (Hier vielleicht ein Vorschlag am Rande: Gleitkommazahlen vor dem Speichern mit einen Faktor X multiplizieren, und nach dem lesen wieder Dividieren).
Wie liest man denn Variablen ein? Dazu gibt es Folgende Funktion:

Input *#Dateinummer, Varliste*

Parameter	Beschreibung
Dateinummer	Eine beliebige gültige Dateinummer.
Varliste	Liste mit Variablen, die jeweils durch ein Komma als Listentrennzeichen voneinander getrennt sind und denen die aus der Datei gelesenen Werte zugewiesen werden. Es kann weder ein Datenfeld noch eine Objektvariable sein. Sie können jedoch Variablen verwenden, die ein Element eines Datenfeldes oder benutzerdefinierten Typs beschreiben.

Diese Funktion liest aus der Textdatei Werte ein und weist die den Variablen zu.

Beispiel:

```
Dim Sek As Double
Dim Min As Double
...
Input #dateinummer , Sek , Min
```

Dieses Beispiel liest aus eine Datei zwei Werte und weist sie den Variablen Sek und Min zu. Habt ihr gut aufgepasst ? Wird es in Deutschland funktionieren ? Bekommt man richtige Werte den Variablen zugewiesen? Was glaubst du ?

Beispiel:

```
Dim Text1 As String
Dim Zahl1 As Long
...
Open "C:\Test.txt" For Input As #1
    Input #1, Text1, Zahl1
Close #1
```

Hier wird erstens eine Textvariable und dann ein Zahl gelesen. Dieses Beispiel sollte einwandfrei funktionieren da Longwerte keine Nachkommastellen haben. (außer das der String ein Komma haben sollte)

Beispiel:

```
Dim Text1 As String
Dim Zahl1 As Long
```

```
...
Open "C:\Test.txt" For Input As #1
    Input #1, Text1
    Input #1, Zahl1
Close #1
```

Hier wird das gleiche getan wie im vorherigem Beispiel jedoch werden zwei Input-Funktionen verwendet.So kommen wir jetzt zum nächstem Thema.

nach Oben

Wie öffne ich Textdateien zum schreiben/speichern ?

Zuerst muss man überlegen ob eine möglicherweise schon vorhandene Datei behalten werden soll und die neuen Daten an die vorhandene angehängt werden sollen, oder soll eine komplett neue Datei angelegt werden? Wenn man eine neue Datei anlegen will, egal ob schon eine mit dem gleichem Namen im gleichem Verzeichnis vorhanden ist, die dann gnadenlos gelöscht wird, übergibt man dem Open-Befehl den Parameter For Output. Dadurch wird eine leere Datei angelegt.

```
Open dateiname For Output As #1
```

Was die anderen Parameter bedeuten muss ich hier wohl nicht noch mal erklären.
Um auf die andere Möglichkeit eingehen, wo eine möglicherweise bestehender Datei auf jeden Fall beibehalten werden soll und die neuen dazukommenden Daten an die Datei angehängt werden sollen (also an das Ende der Datei schreiben), tauschen wir einfach For Output durch For Append.

```
Open dateiname For Append As #1
```

Ist jetzt keine Datei vorhanden wird einfach eine neue angelegt, es kommt zur keine Fehlermeldung.

nach Oben

Wie speichert man Daten in Textdateien ?

Beim speichern in eine Textdatei muss man ähnlich wie beim lesen zwischen zwei verschiedenen Mögöligkeiten unterscheiden.

1. Zeilenweise speichern
2. Variablen speichern

Es ist dabei egal ob die Datei neu angelegt wurde oder ob man an eine schon vorhandene Datei Daten anhängt. Die Ausgabefunktionen unterscheiden sich nicht voneinander.

nach Oben

Zeilenweise speichern in Textdateien

Das ist wohl die einfachste Mögöligkeit Text in eine Datei zu speichern. Die Befehlsyntax dazu sieht folgendermaßen aus:

```
Print #Dateinummer, [Ausgabeliste]
```

Parameter	Beschreibung

Dateinummer	Eine beliebige gültige Dateinummer.
Ausgabeliste	Ausdruck oder Liste mit Ausdrücken, die ausgegeben werden sollen. Es können Texte, Zahlen oder Variabelenwerte sein.

Das ist zwar die einfachste Art Datei zu speichern, einige Besonderheiten muss man aber trotzdem beachten. So kann man das Zeilentrennzeichen mit einem Semikolon (;) unterdrücken, mit einem Komma (,) kann man ein Tabulator einfügen, und das direkte speichern von Zahlen ist auch etwas eigensinnig. Nehmen wir dazu am besten ein Beispiel:

Beispiel:

```
Dim a
a = FreeFile
b = 10
Open "C:\text.txt" For Output As a
    Print #a, "Zeile1"
    Print #a, "Zeile2", "Zeile3"
    Print #a,                                   'leere Zeile 4
    Print #a, "Zeile5"
    Print #a, "Zeile6";
    Print #a, "noch Zeile6"
Close a
```

Wenn man sich jetzt die Textdatei mit dem Notepad anschaut sieht es so:

```
Zeile1
Zeile2        Zeile3

Zeile5
Zeile6noch Zeile6
```

Was ist denn passiert?

`Print #a, "Zeile1"` hier wurde ganz normal eine Zeile in die Textdatei geschrieben.

`Print #a, "Zeile2", "Zeile3"` hier hat das Komma zwischen „Zeile2" und „Zeile3" nicht wie vermutet den Text "Zeile3" in eine neue Zeile geschrieben. Es wurde einfach ein Tabulatorzeichen hinter dem Text "Zeile2" hinzugefügt (ASCII 10) was bewirkt das der nachfolgender, beim betrachten mit Notepad, Text einfach ein bisschen nach rechts verschoben wird. Es werden keine Leerzeichen (SPACE oder ASCII 32) zwischen den Texten Zeile2 und Zeile3 eingefügt !!!

`Print #a,` - hier wurde einfach eine leere Zeile hinzugefügt.

`Print #a, "Zeile5"` hier wieder die normale Ausgabe.

`Print #a, "Zeile6";` - das Semikolon hat hier das Zeilenendzeichen unterdrückt, so das die nächste Print- Ausgabe nicht in eine neue Zeile beginnt sondern direkt nach dem letztem Zeichen der aktuelle Zeile.

`Print #a, "noch Zeile6"` hier kann man nicht erkennen ob der Text in eine neuen Zeile geschrieben wird. Dazu muss man den letzten Print-Befehl genauer ansehen.

So jetzt ist die Verwirrung groß ! Zur Verdeutlichung noch die einzelnen Möglichkeiten als einzelne Progis mit den entsprechenden Aussehen der Dateien:

Möglichkeit 1:

```
Print #a, "Zeile1"
Print #a, "Zeile2"
Print #a, "Zeile3"
Print #a, "Zeile4"
```

Die Datei sieht dann so aus.

```
Zeile1
Zeile2
Zeile3
Zeile4
```

Möglichkeit 2:

```
Print #a, "Zeile1",
Print #a, "Zeile2",
Print #a, "Zeile3",
Print #a, "Zeile4",
```

Die Datei sieht dann so aus.

```
Zeile1     Zeile2                Zeile3                Zeile4
```

Möglichkeit 3:

```
Print #a, "Zeile1";
Print #a, "Zeile2";
Print #a, "Zeile3";
Print #a, "Zeile4";
```

Die Datei sieht dann so aus.

```
Zeile1Zeile2Zeile3Zeile4
```

Natürlich kann man das ganze miteinander kombinieren. Weiter ist es auch möglich Variabelenwerte, es sei den String oder Zahl, Objekteigenschaften, z.B. Text1.Text oder Command1.Caption oder auch andere, dem Print-Befehl zur übergeben. Handelt es sich bei den Werten nicht um Zahlen verhält sich die Ausgabe gleich wie vorhin, werden jedoch zahlen gespeichert (also kein Text) sind einige eigenheiten zur beachten.

Werden Zahlen direkt oder aus eine Variable gespeichert, fügt der Print-Befehl (ohne zufragen) ein Leerzeichen vor der Zahl und ein Leerzeichen hinter der Zahl !!! Egal ob die Ausgabe mit Komma oder Semikolon abgeschlossen wird. Dies kann man aber leicht umgehen (wenn erwünscht). Das Verhalten zeige ich am besten wieder mit einem Beispiel.

Beispiel:

```
Dim a
Dim b As Long
a = FreeFile
b = 10
Open "C:\text.txt" For Output As a
    Print #a, 10
    Print #a, b
    Print #a, 10, 10, 10
    Print #a, b, b, b
    Print #a, 10; 10; 10
    Print #a, b; b; b
    Print #a, Str$(10); Str$(10); Str$(10)
    Print #a, Str$(b); Str$(b); Str$(b)
    Print #a, CStr(10); CStr(10); CStr(10)
    Print #a, CStr(b); CStr(b); CStr(b)
    Print #a, "101010"
Close a
```

Die Textdatei sieht im Notepad so aus (# steht für Leerzeichen):

```
#10#
#10#
#10#            #10#            #10#
#10#            #10#            #10#
#10##10##10#
#10##10##10#
#10#10#10
#10#10#10
101010
101010
101010
```

Die Erklärung:

```
Print #a, 10
```
```
Print #a, b
```
- die zwei Zeilen verhalten sich gleich. Es wird jeweils ein Leerzeichen vor der Zahl und eins nach der Zahl eingefügt. Am Ende der Zeile kommt ein Zeilenendzeichen.

```
Print #a, 10, 10, 10
```
```
Print #a, b, b, b
```
- hier wiederum gleiches Verhalten, jedoch wird noch (durch das Komma) ein Tabulatorzeichen (ASCII 10) eingefügt

```
Print #a, 10; 10; 10
```
```
Print #a, b; b; b
```
- hier werden die Zahlen zwar in eine Zeile gespeichert, aber mit jeweils ein Leerzeichen vor der Zahl und eins nach der Zahl.

```
Print #a, Str$(10); Str$(10); Str$(10)
```
```
Print #a, Str$(b); Str$(b); Str$(b)
```
- mit Str$() ist es möglich das hinzufügen des Leerzeichen nach der Zahl zu unterdrücken.

```
Print #a, CStr(10); CStr(10); CStr(10)
```
```
Print #a, CStr(b); CStr(b); CStr(b)
```
- und durch CSrt() werden sowohl das vorgestellt als auch das nachgestellte Leerzeichen entfernt.

```
Print #a, "101010"
```
- dieser Zeile speichert reinen Text und wurde nur zur Veranschaulichung der Abstände hinzugefügt.

Soviel zum Zeilenweise speichern. Kommen wir nun zur...

nach Oben

Variablen speichern

Man kann natürlich auch in Textdateien Variabelenwerte speichern. Wie man die wieder auslesen kann haben wir bereits erfahren. Das speichern von Variablen geschieht mit:

Write *#Dateinummer*, [*Ausgabeliste*]

Parameter	Beschreibung
Dateinummer	Eine beliebige gültige Dateinummer.
Ausgabeliste	Eine oder mehrere (durch Kommas getrennte) numerische Ausdrücke oder Zeichenfolgenausdrücke, die in eine Datei geschrieben werden sollen.

Daten, die mit der Write #-Anweisung geschrieben worden sind, werden normalerweise mit der Input #-Anweisung aus einer Datei gelesen.

Mehrere Ausdrücke können durch Leerzeichen, Semikolons oder Kommas voneinander getrennt werden. Leerzeichen und Semikolon haben dieselbe Wirkung.

Wenn Sie mit Write # Daten in eine Datei schreiben, gelten mehrere allgemeingültige Voraussetzungen, damit die Daten unabhängig vom Gebietsschema mit Input # immer richtig gelesen und interpretiert werden können:

- Numerische Daten werden immer mit dem Punkt als Dezimaltrennzeichen geschrieben.
- Daten vom Typ Boolean, werden entweder mit #TRUE# oder #FALSE# ausgegeben.
  Die Schlüsselwörter True und False werden nicht übersetzt, unabhängig vom Gebietsschema.
- Daten vom Typ Date werden immer im universellen Datums- und Zeitformat in die Datei geschrieben.
  Fehlt entweder die Datums- oder die Zeitangabe, oder ist sie gleich Null, so wird nur der vorhandene Teil in die Datei geschrieben.
- Wenn Daten in der Ausgabeliste den Wert Empty haben, werden keine Daten in die Datei geschrieben.
  Haben die Daten jedoch den Wert Null, so wird #NULL# in die Datei geschrieben.
- Wenn die Daten in der Ausgabeliste Null-Daten sind, wird #NULL# in die Datei geschrieben.
- Wenn die Daten den Wert Error haben, wird #ERROR Fehler-Code# ausgegeben.
  Das Schlüsselwort Error wird unabhängig vom Gebietsschema nicht übersetzt.

Im Gegensatz zur Print #-Anweisung fügt die Write #-Anweisung beim Schreiben in die Datei Kommas zwischen Elementen und Anführungszeichen eingeschlossener Zeichenfolgen ein. Sie brauchen daher in der Liste kein explizites Trennzeichen anzugeben.

Die Write #-Anweisung fügt einen Zeilenumbruchzeichen, d.h. eine Kombination aus Wagenrücklauf und Zeilenvorschub (Chr(13) + Chr(10)), ein, nachdem sie das letzte Zeichen aus der Ausgabeliste in die Datei geschrieben hat.

Ein Praktisches Beispiel noch an dieser Stelle, und dann ist das Thema auch schon fast fertig.

Beispiel:

```
Dim a
Dim b
a = FreeFile
b = 10
Open "C:\text.txt" For Output As a
    Write #a, 10, 10, 10
    Write #a, b, b, b
    Write #a, 10; 10; 10
    Write #a, b; b; b
    b = True
    Write #a, b, b, b, b
    b = 0.001
    Write #a, b, b, b
    b = "Hallo"
    Write #a, b, b
    Write #a, "Hallo", "Hallo"
Close a
```

Die Datei betrachtet mit Notepad sieht so aus:

```
10,10,10
10,10,10
10,10,10
10,10,10
#TRUE#,#TRUE#,#TRUE#,#TRUE#
.001,.001,.001
"Hallo","Hallo"
"Hallo","Hallo"
```

Wie man sieht ist hier egal ob man die Ausgabeliste durch Kommas oder Semikolons trennt, in der Datei werden auf jeden Fahl die Werte durch Kommas getrennt.

Bis jetzt haben wir schon eine Menge Wissen gesammelt. Jetzt können wir uns mit Textdateien aus. Was jetzt noch bleibt ist zwar nicht mehr so umfangreich aber genau so interessant.

nach Oben

Typisierten Dateien

Typisierten Dateien ist eine Art von Textdateien die sich jedoch besonderes zum speichern von numerischen Daten und Nutzertypen eignen.

Diese Art von Dateien besitzen eine konstante Datensatzlänge, zwischen den einzelnen Datensätzen ist kein Trennzeichen vorhanden. Man muss also wissen wie groß ein Datensatz ist.

Für das speichern von Zeichenketten ist eine typisierte Datei weniger geeignet, da auch für ein Leerstring die einmal definierte Anzahl von Zeichen freigehalten wird.

nach Oben

Wie kann ich eine typisierte Datei öffnen ?

Auf eine typisierte Datei kann gleichzeitig im Schreibmodus als auch im Lesemodus zugegriffen werden. Das öffnen eine typisierten Datei geschieht mit:

Open *dateiname* **For Random As** *dateinummer* **Len** = *satzlänge*

Parameter	Beschreibung
Dateiname	Pfad und Dateiname der zu öffnenden Datei
Dateinummer	Eine gültige Dateinummer im Bereich von 1 bis 511 (einschließlich). Mit der FreeFile-Funktion erhält man die nächste verfügbare Dateinummer. Unter dieser Nummer erfolgen dann alle weitere Zugrigge des Programms auf die Datei.
Satzlänge	Zahl kleiner oder gleich 32.767 (Bytes) die die Datensatzlänge angibt.

Was hier vielleicht noch näher erklärt werden sollte ist der Parameter Len. Dieser bestimmt die Datensatzlänge, also die Länge der Daten in „eine Zeile". Übereinstimmt die Datensatzlänge nicht mit der tatsächlichen Länge bekommt man beim lesen der Datei eine Fehlermeldung, und im schlimmstem Falle falsche Werte gelesen. Wird keine Länge angegeben legt VB automatisch 128 Byte fest. Ein Beispiel dazu:

```
Private Type Datensatz
    Nr As Long
    Nam As String * 20
    Vornam As String * 20
End Type
...
Dim a
Dim Data As Datensatz
a = FreeFile
Open "C:\text.txt" For Random As a Len = Len(Data)
...
```

Es wird erst mal ein Datentyp definiert der sich aus einem Longwert, einem String der Größe 20 Zeichen und noch einem String der Größe 20 Zeichen zusammensetzt. Danach wird der Variable Data dieser Datentyp zugewiesen. Mit dem Len(Data) wird die Länge der Variable bestimmt und dem Len-Parameter übergeben. Diesen Beispiel werden wir später noch bisschen erweitern.

nach Oben

Was ist der Datensatzzeiger und was kann man damit machen ?

Bei typisierten Dateien gibt es eine Art Zeiger der immer auf den aktuellen Datensatz zeigt. Nach dem öffnen der Datei zeigt der Datensatzzeiger immer auf den ersten Datensatz. Der erste Datensatz bei typisierten Dateien hat immer die Nummer 1. Wird aus der Datei gelesen oder werden Daten geschrieben, wird der Datensatzzeiger immer um ein Datensatz weiter bewegt. D.h wenn man den 20. Datensatz liest zeigt der Datensatzzeiger nach dem Lesevorgang auf den Datensatz 21. Das ist wichtig zu wissen wenn man in eine Schleife Datensätze lesen will.

Will man jetzt ab einem bestimmten Datensatz lesen kann man den Datensatzzeiger direkt an den Datensatz bewegen. Dazu gibt es ein einfaches Befehl:

Seek *dateinummer, datensatznummer*

Eine genauere Beschreibung ist hier wohl nicht mehr nötig. Sollte der Zeiger auf ein Datensatz gesetzt werden der noch nicht existiert kommt es zur keine Fehlermeldung !!! Es ist dadurch möglich Datensätze zu lesen die es gar nicht gibt. Ob dabei jedoch was sinnvolles rauskommt ist fraglich. Anderes ist es beim Speichern. Hier könnte sich die Tatsache durchaus nützlich machen, allerdings kann man nicht sagen welche Daten zwischen dem letztem vorhandenem Datensatz und dem neuem enthalten sind. Der Speicherplatz für mögliche Daten wird aber reserviert

und bis zum erstem beschreiben der "leeren" Datensätze steht dar nur Müll drin !!!!

Wenn man jetzt aber umgekehrt die aktuelle Datensatznummer wissen will ? Dann nutzt man Seek als Funktion:

Datensatznummer = **Seek** (*dateinummer*)

Hier spare ich mir auch die Beschreibung, aber im Zusammenhang möchte ich noch beschreiben wie man die Anzahl der vorhandenen Datensätze herausbekommen kann. Ist nur bisschen Mathematik. Und zwar:

Mit dem öffnen der Datei...

```
Open dateiname For Random As dateinummer Len = Len(Data)
Anzahl = LOF(dateinummer) / Len(Data)
```

Oder ohne öffnen der Datei...

```
Anzahl = FileLen(dateiname) / Len(Data)
```

In beiden Fällen enthält Anzahl die Anzahl der Datensätze.

nach Oben

Wie schreibe ich Datensätze in eine typisierte Datei ?

Das schreiben ist wie bei den Textdateien gar nicht so schwer. Der zugehöriger Befehl dazu ist:

Put *dateinummen*, [*datensatznummer*], *variable*

Parameter	Beschreibung
Dateinummer	Eine gültige Dateinummer im Bereich von 1 bis 511 (einschließlich). Mit der FreeFile-Funktion erhält man die nächste verfügbare Dateinummer. Unter dieser Nummer erfolgen dann alle weitere Zugrigge des Programms auf die Datei.
datensatznummer	Nummer des Datensatzes in den Dateien geschrieben werden sollen. Ist nicht zwingend erforderlich.
variable	Name der Variablen, die die auf den Datenträger zu schreibenden Daten enthält.

Jetzt machen wir an dem vorherigem Beispiel weiter.

```
Private Type Datensatz
    Nr As Long
    Nam As String * 20
    Vornam As String * 20
End Type
...
Dim a
Dim Data As Datensatz
a = FreeFile
Open "C:\text.txt" For Random As a Len = Len(Data)
    Data.Nr = 1
    Data.Nam = "Testname1"
    Data.Vornam = "Testvorname1"
    Put a, , Data
    Data.Nr = 2
    Data.Nam = "Testname2"
    Data.Vornam = "Testvorname2"
    Put a, , Data
Close 1
```

So hier arbeiten wir weiter mit dem selbstdefiniertem Datentyp, weisen der Variable die entsprechender Daten zu und speichern das ganze in die Datei. Die Daten werden in den aktuellen Datensatz geschrieben, dann automatisch der Datensatzzeiger erhöht. Man könnte das ganze auch so machen:

```
Open "C:\text.txt" For Random As a Len = Len(Data)
    Data.Nr = 1
    Data.Nam = "Testname1"
    Data.Vornam = "Testvorname1"
    Put a,1 , Data
    Data.Nr = 2
    Data.Nam = "Testname2"
    Data.Vornam = "Testvorname2"
    Put a,2 , Data
Close 1
```

Hier sagen wir direkt in welchen Datensatz geschrieben werden soll (Put a,1,...) Je nach dem welche Möglichkeit man benutzt werden die Daten in die Datensätze geschrieben. Die zwei letzten Beispiele schreiben in die gleichen Datensätze, betrachten wir jetzt ein ähnlichen fall:

```
Open "C:\text.txt" For Random As a Len = Len(Data)
    Seek a, 10
    Data.Nr = 1
    Data.Nam = "Testname1"
    Data.Vornam = "Testvorname1"
    Put a, , Data
    Data.Nr = 2
    Data.Nam = "Testname2"
    Data.Vornam = "Testvorname2"
    Put a, , Data
Close 1
```

Hier werden die Daten in die Datensätze 10 und 11 geschrieben, da wir vor dem Schreiben den Satzzeiger auf die Position 10 gestellt haben. Und zur vewirung noch ein Beispiel:

```
Open "C:\text.txt" For Random As a Len = Len(Data)
    Seek a, 10
    Data.Nr = 1
    Data.Nam = "Testname1"
    Data.Vornam = "Testvorname1"
    Put a, 1, Data
    Data.Nr = 2
    Data.Nam = "Testname2"
    Data.Vornam = "Testvorname2"
    Put a, , Data
Close a
```

Gleich dazu eine Frage. In welche Datensätze werden die zwei Put-Ausgaben gemacht ? Sind es:
A - 10 und 11
B - 1 und 10
C - 1 und 2
D - 10 und 2

Wer kann diese Frage beantworten ?
Versuchen wir den Code zur Analisieren. Erstens wird die Datei geöffnet, das soll außer acht bleiben.
Dann wird der Satzzeiger auf den Datensatz 10 verschoben, und der Variable Data einige Werte zugewiesen.
Jetzt wird die Variable mit dem Put-Befehl in dem erstem Datensatz gespeichert, und der Satzzeiger automatisch auf den nächsten Datensatz gebracht. Wieso werden die Daten im ersten Datensatz gespeichert?
Schauen wir uns den Put-Befehl genauer an.

```
Put a, 1, Data
```

Was heißt das genauer?
Put - schreibe
a - in die Datei mit der Datennummer die in der Variable a steht
1 - in den ersten Datensatz
Data - die Daten die Data enthält, und verschiebe den Satzzeiger um eins.
Jetzt zeigt der Satzzeiger auf zwei. Weiter im Code werden der Data-Variable wieder neue Werte zugewiesen, und mit dem nächstem Put-Befehl in den aktuellen, also auf den der Datensatzzeiger zeigt, geschrieben. Da der Datensatzzeiger auf den Datensatz zwei zeigt werden die Daten in den Datensatz zwei geschrieben. Und somit ist die Antwort C richtig. Sie haben gewonnen !!!

nach Oben

Wie kann ich Datensätze aus typisierten Dateien lesen ?

Jetzt lernen wir noch ein Befehl kennen und dann können die typisierten Dateien kommen.
Zum lesen benutzt man:

Get *dateinummer* ,[*datensatznummer*], *variable*

Parameter	Beschreibung
Dateinummer	Eine gültige Dateinummer im Bereich von 1 bis 511 (einschließlich). Mit der FreeFile-Funktion erhält man die nächste verfügbare Dateinummer. Unter dieser Nummer erfolgen dann alle weitere Zugrigge des Programms auf die Datei.
datensatznummer	Nummer des Datensatzes in den Dateien geschrieben werden sollen. Ist nicht zwingend erforderlich.
variable	Name der Variablen, die die auf den Datenträger zu schreibenden Daten enthält.

Dieser Befehl verhält sich gleichermaßen wie Put, nur mit dem Unterschied das hier Datensätze gelesen und nicht geschrieben werden. Aus diesem Grund verzichte ich hier wiederum auf eine Detaillierte Beschreibung und komme gleich zur einem Beispiel in dem gezeigt wird wie man auf einfache Weise die die Position, Größe eines Fensters beim schließen speichert, und bei erneutem starten des Programms das Fenster wieder an der letzte Position, in der letzte Größe öffnet.

```
Private Type Koordinaten
    X As Long
    Y As Long
End Type
...
Public Sub Form_Load()
If FileExist("WindowPrefs.dat") Then
    Dim Fnr
    Dim Data As Koordinaten
    Fnr = FreeFile
    Open "WindowPrefs.dat" For Random As Fnr Len = Len(Data)
        Get Fnr, , Data
        Form1.Top = Data.X
        Form1.Left = Data.Y
        Get Fnr, , Data
        Form1.Width = Data.X
        Form1.Height = Data.Y
    Close Fnr
End If
End Sub
'...
' hier geht das Programm weiter
'...
Public Sub Form_Unload(Cancel As Integer)
Dim Fnr
Dim Data As Koordinaten
Fnr = FreeFile
Open "WindowPrefs.dat" For Random As Fnr Len = Len(Data)
    Data.X = Form1.Top
    Data.Y = Form1.Left
    Put Fnr, , Data
    Data.X = Form1.Width
    Data.Y = Form1.Height
    Put Fnr, , Data
Close Fnr
End Sub
...
' und noch eine Funktion mit der man das Vorhandensein
' eine Datei überprüft
Public Function FileExist(Dateiname As String) As Boolean
    On Error GoTo Fehler
    FileExist = Dir$(Dateiname) <> ""
    Exit Function
Fehler:
    FileExist = False
    Resume Next
End Function

Private Sub Command1_Click()
    Unload Me
    End
End Sub
```

Dieser Code funktioniert jedoch nur dann einwandfrei wenn die StartUpPosition des Fenster auf 0 oder 3 steht, und die WindowState-Eigenschaft auf 0 steht, weiter falls das Programm mit End Beendet werden soll (Menü, Button)

sollte vor dem End-Befehl Unload Me stehen damit die Form_Unload Prozedur aufgerufen wird. Was passiert in dem Code genauer ?
In der Form_Load Prozedur wird erst geprüft ob die Datei WindowPrefs.dat überhaupt schon existiert (es kann ja sein das der User die Datei gelöscht hat). Ist die Datei nicht vorhanden wird das Fenster ganz normal angezeigt, ist sie jedoch vorhanden werden erst die obere und linke Position des Fensters gelesen und in die zugehörigen Eigenschaften gespeichert, dann werden die Breite und die Höhe des Fensters gelesen und wiederum in die entsprechenden Eigenschaften geschrieben, und die Datei Geschlossen. Beim schließen des Fensters werden die entsprechenden Fenstereigenschaften in die Datei geschrieben. Falls die Datei noch nicht existiert wird sie automatisch angelegt.
Die Funktion FileExist ist eine universale Funktion mithilfe dessen man ganz leicht überprüfen kann ob eine Datei schon existiert, in dem Fahl liefert FileExist True, oder ob die Datei noch nicht vorhanden ist, hier wird dann False geifert.

nach Oben

Binärdateien

Eine Binärdatei ist eigentlich eine spezielle Form eine typisierten Datei. Die Datensatzlänge kann vom Datensatz zur Datensatz unterschiedlich sein, was alleine vom verwendetem Datentyp der Variablen abhängt. Es ist hierbei also möglich das man den ersten Datensatz in eine Variable vom Typ Long speichert, den nächsten in eine Variable vom Typ Byte usw. Es werden jeweils so viele Bytes gelesen bzw. geschrieben wie die Variable Groß ist. Das muss man steht's im Hinterkopf behalten, denn es kann schnell passieren das man an eine falsche Stelle der Datei versucht mit einem falschem Variabelentyp Daten lesen bzw. speichern. Es kommt dabei zur keine Fehlermeldung, man bekommt jedoch falsche Werte oder im schlimmsten Fall macht die Datei teilweise unbrauchbar.

Bei den Binärdateien gibt es wie bei den typisierten Dateien kein Trennzeichen zwischen den Datensätzen, die Daten werden hintereinander in die Datei geschrieben.

nach Oben

Wie öffne ich Binärdateien ?

Binärdatei ist eigentlich eine völlig falsche Bezeichnung. Binär heißt eigentlich die kleinste Speichemögöligkeit von Daten (Bit), die nur 0 oder 1 enthalten kann. Man speichert jedoch die Datei in Binärdateien nicht als Nullen oder Einzeln (strenggenommen schon) sondern vielmehr als Bytes. Und ein Byte besteht aus acht Bits. Also sollten die Dateien eigentlich Bytedateien heißen, wir bleiben jedoch bei der offizielle Bezeichnung. Um eine Binärdatei zu öffnen muss man nicht eine höhere Schule besucht haben, es reicht lediglich dieses Toturial aufmerksam zu lesen und den Folgenden Befehl zu benutzen:

Open *dateiname* **For Binary As** *dateinummer*

Die Parameter dateiname und dateinummer sollten bereits bekant sein, so das hier keine entsprechende Beschreibung stattfindet. Wichtig ist jedoch noch das im Binary-Modus der Len-Abschnitt (falls mitangegeben) nicht beachtet.

Beispiel:

```
Dim Fnr
Fnr = FreeFile
Open "C:\test.txt" For Binary As Fnr
...
```

nach Oben

Was hat es hierbei mit dem Satzzeiger an sich ?

Mit dem Seek-Befehl kann man wie bei den typisierten Dateien den Satzzeiger in der Datei bewegen bzw. die Position abfragen. Es besteht jedoch ein gravierender Unterschied den man nicht vergessen sollte. Wird die Position bei den typisierten Dateien in Datensätzen

bestimmt ist es bei den Binären Dateien völlig anderes, hier wird die Position in Bytes bestimmt !!! Das heißt also das der Datensatzzeiger bei jedem zugriff auf die Datei, es sei denn schreibend oder lesend, immer um die Anzahl von Bytes verschoben wird wie die Variable groß ist. Wenn wir z.B. ein Wert aus der Datei lesen dessen Größe 1 Byte ist (eine Bytevariable) wird der Satzzeiger um eins verschoben, lesen wir jetzt einen Longwert (eine Longvariable) wird der Satzzeiger um vier verschoben, da die Longvariable vier Bytes groß ist.

Später werde ich es an einem Beispiel versuchen zur verdeutlichen.

nach Oben 

Wie kann ich Daten aus eine Binärdatei lesen ?

Hierzu benutzt man wie bei den typisierten Dateien den Get-Befehl. Die Syntax ist identisch so das es keine gesonderten Beschreibung nötig ist. Man sollte jedoch immer an den Byte-Satzzeiger denken.
Hier möchte ich nun das angesprochene Beispiel vorstellen, an dem man das Verhalten des Satzzeigers leicht erkennen kann.

Beispiel:

```
Dim Data1 As Byte
Dim Data2 As Long
Dim Fnr
Fnr = FreeFile
Open "C:\test.txt" For Random Len = Len(Data2)
    MsgBox "Satzzeiger steht an der Position:" & Seek(Fnr)
    Get Fnr, , Data2
    MsgBox "Satzzeiger steht an der Position:" & Seek(Fnr)
    Get Fnr, , Data2
    MsgBox "Satzzeiger steht an der Position:" & Seek(Fnr)
Close Fnr
Open "C:\test.txt" For Binary As Fnr
    MsgBox "Satzzeiger steht an der Position:" & Seek(Fnr)
    Get Fnr, , Data2
    MsgBox "Satzzeiger steht an der Position:" & Seek(Fnr)
    Get Fnr, , Data2
    MsgBox "Satzzeiger steht an der Position:" & Seek(Fnr)
    Get Fnr, , Data1
    MsgBox "Satzzeiger steht an der Position:" & Seek(Fnr)
    Get Fnr, , Data1
    MsgBox "Satzzeiger steht an der Position:" & Seek(Fnr)
Close Fnr
```

Zuerst wird eine Datei im Random-Modus (also als typisierte Datei) geöffnet und die Datensatzlänge auf die Länge von Data2 eingestellt. Danach wird die Position des Satzzeigers in eine MsgBox ausgegeben, diese sollte direkt nach dem öffnen der Datei auf eins stehen. Jetzt wird ein Datensatz aus der Datei gelesen und der Datensatzzeiger automatisch auf den nächsten Datensatz gebracht. Dieser wird wieder in der MsgBox angezeigt. Obwohl wir vier Bytes aus der Datei gelesen haben zeigt der Satzzeiger auf den zweiten Datensatz. Das ist bei typisierten Dateien auch richtig so. Beim nächstem lesen aus der Datei wird der Datensatzzeiger wieder um eins erhöht. Wir schließen jetzt die Datei und öffnen sie erneut, diesmal jedoch im Binärem Modus. Die MsgBox zeigt das der Datensatzzeiger nach dem öffnen der Datei wie bei den typisierten Dateien aus eins steht. Jetzt lesen wir ein Long-Wert aus der Datei. Der Datensatzzeiger steht diesmal nicht auf zwei sondern auf fünf. Das ist völlig korrekt, ist doch eine Longvariable vier Bytes groß und 1 + 4 = 5. Jetzt wird noch mal ein Longwert gelesen, der Satzzeiger steht jetzt auf 9. Als nächstes wird ein Bytewert gelesen und der Satzzeiger um ein Byte erhöht was 10 ergibt. Wie es in dem Code weiter geht sollte eigentlich schon jedem bekant sein.

Wie man bereits sehen kann ist es nicht ohne weiteres möglich Daten die nicht von konstanter länger sind mitten aus der Datei zu lesen, es sei denn man merkt sich die Position an der die Daten liegen und liest diese mit der Datensatznummer aus.
Sollte die Position der Daten in der Datei bekant sein ist es empfehlenswert diese mit in dem Get-Befehl einzugeben.

Noch eine Anmerkung am Rande. Sollte man unter VB 3 mit Byte-Werten gearbeitet haben (Nachbildung eines Byte as String * 1) muss man diese Definition ab VB 4 in den neuen Datentyp Byte umwandeln da mittlerweile die

Deklaration String * 1 einem zwei Byte langem Wert entspricht.

nach Oben

Wie kann ich in eine Binärdatei schreiben ?

Wie man wahrscheinlich schon vermutet mit dem gleichem Befehl wie bei den typisierten Dateien. Außer der Besonderheit des Satzzeigers besteht hier überhaupt kein Unterschied. Aus diesem Grund spare ich mir eine genauere Beschreibung und komme direkt zur einem Beispiel. Bei diesem Beispiel handelt es sich um eine leicht abgeänderte Version des Programms wo das Fenster an der letzten Position geöffnet wird an der es vor dem Beenden des Programms stand, mit dem unterschied das hier auch die WindowState-Eigenschaft mitgespeichert und berücksichtigt wird.

Beispiel:

```
Public Sub Form_Load()
If FileExist("WindowPrefs.dat") Then
    Dim Fnr
    Dim Data As Long
    Fnr = FreeFile
    Open "WindowPrefs.dat" For Binary As Fnr
        Get Fnr, , Data
        If Data > 3 Then GoTo DatenFalsch
        Form1.WindowState = Data
        If Form1.WindowState = 0 Then
            Get Fnr, , Data
            Form1.Top = Data
            Get Fnr, , Data
            Form1.Left = Data
            Get Fnr, , Data
            Form1.Width = Data
            Get Fnr, , Data
            Form1.Height = Data
        End If
DatenFalsch:
    Close Fnr
End If
End Sub

'...
' hier geht das Programm weiter
'...
Public Sub Form_Unload(Cancel As Integer)
Dim Fnr
Dim Data As Long
Fnr = FreeFile
Open "WindowPrefs.dat" For Binary As Fnr
    Data = Form1.WindowState
    Put Fnr, , Data
    If Data = 0 Then
        Data = Form1.Top
        Put Fnr, , Data
        Data = Form1.Left
        Put Fnr, , Data
        Data = Form1.Width
        Put Fnr, , Data
        Data = Form1.Height
        Put Fnr, , Data
    End If
Close Fnr
End Sub

' und noch eine Funktion mit der man das Vorhandensein
' eine Datei überprüft
Public Function FileExist(Dateiname As String) As Boolean
    On Error GoTo Fehler
    FileExist = Dir$(Dateiname) <> ""
    Exit Function
Fehler:
    FileExist = False
```

```
    Resume Next
End Function

Private Sub Command1_Click()
    Unload Me
    End
End Sub
```

Wichtigste Änderungen zur der letzten Version:
- Fensterdaten werden in Binärdatei gespeichert
- zusätzlich wird der Fensterstatus mitgespeichert
- wenn das Fenster geschlossen wird und es maximiert ist werden die Fensterdaten nicht in die Datei gespeichert
- beim öffnen werden die Fensterdaten nicht eingestellt wenn das Fenster maximiert angezeigt werden soll
- der definierte Datentyp wird nicht mehr benutzt und wurde entfernt

So jetzt sind wir am Ende des Toturials angelangt, können uns mit den drei Datenarten aus, wissen wie man die lesen und schreiben kann und was man dabei beachten soll. Jetzt möchte ich noch eins loswerden, und zwar, in den Modi Binary, Input und Random können die eine Dateien mit einer anderen Dateinummer öffnen, ohne sie zuvor schließen zu müssen. In den Modi Append und Output muss eine Datei erst geschlossen werden, bevor sie mit einer anderen Dateinummer geöffnet werden kann.

Als Abschluss möchte ich noch auf die Datei xyz hinweisen. Es ist ein Programm mit dem jede Art von Dateien betrachtet und bearbeitet werden kann. Es handelt sich dabei um ein einfaches Hexviwer. Man kann daraus einige Tipps entnehmen die später nützlich sein können.

Und jetzt steht das Toturial zur Diskussion. Fragen, Anregungen, Korrekturen können ins Forum geprostet werden die dann von allen besprochen werden können oder ihr könnt mir auch Mailen.

nach Oben