Günter Born

Dateiformate-Referenz

Galileo Computing

Inhalt

dBASE Datenbankformate

dBASE II

dBASE III/III+

dBASE IV

FoxPro

Datenaustausch über das SDF-Format

Der Aufbau einer CSV-Datei

1 Dateiformate in dBASE II

dBASE II war das erste Produkt, das Datenbankfunktionen für einen breiteren Anwenderbereich auf dem PC zur Verfügung stellte. Da noch einige Daten in diesem Format existieren, möchte ich dieses kurz beschreiben.

dBASE II – Format der DBF-Dateien

dBASE II legt Daten in Dateien mit der Erweiterung DBF ab. Der Aufbau dieser Dateien wurde so gewählt, daß sowohl die Daten als auch die notwendigen Definitionen abspeicherbar sind. Jede DBF-Datei besteht deshalb aus drei Teilen: einem Header, der Satzbeschreibung und den eigentlichen Daten (Bild 1.1).

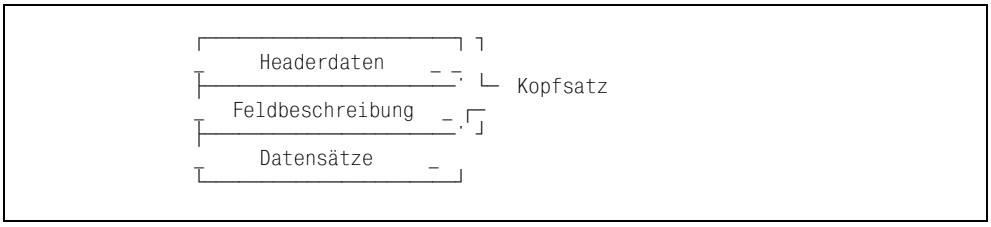


Abbildung 1.1 DUMP-Auszug aus einer dBASE II-NDX-Datei

Der Kopfsatz mit dem Header und der Datensatzbeschreibung umfaßt 520 Byte und ist gemäß Tabelle 1.1 aufgebaut:

Offset	Bytes	Bemerkungen
0	1	Nummer der dBASE-Version 02H dBASE II-DBF-Datei
1	2	Zahl der Datensätze (0 bis FFFF)
3	3	Datum des letzten Schreibzugriffs im Binärformat (TTMMJJ)
6	2	Recordlänge in Byte (bis 1000)
8-519	16*N	16 Bytes pro Feld mit der Beschreibung des Aufbaus (N max. 32)
16*N+1	1	Wert 0DH als Markierung Header Ende

Tabelle 1.1 Format eines DBF-Headers in dBASE II

Der eigentliche *Header* umfaßt die Bytes 0 bis 7. Das erste Byte enthält als Signatur für die jeweilige Dateiversion in den DBF-Dateien immer den Wert 02H. Spätere dBASE-Versionen besitzen andere Werte zur Kennung. Das folgende Wort enthält die Zahl der Datensätze innerhalb der DBF-Datei. Hierin sind auch Datensätze eingeschlossen, die bereits zur Löschung markiert sind, aber noch nicht mit PACK entfernt wurden. (Dieser Sachverhalt wird später noch diskutiert.) Insgesamt lassen sich unter dBASE II bis zu 65535 Sätze abspeichern.

In den Bytes 3 bis 5 speichert dBASE II das Datum des letzten Schreibzugriffs ab. Ein Byte dient dabei jeweils zur Darstellung des Tages, Monats oder Jahres. Ein Beispiel: Die Hexwerte 0F 07 59 für diese Bytes ergeben das Datum 15-07-1989.

Die Länge eines Datensatzes wird in den Bytes 6 und 7 vermerkt. dBASE II erlaubt eine maximale Datensatzlänge von 1000 Byte, wobei ein Satz in maximal 32 Felder aufgeteilt werden kann. In der Regel wird diese Limitierung von 32 Feldern vor der Satzlänge mit 1000 Byte erreicht.

An den Header schließt sich die Beschreibung der Datenfelder an. Jedem der maximal 32 Felder ist ein 16 Byte langer Eintrag zugeordnet, der den Typ, die Länge, den Namen und andere Daten des Feldes enthält. Dabei gilt die in Tabelle 1.2 gezeigte Kodierung:

Offset	Bytes	Bemerkungen
0	11	Name des Feldes als ASCIIZ-String
11	1	Feldtyp in ASCII
12	1	Feldlänge in Byte als Binärzahl (0 bis FFH)
13	2	Datenadresse des Feldes im Speicher
15	1	Zahl der Nachkommastellen in Byte

Tabelle 1.2 Format der DBF-Feldbeschreibung in dBASE II

Die ersten 11 Byte sind für den Feldnamen vorgesehen. Dieser wird als ASCIIZ-String (*ASCII-Zero-String*) abgespeichert. Falls der Name kürzer als 11 Zeichen ist, sind die restlichen Bytes mit dem Wert 00H abzuschließen. Bei einem nichtdefinierten Namen sind alle Bytes mit dem Wert 00H belegt.

Ab Byte 11 ist in der Felddefinition der *Feldtyp* gespeichert. Dafür wird der jeweilige ASCII-Buchstabe C, N oder L eingesetzt. Innerhalb der Felder dürfen die in Tabelle 1.3 gezeigten Werte eingetragen werden:

Zeichen	Feldtyp	erlaubte Zeichen
C	Character	ASCII-Zeichen
N	Numerisch	- . 0...9
L	Logical	JjNnTtFf 20H

Tabelle 1.3 Kodierung der Feldtypen in dBASE IIKodierung der Feldtypen in dBASE II

Ab Byte 12 findet sich die Zahl der durch das Feld belegten Bytes. Bei Strings entspricht dies der maximal möglichen Textlänge. Logical-Felder besitzen immer die Länge 1. Bei Dezimal- oder Ganzzahlen gibt das Feld die Zahl der Stellen an. Die Anzahl der Nachkommastellen ist ab Byte 15 verzeichnet, wobei der Dezimalpunkt mit abgespeichert wird. (Die Rechengenauigkeit bei Dezimalzahlen ist in dBASE II allerdings auf 10 Stellen beschränkt.)

Die *Datenadresse ab Byte 13* wird intern durch dBASE II benutzt und ist für externe Programme nicht weiter relevant.

Die *Felddefinition* darf maximal 32 Felder umfassen, womit der Bereich von Byte 8 bis Byte 519 belegt ist. Bei 32 definierten Feldern steht deshalb in Byte 520 das Zeichen 0DH (CR, *Carriage Return*). Es signalisiert den Abschluß der Felddefinition. Falls nun weniger als 32 Felder definiert sind, findet sich hinter der letzten Felddefinition das Zeichen 0DH als Endemarkierung. Die restlichen Bytes bis zum Eintrag 520 werden in diesem Fall mit Nullbytes (00H) aufgefüllt.

Für die eigentlichen *Daten* existiert ein eigenes Speicherverfahren – sie werden satzweise hinter dem Kopfsatz abgelegt. Dabei gilt, wie in Bild 1.2 dargestellt, für jeden Satz der gleiche Aufbau:

Kodierung der Feldtypen in dBASE II

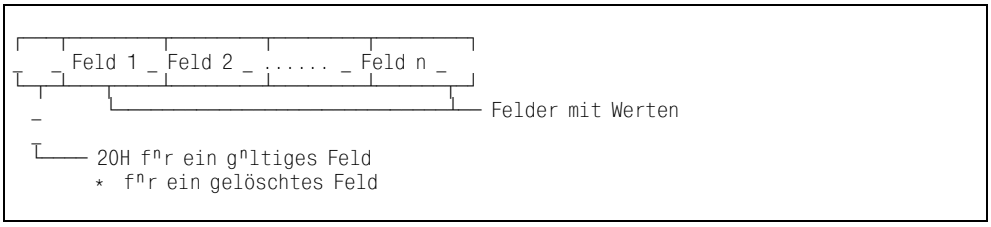


Abbildung 1.2 Aufbau eines Datensatzes in dBASE II

Das erste Byte eines Satzes spezifiziert, ob dieser gültig oder als gelöscht markiert ist. Alle aktuell gültigen Sätze enthalten im ersten Byte den Wert 20H (*Blank*). Dieser Wert steht bereits standardmäßig nach einem Befehl des Typs *Append Blank* im ersten Byte, da bei der Ausführung der Anweisung lediglich ein Satz mit Leerzeichen an das Dateiende angefügt wird. Sobald ein Satz durch den Benutzer gelöscht wird, setzt dBASE II in das erste Byte das Zeichen »*« ein. Damit wird dieser Satz bei einer nachfolgenden PACK-Operation aus der Datenbank entfernt. Bei einem *Undelete* überschreibt dBASE den Eintrag einfach mit einem Blank. Bild 1.3 zeigt die Struktur einer DBF-Datei, die in Bild 1.4 als Speicherdump dargestellt wird.

Kodierung der Feldtypen in dBASE II

Name	Typ	Länge	Dezimalstellen
Feld1	C	020	
Feld2	N	010	
Feld3	N	005	002
Feld4	L	001	

Abbildung 1.3 Struktur der DBF-Datei TEST.DBF

Nachdem die Datenstruktur aus Bild 1.3 als DBF-Datei vereinbart ist, ergibt sich der in Bild 1.4 dargestellte Datenausschnitt.

Interessant ist dabei die Abbildung der Datenbank auf das DOS-Dateisystem. dBASE II erzeugt zuerst den Kopfsatz und trägt hier die erforderlichen Daten ein. Dann beginnt das Programm mit der Ablage der Nutzdaten. Jedes APPEND BLANK hängt einen Satz mit *n* Leerzeichen an die Datei an. Die Zahl *n* entspricht dabei der aus der Felddefinition berechneten Satzlänge. Anschließend werden die Leerzeichen durch die jeweiligen Daten der Felder überschrieben. Zwischen den Felddaten gibt es keine Trennzeichen, da ja die Feldgrenzen exakt in der Definition beschrieben sind. Lediglich das erste Byte wird durch dBASE II verwaltet. Hier findet sich der Wert 20H (*Blank*) für gültige Sätze, während das Zeichen »*« alle zum Löschen freigegebenen Einträge markiert. Allerdings befinden sich alle markierten Sätze nach wie vor in der Datenbank, was sich auch im Headereintrag (Zahl der Records) widerspiegelt. Erst nach einer PACK-Operation werden die mit »*« markierten Sätze entfernt. Hierzu durchsucht dBASE einfach alle Sätze und verschiebt die gültigen Einträge in Richtung des Headers auf die als gelöscht markierten Plätze. Das Ende des gültigen Datenbereichs wird immer durch das Byte 1AH markiert. Dabei tritt der merkwürdige Effekt auf, daß die Größe einer DBF-Datei durch die PACK-Operation nicht verändert wird, obwohl – laut Benutzerhandbuch – die Sätze entfernt werden (siehe Bild 1.4).

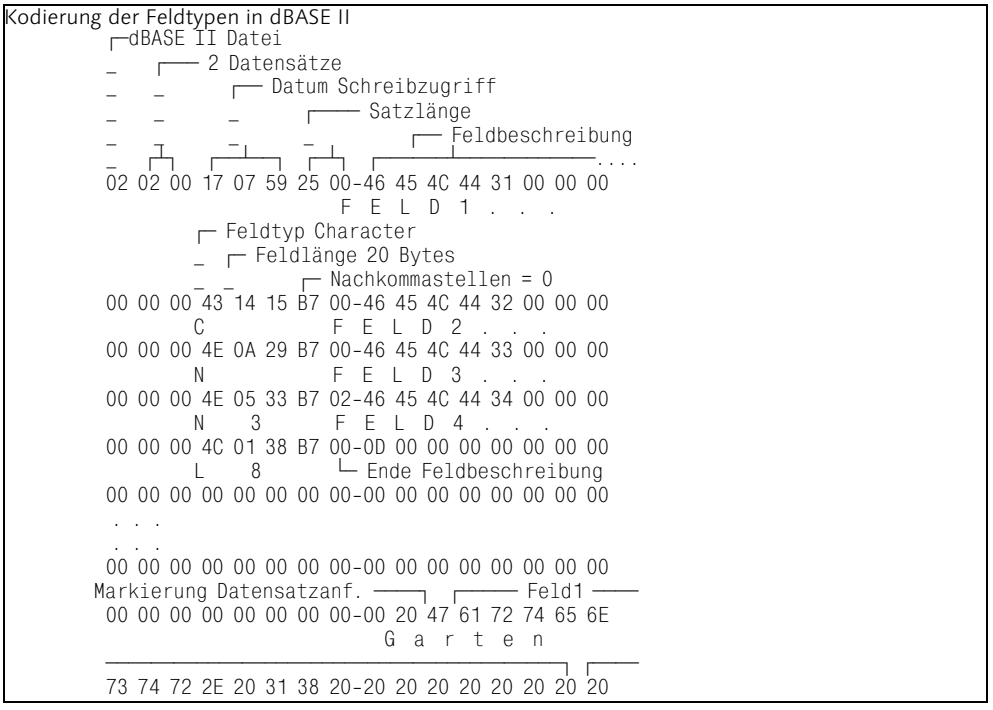


Abbildung 1.4 Dump der Datenbank TEST.DBF

```
 s  t  r  .     1  8                  ┌─ Feld 4
─────  Feld 2 ──────┐   ┌─  Feld 3 ─┐  _  ┌─ 2. Satz
20 20 31 32 33 34 35 36-31 32 2E 30 30 74 20 53
      1  2  3  4  5  6  1  2  .  0  0  t     S
61 74 7A 31 20 20 20 20-20 20 20 20 20 20 20 20
 a  t  z  1
20 20 20 20 20 20 20 20-20 33 34 35 32 20 31 2E
                         3  4  5  2     1  .
30 30 74 1A 57 69 6C 6C-69 20 20 20 20 20 20 20
 0  0  t  └─── Ende Markierung
```

Abbildung 1.4 Dump der Datenbank TEST.DBF

Die Erklärung: dBASE II überschreibt lediglich *gelöschte* Sätze durch gültige Sätze, die alten Sätze bleiben am Dateiende erhalten. Sie sind durch dBASE II nicht mehr adressierbar, da das Zeichen 1AH den letzten gültigen Satz abschließt. Mit geeigneten Hilfswerkzeugen lassen sich die gelöschten Daten noch anzeigen und gegebenenfalls rekonstruieren. Erst wenn die Datenbank per dBASE-COPY-Befehl in eine zweite Datenbank umkopiert wurde, reduziert sich die Dateigröße auf den korrekten Wert. Dieser Aspekt ist sicherlich auch im Hinblick auf den Datenschutz nicht unwichtig – Daten lassen sich also nur durch Anwendung der Befehle PACK und COPY löschen.

Indexdatei-Struktur in dBASE II

Die Datenbank benutzt eigene Indexdateien – die sogenannten NDX-*Dateien* – zum Zugriff auf die Daten über Schlüssel. In dBASE II unterstützen diese sowohl einen indexsequentiellen Zugriff als auch die sequentielle Suche. Bild 1.5 skizziert den grundlegenden Aufbau dieser Dateien:

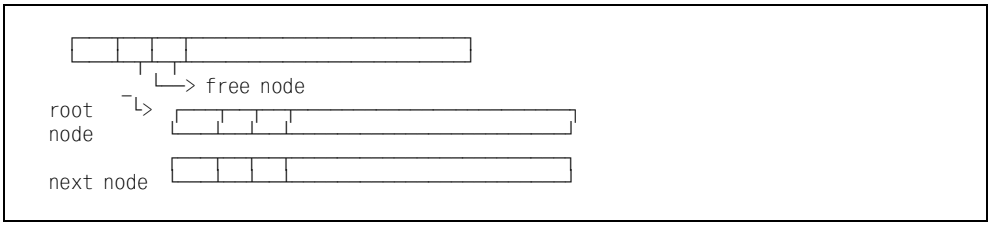


Abbildung 1.5 Aufbau der NDX-Knoten in dBASE II

Die Datei besteht aus einem (Anker-)*Knoten*, der die Zeiger zu den folgenden Knoten mit den Schlüsseldaten enthält. An diese Knoten schließen sich die *Datenknoten* an, in denen die jeweiligen Zeiger auf die Datenrecords in der DBF-Datei gespeichert sind. Die NDX-Dateien besitzen eine feste, 512 Byte lange Recordstruktur, wobei der erste Record als Ankerknoten benutzt wird. In Tabelle 1.4 findet sich dessen Struktur:

Offset	Bytes	Bemerkungen
0	2	reserviert
2	2	Zeiger auf den root node
4	2	Zeiger auf den nächsten freien Knoten
6	1	Länge eines Schlüsseleintrages in Byte + 2 (Key_Length)
7	1	Größe eines Schlüsseleintrages = 2 + 2 + Zahl der Bytes im Schlüssel
8	1	Zahl der Schlüssel pro Knoten
9	1	Numeric Key Flag = 00H bei Characterschlüsseln, sonst numerischer Index
10-109	100	Ausdruck des Schlüssels als ASCIIZ-String mit maximal 100 Byte
110-511		unbenutzt

Tabelle 1.4 Format des Kopfsatzes einer NDX-Datei in dBase II

Der Zeiger ab Offset 2 gibt an, welcher der Knoten innerhalb der Datei als Wurzel benutzt wird. An Hand weiterer Zeiger läßt sich die Datei dann sukzessive bearbeiten. Um bei einem neu anzufügenden Satz schnell den nächsten freien Eintrag zu finden, benutzt dBASE weitere Zeiger. So steht beispielsweise ab Offset 4 die Adresse des nächsten verfügbaren (freien) Records; zusätzliche Zeiger sind innerhalb der einzelnen Schlüsselsätze angelegt.

Die Bytes 6 und 7 geben die Größe eines Schlüssels an, wobei die Bedeutung dieses Parameters allerdings nicht ganz klar ist. Die Sätze mit den eigentlichen Schlüsseln sind immer mit einer festen Recordlänge von 512 Byte aufgebaut; pro Knoten lassen sich *n* Schlüssel speichern. Die *maximale Schlüsselzahl pro Knoten* findet sich ab Offset 08H.

In Byte 9 markiert dBASE den *Schlüsseltyp*. Bei einem numerischen Schlüssel ist der Wert des Bytes ungleich 00H. Der Eintrag 00H definiert dementsprechend einen Schlüssel über ein alphanumerisches Feld.

Den Abschluß des Ankerknotens bildet der ASCIIZ-String mit dem *Schlüsselausdruck*, dessen maximale Länge 100 Byte sein darf. Bei kürzeren Texten besetzt dBASE II die restlichen Bytes mit dem Wert 00H. Der Bereich von Byte 110 bis 511 des Kopfsatzes bleibt bei dBASE II-NDX-Dateien unbenutzt.

Für die Knoten mit den Schlüsseln gilt dann folgende, in Tabelle 1.5 dargestellte Kodierung:

Offset	Bytes	Bemerkungen
0	1	Zahl der Schlüssel in diesem Knoten
1-511	510	Einträge mit den Schlüsselrecords

Tabelle 1.5 Format eines Schlüsselsatzes einer dBASE II-NDX-Datei

Das erste Byte eines Knotens enthält die aktuell eingetragene Anzahl an Schlüsselsätzen. Damit läßt sich in jedem Knoten eine unterschiedliche Zahl an Schlüsseln speichern. Die maximale Zahl wird allerdings durch den Eintrag im Ankerknoten bestimmt. Der restliche Bereich ist mit *n* Records des Schlüssels aufgefüllt – die Struktur dieser Records sieht wiederum so aus:

Bytes	Bemerkungen
0-1	Zeiger auf den Folgeschlüssel
2-3	Satznummer in der DBF-Datei
4-*n*	Schlüsselausdruck als ASCII-Text

Tabelle 1.6 Format eines Schlüsselsatzes einer NDX-Datei in dBASE IIFormat eines Schlüsselsatzes einer NDX-Datei in dBASE II

Im ersten Wort steht ein Zeiger auf den folgenden Schlüsselsatz. Danach folgt der Zeiger, der den zugehörigen Datensatz in der DBF-Datei angibt. Der Rest der Struktur wird durch den jeweiligen Schlüsselausdruck in ASCII-Zeichen belegt.

```
free node    ──┐  key  ┌ key size
root node ┐     length _  ┌── keys per node
          ┌──┐ ┌──┐  _  _  _  ┌ character key
  00 00 01 00 02 00 16 18-15 00 66 65 6C 64 31 00
             Schl^nssel ■ïïïïïï  f  e  l  d  1
    ...
    ...
   ┌── 4 Schl^nssel im Knoten
   _   ┌── Folgesatz ist leer
   _ ┌──┐ ┌──┬── dBASE DBF-Satz Nr.
  04 00 00 01 00 47 61 72-74 65 6E 73 74 72 2E 20
  Schl^nssel ■ïïïïï  G  a  r  t  e  n  s  t  r  .
  31 38 20 20 20 20 20 20-20 00 00 04 00 53 61 74    2. Satz
   1  8                              S  a  t
  7A 31 20 20 20 20 20 20-20 20 20 20 20 20 20 20
   z  1
  20 00 00 03 00 57 69 6C-6C 69 20 20 20 20 20 20    3. Satz
               W  i  l  l  i
  20 20 20 20 20 20 20 20-20 00 00 02 00 64 61 73    4. Satz
                                    d  a  s
  20 64 61 20 20 20 20 20-20 20 20 20 20 20 20 20
     d  a
  20 00 00 9A 99 00 99 1D-9A 2B 00 67 1D 8D 66 F8
```

Abbildung 1.6 DUMP-Auszug aus einer dBASE II-NDX-Datei

Weitere Informationen sind den entsprechenden NDX-Dateien mit einem DUMP-Programm (z. B. DEBUG) zu entnehmen.

MEM-Dateiformat in dBASE II

dBASE II erlaubt es, den Inhalt der aktuell definierten Variablen in einer speziellen Datei, der MEM-*Datei*, zu sichern. Anschließend lassen sich neue Variablen definieren oder die alten Werte überschreiben. Derart überschriebene »Urwerte« lassen sich bei Bedarf allerdings aus der MEM-Datei restaurieren. Intern besitzen MEM-Dateien folgende Struktur:

Bytes	Bemerkungen
0-10	Variablenname als ASCIIZ-String
11	Variablentyp C3 Zeichenvariable CE numerische Variable CC logische Variable
12	Bytes des abgespeicherten Wertes
13-14	unbekannt
15	'E' markiert den Anfang der Definition
16	Zahl der gültigen Vorkommastellen
17-18	Nullbytes
19-*n*	Variablenwert

Tabelle 1.7 Format einer MEM-Datei in dBASE II

Bei Zeichenvariablen wird der Inhalt als ASCIIZ-String abgespeichert. Falls der Text kürzer als das reservierte Feld ist, werden die führenden Stellen mit Nullbytes aufgefüllt. Bei logischen Variablen reserviert dBASE II 17 Byte für den Wert, belegt allerdings nur das letzte Byte mit dem Wert 00 (*False*) oder 01 (*True*). Bei numerischen Werten erfolgt die Kodierung in einer dBASE II-internen Notation. Das Ende des gültigen Speicherbereiches (EOF) wird durch den Code 1AH markiert.

Die folgenden Informationen wurden durch Reverse Engineering gewonnen – es kann also durchaus sein, daß verschiedene Bytes neben der aufgeführten mit einer zusätzlichen Bedeutung belegt wurden.

2 Dateiformate in dBASE III

Als Nachfolger des Produkts dBASE II entwickelte Ashton Tate die Versionen dBASE III und dBASE III+. Die Dateiformate sind intern weitgehend identisch, so daß hier nur der Aufbau der Dateien aus dBASE III+ beschrieben wird.

DBF-Dateiformat in dBASE III und dBASE III+

Der Aufbau dieser Dateien lehnt sich an die Struktur von dBASE II an, wenn auch die Leistungen der neuen Version erheblich gesteigert wurden. Die folgende Tabelle gibt die Leistungsunterschiede zwischen den beiden Versionen an:

Parameter	dBASE II	dBASE III
Zahl der Datensätze	65535	1 Milliarde
Satzlänge in Bytes	1000	4000
Felder im Satz	32	128
Zeichenfeldlänge	256	256
Länge logisches Feld	1	1
numerisches Feld (Stellen)	10	15
Datumsfeld	-	8
Memofeld	-	10

Tabelle 2.1 Unterschiede dBASE II und dBASE III (+)

Jede DBF-Datei in dBASE III besteht ähnlich wie bei dBASE II (siehe Bild 1.1) wieder aus Header, Satzbeschreibung und den eigentlichen Daten.

Die Belegung des Kopfsatzes mit Header und Datensatzbeschreibung hängt von der Programmversion ab. Seine Struktur ist in Tabelle 2.2 dargestellt:

Offset	Bytes	Bemerkungen
0	1	Nummer der dBASE-Version 02H dBASE II-DBF-Datei 03H dBASE III-DBF-Datei 83H dBASE III-DBF-Datei mit Memofeld
1	3	Datum des letzten Schreibzugriffs im Binärformat (JJMMTT)
4	4	Zahl der Datensätze in der Datei
8	2	Headerlänge in Byte
10	2	Datensatzlänge in Byte
12	20	reserviert
32	32 * N	32 Byte pro Feld mit der Beschreibung des Aufbaus

Tabelle 2.2 Format eines DBF-Headers in dBASE III

Offset	Bytes	Bemerkungen
32*N+1	1	Wert 0DH als Markierung *Header Ende*

Tabelle 2.2 Format eines DBF-Headers in dBASE III

Die Informationen sind ähnlich wie in dBASE II wieder gemischt im ASCII- und Binärformat gespeichert.

Das erste Byte dient zur Identifizierung der dBASE-Version. Bei dBASE II wurde hier der Wert 02 abgelegt. Ab dBASE III findet sich im unteren Nibble (Bit 0 ... 3) der Wert 3H. Das oberste Bit (7) zeigt, ob in der Datei Memofelder vorhanden sind. Falls ja, ist der DBF-Datei eine DBT-Datei mit den Memotexten zugeordnet, und das Byte enthält demnach den Code 83H. In allen anderen Fällen befindet sich der Wert 03H im ersten Byte. Findet dBASE einen anderen Wert, lehnt es einen Zugriff ab, da es sich dann nicht um eine DBF-Datei handeln kann.

Das nächste Feld umfaßt drei Byte mit dem im Binärformat kodierten Datum des letzten Schreibzugriffes. Das verwendete Format ist JJMMTT – die Jahreszahl (0...99) ist also jeweils zuerst notiert.

Das folgende Feld umfaßt 4 Byte, in denen die Zahl der Datensätze in der DBF-Datei geführt wird. Die Bytes werden als vorzeichenlose 32-Bit-Zahl interpretiert, wobei die üblichen Intel-Konventionen zur Speicherbelegung (niederwertiges Byte der Zahl auf der untersten Adresse) gelten. Der Wert umfaßt alle Sätze, also auch solche, die bereits zum Löschen markiert sind.

Das darauffolgende Feld umfaßt eine vorzeichenlose 16-Bit-Zahl, in der die Headerlänge in Byte steht. Diese Information ist insofern von Bedeutung, da die DBF-Datei eine variable Anzahl von Felddefinitionen (siehe unten) enthalten kann.

Die Länge eines Datensatzes wird ab Byte 10 als vorzeichenlose 16-Bit-Zahl geführt. Dieser Wert ist immer um ein Byte höher als die rechnerische Summe der einzelnen Feldlängen, was dadurch begründbar ist, daß am Anfang eines Datensatzes immer ein Byte zur Markierung gelöschter Sätze reserviert wird.

Ab Byte 12 folgt ein 20 Byte großer reservierter Bereich, der für die interne Verwendung vorgesehen ist. In der Netzwerkversion sind 13 Byte innerhalb des reservierten Bereichs belegt. Genaue Informationen hierüber sind aber nicht bekannt. Die 20 reservierten Byte sorgen dafür, daß der Header genau 32 Byte umfaßt.

Die Informationen über den Aufbau der Datensätze schließen sich wie in dBASE II an den Header an. Auch hier gibt es wieder einzelne Feldbeschreibungen, wobei ab dBASE III bis zu 128 Felder definierbar sind. Für jedes Feld der Datenbank findet sich ein Satz mit 32 Byte, der das in Tabelle 2.3 gezeigte Format besitzt:

Offset	Bytes	Bemerkungen
0	11	Name des Feldes in ASCII-Zeichen
11	1	Feldtyp in ASCII (C, N, L, D, M)
12	4	Datenadresse des Feldes im Speicher
16	1	Feldlänge in Byte als Binärzahl
17	1	Zahl der Nachkommastellen in Byte
18	2	reserviert für Mehrbenutzerbetrieb
20	1	ID für Arbeitsbereich
21	2	reserviert für Mehrbenutzerbetrieb
23	1	Kennzeichnung für SET FIELDS
24	8	reserviert

Tabelle 2.3 Die DBF-Feldbeschreibung in dBASE III

Die *ersten* 11 Byte der Feldbeschreibung enthalten den *Feldnamen*, der als ASCIIZ-Text abgelegt wird. dBASE lehnt sich hier stark an die Sprache C an, die Zeichenketten ebenfalls mit einem Nullbyte abschließt. Umfaßt der Feldname keine 11 Zeichen, werden die restlichen Byte mit dem Wert 00H aufgefüllt.

Im nächsten Byte steht das ASCII-Zeichen für den *Feldtyp*. In Tabelle 2.4 findet sich die Kodierung der gültigen Feldtypen ab dBASE III. Gegenüber dBASE II gibt es nun auch Datums- und Memofelder.

Zeichen	Feldtyp	erlaubte Zeichen
C	Character	ASCII-Zeichen
N	Numerisch	- 0...9
L	Logical	JjNnTtFf ?
D	Datum	JJJJMMTT
M	Memofeld	DBT Blocknr

Tabelle 2.4 Kodierung der Feldtypen in dBASE IIIKodierung der Feldtypen in dBASE III

An den Feldtyp schließt sich ein 4-Byte-Vektor an, der von dBASE intern zur Speicherung der Feldadresse benutzt wird. Für den Benutzer hat dieser Wert keine weitere Bedeutung.

Die *Feldlänge* findet sich ab Offset 16 und ist in einem Byte als Binärcode abgelegt. Damit kann ein Feld maximal 256 Zeichen umfassen. Diese Länge wird aber nur bei Characterfeldern ausgenutzt; bei numerischen Feldern gibt der Wert die Zahl der Dezimalstellen einschließlich des Dezimalpunktes an (die Verarbeitungsgenauigkeit für Zahlen ist in dBASE III jedoch auf 15 Stellen beschränkt). Bei Memofeldern beträgt die Feldlänge immer 10 Byte, da dort die Blocknummer für den Satz in der zugehörigen DBT-Datei gespeichert

wird – weitere Erläuterungen hierzu finden Sie bei der Beschreibung des DBT-Formates. Logische Felder besitzen die Länge 1, während bei Datumsfeldern immer 8 Byte reserviert werden.

Bei numerischen Feldern spezifiziert das Folgebyte die *Zahl der Nachkommastellen*. Bei allen anderen Feldtypen besitzt der Eintrag den Wert 00H. Wichtig ist, daß die Zahl der Nachkommastellen immer kleiner als die Feldlänge ist.

Die restlichen 14 Byte sind für interne Zwecke reserviert. Sie müssen lediglich beim Zugriff auf die Feldbeschreibung überlesen werden. Auch das SET FIELDS-Byte ist nicht weiter relevant, da DBASE III diesen Eintrag offensichtlich nur im Speicher benutzt.

Jedes definierte Feld der Datenstruktur besitzt einen eigenen 32-Byte-Satz im Kopf der DBF-Datei. Das Ende der Felddefinition wird durch das Zeichen 0DH markiert.

Die eigentlichen *Datensätze* werden in dBASE III und dBASE III+ (wie bei dBASE II) an den Definitionsteil angehängt. Die Recordlänge richtet sich nach der Länge der jeweiligen Felder und wird im Dateiheader geführt. Der Wert ist immer um 1 Byte größer als die Summe der Feldlängen, da vor jedem Record ein Byte für die *Delete*-Markierung reserviert ist. Ein Datensatz wird im reinen ASCII-Format ohne Trennzeichen gespeichert, was den Datenimport und -export über die Option SDF natürlich recht einfach macht.

Daten in Zeichenfeldern werden durch eine Sequenz von ASCII-Zeichen dargestellt. Ist der Text kürzer als die in der Felddefinition spezifizierte Länge, werden die restlichen Byte mit Leerzeichen (Code = 20H) aufgefüllt.

Numerische Werte werden ebenfalls als ASCII-String gespeichert. Die Zahl der Stellen wird in der Felddefinition spezifiziert. Bei Dezimalzahlen wird die Zahl der Nachkommastellen ebenfalls in der Felddefinition festgehalten. Es ist zu beachten, daß der Dezimalpunkt auch im Feld auftritt und somit eine Ziffer »verlorengeht«. Ist die Zahl kleiner als die vorgesehene Feldlänge, werden die führenden Stellen mit Leerzeichen (20H) aufgefüllt (z.B. » 999.99«).

Logische Werte werden in einem Byte mit dem Zeichen »F« oder »T« dargestellt.

Das Datumsfeld enthält das Datum als ASCII-String mit 8 Zeichen Länge im Format »JJJJMMDD«.

Ein Memofeld enthält eine 10-Byte-Zahl, die den Satz in der DBT-Datei spezifiziert. In der DBT-Datei findet sich der zugehörige Text. Führende Stellen werden gegebenenfalls mit Leerzeichen aufgefüllt. Enthält das Feld 10 Leerzeichen, dann existiert kein Text-Record im zugehörigen DBT-File.

Weiterhin lassen sich unabhängig von ihrem Typ alle Felder als ASCII-Text bearbeiten.

Sobald über den Befehl APPEND BLANK ein neuer Satz an die Datei angehängt wird, füllt dBASE diesen Satz mit Blanks auf. Das erste Byte im Satz dient zur Markierung gelöschter Daten, und da auch bei neuen Sätzen ein Blank eingetragen ist, können sie nicht gelöscht werden. Erst wenn das erste Byte das Zeichen »*« enthält, wird der Satz beim nächsten PACK-Befehl aus der DBF-Datei entfernt. Dies führt zu sehr schnellen DELETE-Operationen und ermöglicht sogar ein UNDELETE ohne großen Aufwand. Derart behandelte Sätze stehen jedoch nach wie vor in der Datenbank, so daß dort durchaus mehrere hundert Sätze enthalten sein können, von denen keiner gültig ist. Zugriffe ohne Index werden dann natürlich recht langsam, da alle Sätze (gelöscht oder ungelöscht) zunächst gelesen werden müssen. Um diesen Nachteil zu korrigieren, sollte man gelöschte Sätze möglichst häufig über PACK aus der DBF-Datei entfernen. Records, die schon zum Löschen markiert sind, werden dann durch nachfolgende gültige Sätze überschrieben, und der Eintrag im Header wird auf die Zahl der ungelöschten Records reduziert.

Der *Abschluß des gültigen Datenbereiches* wird durch das Zeichen 1AH markiert. Wichtig ist die Erkenntnis, das diese EOF-Marke nicht durch DOS, sondern durch dBASE verwaltet wird. Da die PACK-Operation die Größe der DBF-Datei nicht verändert, finden sich hinter der EOF-Marke durchaus noch gelöschte Sätze. Diese lassen sich mit entsprechenden Hilfsmitteln restaurieren, was beim Thema Datenschutz beachtet werden sollte. Erst der dBASE-Befehl COPY FILE TO überträgt nur die gültigen Sätze in die andere Datei und verkleinert damit die Dateilänge. Bild 2.1 gibt den Ausschnitt einer DBF-Datei in dBASE III als Hexdump wieder.

```
┌─dBASE III Datei mit Memofeld
_      ┌── Datum letzter Schreibzugriff
_      _        ┌── Zahl der Records
_      _        _       ┌──── Länge des Headers
_      _        _       _    ┌── Länge Datensatz
_   ┌──┴──┐ ┌──┴──┐ ┌──┴──┐ ┌┴┐ ┌┴┐ ┌ reserviert
83 58 0B 1E 0A 01 00 00-C1 00 2D 00 00 00 00 00
00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
     Feldname                  ┌ Feldtyp
┌──────────┴──────────────────────┐ _ ┌── Adresse
46 45 4C 44 31 00 00 00-00 00 00 43 11 00 A0 47
 F  E  L  D  1  .  .  .
 ┌ Feldlänge in Byte
 _  ┌ Nachkommastellen = 0
14 00 00 00 01 00 00 00-00 00 00 00 00 00 00 00
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
46 45 4C 44 32 00 00 00-00 00 00 4E 25 00 A0 47
 F  E  L  D  2  .  .  .  .  .  .  N  .  .  .  G
05 02 00 00 01 00 00 00-00 00 00 00 00 00 00 00
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
46 45 4C 44 33 00 00 00-00 00 00 44 2A 00 A0 47
 F  E  L  D  3                 D  .  .  .  G
08 00 00 00 01 00 00 00-00 00 00 00 00 00 00 00
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
46 45 4C 44 34 00 00 00-00 00 00 4C 32 00 A0 47
```

Abbildung 2.1 Hexdump der dBASE III-Datenbank TEST.DBF

```
 F  E  L  D  4  .  .  .  .  .  .  L  2  .  .  G
01 00 00 00 01 00 00 00-00 00 00 00 00 00 00 00
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
46 45 4C 44 35 00 00 00-00 00 00 4D 33 00 A0 47
 F  E  L  D  5  .  .  .  .  .  .  M  3  .  .  G
0A 00 00 00 01 00 00 00-00 00 00 00 00 00 00 00
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
0D 20 20 20 20 20 20 20-20 20 20 31 20 20 20 20
    └─ Markierung Datensatzanf. 1
 └─ Ende Header
20 20 20 20 20 20 20 31-2E 30 30 31 39 38 38 30
                     1  .  0  0  1  9  8  8  0
33 31 32 54 20 20 20 20-20 20 20 20 20 20 20 20
 3  1  2  T
20 20 20 20 20 20 20 20-32 20 20 20 20 20 20 20
                        2
20 20 20 20 32 2E 30 30-31 39 33 33 30 33 31 32
           2  .  0  0  1  9  3  3  0  3  1  2
46 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
 F
20 20 20 20 20 33 20 20-20 20 20 20 20 20 20 20
 .....
```

Abbildung 2.1 Hexdump der dBASE III-Datenbank TEST.DBF

Indexfilestruktur (NDX) in dBASE III

dBASE verwendet eigene Indexdateien (NDX) zum Zugriff auf die Daten über Schlüssel. Die dBASE-NDX-Files nutzen eine modifizierte B-Baum-Struktur zur Verwaltung der Indexdaten. (Es gibt allerdings einige Unterschiede zur Clipper-NTX-Struktur).

Der Aufbau des NDX-Headers

Ein NDX-File wird in Seiten zu je 512 Byte aufgeteilt. Die erste Seite dient als Header, daran schließen sich die Indexseiten an. Die Struktur des Headers ist Tabelle 2.5 zu entnehmen.

Offset	Byte	Bemerkungen
00H	4	Start_key_page (root page)
04H	4	Total_pages (max. Seitenzahl)
08H	4	reserviert
0CH	2	Index_key_len (Länge Schlüssel)
0EH	2	Max_keys_page
10H	2	NDX_key_type
12H	4	Size_key_record
16H	1	reserviert
17H	1	UNIQUE-Flag

Tabelle 2.5 Format des dBASE-NDX-Headers

Offset	Byte	Bemerkungen
18H	488	Key_name

Tabelle 2.5 Format des dBASE-NDX-Headers

Der 512 Byte lange Header beschreibt den Aufbau der Indexseiten und enthält die Zeiger zum Zugriff auf die Wurzel des B-Baumes.

Das Feld »Start_key_page« (Offset 00H) enthält die Recordnummer (4 Byte) der »root_page«. Dies ist die erste Seite (Wurzel) des B-Baumes. Um den Offset in Byte zu ermitteln, ist der Wert mit der Recordlänge von 512 Byte zu multiplizieren. (Anmerkung: Wegen der Begrenzung des DOS-Filesystems auf 4-Byte-Offset-Zeiger läßt sich als größte Zahl der (theoretische) Wert 7FFFFF (entspricht 8388607 Seiten) nutzen. Diese Grenze wird in der Praxis unter DOS wegen der Begrenzung der Plattenkapazität nie erreicht.)

Beim Aufbau der Indexseiten wird der B-Baum öfters »umsortiert«. Sofern sich die Recordnummer der Startseite (Wurzel des B-Baumes) ändert, muß anschließend die Recordnummer der Startseite im Header nachgetragen werden.

Im 4-Byte-Feld »Total_pages« (Offset 04H) findet sich die Information, wie viele 512-Kbyte-Seiten in der NDX-Datei vorliegen.

Das Feld »Index_key_len« (Offset 0CH) spezifiziert die Länge des Schlüssels in Byte oder Zeichen, der der NDX-Datei zugrunde liegt.

Mit »Max_keys_page« (Offset 0EH) wird festgelegt, wie viele Schlüsseleinträge (keys) eine 512 Byte große Seite maximal aufnehmen kann. Dies hängt im wesentlichen von der Länge des Schlüssels ab. Die wirkliche Zahl der Einträge einer aktuellen Seite wird am Anfang der jeweiligen Seite festgehalten (siehe Aufbau der Indexseiten).

Der Typ des Index wird im Feld »NDX_key_typ« (Offset 10H) angegeben. Möglich sind numerische Schlüssel (Code = 01H) oder alphanumerische Schlüssel (Code = 00H). Die Darstellung der Schlüsseleinträge in den Seiten des Baumes ist abhängig vom Typ des Schlüssels. Numerische Indizes und ein Index über das Datum werden als Fließkommazahl im IEEE-Format (8 Byte) abgelegt. Alphanumerische Indizes werden als ASCII-Strings gespeichert. Ist der String kürzer als die vorgesehene Länge des Schlüssels, werden die fehlenden Stellen (rechts) durch angehängte Leerzeichen aufgefüllt.

Die Länge eines Schlüsselrecords in Byte wird im Feld »Size_key_rec« (Offset 12H) festgehalten. Über dieses Feld läßt sich der Abstand zwischen zwei Schlüsselseiten in der Indexseite bestimmen (siehe auch Aufbau der Indexseite).

Das Byte-Feld »UNIQUE« (Offset 17H) fungiert als Flag. Enthält das Flag den Wert 1, wurde der Index mit der Option »UNIQUE ON« angelegt. Mit dem Wert 0 wurde der Index mit »UNIQUE OFF« erzeugt.

Ab Offset 18H (24 dezimal) beginnt ein 488 Byte langer Bereich, der einmal als Füller dient, um die Seite auf 512 Byte zu verlängern. In diesem Füllbereich wird gleichzeitig der Name des Schlüssels als ASCIIZ-String abgelegt. Dies ist der String, der beim Aufbau der Indexdatei angegeben wird (SET INDEX ON ...).

Der Aufbau der Indexseiten

An diesen Header schließen sich *n* Indexseiten à 512 Byte an. Die Zahl *n* wird dabei im Header ab Offset 04H als 4-Byte-Zahl angegeben. In diesen Indexseiten sind die Schlüssel auf die zugehörigen Datensätze der DBF-Datei gespeichert. Eine Indexseite besitzt dabei den Aufbau gemäß Tabelle 2.6.

Offset	Byte	Bemerkungen
00H	4	Key_record_page
04H	4	Left_page_num_1
08H	4	DBF_rec_num_1
0CH	Len	Key_data_1 (Schlüssel)
..H	4	Left_page_num_2
..H	4	DBF_rec_num_2
..H	Len	Key_data_1 (Schlüssel)
		....
..H	4	Left_page_num_n
..H	4	DBF_rec_num_n
..H	Len	Key_data_n (Schlüssel)

Tabelle 2.6 Aufbau einer dBASE-NDX-IndexseiteAufbau einer dBASE-NDX-Indexseite

Die ersten 4 Bytes einer Indexseite (Feld »Key_records_page«) enthalten einen Wert, der die Zahl der folgenden Schlüsseleinträge innerhalb der Seite angibt. Ist die Seite leer, steht hier der Wert 00 00 00 00.

Daran schließen sich *n* Einträge mit den Schlüsselwerten an. Der Wert von *n* entspricht dabei der Angabe im Feld »Key_records_page«. Jeder Schlüsseleintrag besteht dabei aus einer Datenstruktur mit den Einträgen »Left_page_num«, »DBF_rec_num« und »Key_data«.

Im Feld »Left_page_num« wird die nächste Seite adressiert, die im B-Baum links vom aktuellen Schlüssel liegt (siehe Bild 2.2) Die betreffende Indexseite enthält alle Schlüssel, die in der Sortierreihenfolge kleiner oder gleich dem gesuchten Schlüssel sind. Die Zeiger auf die Seiten werden dabei als Recordnummern gespeichert, d.h. der Offset errechnet sich durch Multiplikation mit der Recordlänge von 512 Byte. (In den Clipper-NTX-Dateien werden dagegen absolute Recordadressen benutzt.)

Das Feld »DBF_rec_num« enthält die Recordnummer der zugehörigen Daten der DBF-Datei. Innerhalb der Knoten des B-Baumes wird dieser 4-Byte-Eintrag auf 0 gesetzt, d.h. in diesem Fall existiert eine weitere Indexseite, wo der eigentliche Schlüssel eingetragen ist. Erst im »Blatt des Baumes« wird auf den DBF-Record verwiesen. Hier enthält das Feld dann die betreffende Recordnummer. Um den Offset (in Byte) zu den Daten zu erhalten, ist die Recordnummer mit der Satzlänge (Offset 0AH) aus dem DBF-Header (siehe Tabelle 2.2) zu multiplizieren.

Im letzten Feld »Key_data« wird der eigentliche Schlüssel gespeichert. Die Länge wird durch die Länge des Indexfeldes (Offset 12H im Header der NDX-Datei) bestimmt. Bei ASCII-Feldern werden beim Index die fehlenden Stellen rechts mit Leerzeichen aufgefüllt. Numerische Indizes und der Datumsindex werden als 8-Byte-IEEE-Fließkommazahl gespeichert.

Dieser Sachverhalt wird nochmals schematisch in Bild 2.2 verdeutlicht.

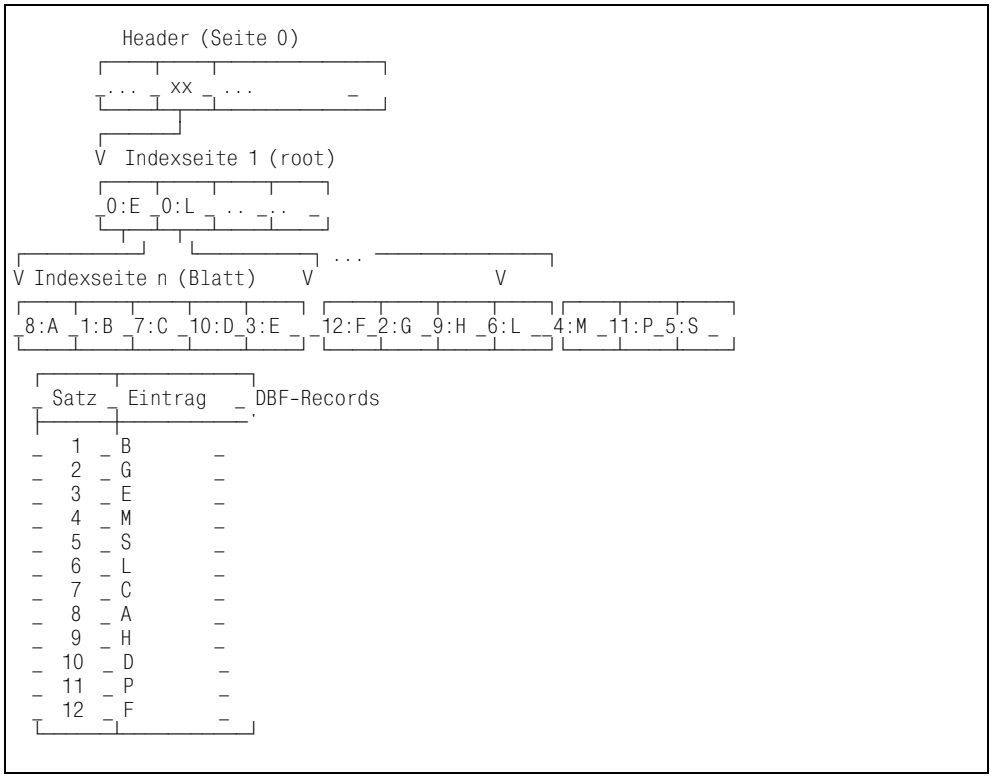
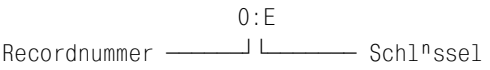


Abbildung 2.2 Schema des NDX-B-Baumes

Die einzelnen Seiten besitzen im Gegensatz zu den NTX-Dateien (Clipper) kein Zeigerfeld mit Verweisen auf die Folgeschlüssel. Vielmehr sind die Einträge in den Indexseiten

als »Verweise« auf andere Indexseiten zu interpretieren. Dies wird schematisch durch die in Bild 2.2 angegebenen Schlüssel dargestellt. Der Eintrag



in einer Seite besteht dann aus einer Zahl, die als Recordnummer in die DBF-Datei zu interpretieren ist. Ist der Wert 0, existiert ein Verweis auf eine weitere Indexseite. Bei der Angabe »3:E« ist der gesuchte Begriff »E« dagegen im dritten DBF-Record gespeichert. Der Text hinter dem Doppelpunkt gibt den eigentlichen Schlüssel an. Aus Aufwandsgründen habe ich mich hier auf einen Buchstaben beschränkt. (Beachten Sie weiterhin, daß der Doppelpunkt lediglich zur Veranschaulichung in obige Abbildung aufgenommen wurde. In der NDX-Datei existiert für jeden Eintrag die in Tabelle 2.6 beschriebene Datenstruktur.)

Der Eintrag »0:E« bedeutet dann, daß eine weitere Indexseite existiert, in der alle Schlüssel gespeichert sind, die in der Sortierreihenfolge kleiner oder gleich dem gesuchten Begriff sind. Bei dieser Indexseite handelt es sich um den linken Ast (vom Knoten aus gesehen) des B-Baumes. Diese Indexseite kann ihrerseits wieder auf weitere Seiten verweisen.

Ist der gesuchte Schlüssel dagegen größer als der aktuelle Wert in »Key_data_n«, muß der nächste Eintrag »Key_data_n+1« der aktuellen Indexseite nach dem gerade beschriebenen Verfahren untersucht werden. Ist der letzte Eintrag der Seite erreicht, ohne daß der Schlüssel gefunden oder zu einer Folgeseite verzweigt wurde, existiert kein Eintrag in der Indexdatei.

Der Aufbau der Indexseiten führt dazu, daß die Seiten jeweils sequentiell nach dem Schlüssel durchsucht werden, was nicht gerade effizient ist. Weiterhin kann dBASE leere Seiten einer NDX-Datei nicht wieder belegen. Die nachfolgend vorgestellte Clipper-NTX-Struktur ist hier wesentlich effizienter.

Indexdatei-Struktur im dBASE III-Clipperformat (NTX)

Für dBASE III-Anwender wird von der Firma Nantucket ein eigener Compiler angeboten, der die Programme in lauffähigen Code umsetzt. Aus Performancegründen definierten die Entwickler neben der dBASE-NDX-Indexstruktur eine weitere (effizientere) Struktur (NTX) für Indexdateien.

Diese NTX-Dateien bestehen aus *n* Seiten mit jeweils 1024-Byte-Länge. Die erste Seite enthält den NTX-Header, während die restlichen Seiten die Schlüssel und die Zeiger des B-Baumes aufnehmen.

Der Aufbau des NTX-Headers

Der Header der NTX-Datei ist ebenfalls 1024 Byte lang und besitzt das Format gemäß Tabelle 2.7.

Offset	Byte	Bemerkungen
00H	2	Clipper-Signatur (sign)
02H	2	Version des Compilers (version)
04H	4	Zeiger auf erste Seite (root node)
08H	4	Leerseite (empty page)
0CH	2	Größe eines Eintrages (item size)
0EH	2	Größe des Schlüssels (key size)
10H	2	Stellenzahl numerischer Key (key dec)
12H	2	Maxim. Einträge pro Knoten (max item)
14H	2	Größe halbe B-Baumseite (half page)
16H	256	Schlüsselausdruck als ASCIIZ-String
272H	1	Unique Flag

Tabelle 2.7 Der NTX-Header

Das 2-Byte-Feld »sign« enthält die Signatur 0003H für gültige Clipper-87-Indexfiles und 0006H für Clipper 5.x. In der Datei ergibt sich wegen der Speicherung der Daten dann die Bytefolge 03 00 bei Clipper 87 bzw. 06 00 bei Clipper 5.x.

In dem Feld »version« (Offset 02H) wird die Versionsnummer des Clipper-Compilers eingetragen. Damit läßt sich feststellen, mit welcher Softwareversion dieser Index erstellt wurde.

Im Feld »root node« (Offset 04H) befindet sich ein Zeiger auf die erste Indexseite. Hierbei handelt es sich um einen 4-Byte-Zeiger, der den Offset (in Byte) vom Fileanfang zum Beginn der ersten Indexseite angibt. Ab dieser Indexseite beginnt die Speicherung der Indexschlüssel.

Mit dem Feld »empty page« (Offset 08H) werden leere Indexseiten verwaltet. Eine Clipper-Indexdatei besteht aus einzelnen Seiten zu je 1024 Byte. Nach der ersten Seite mit den Headerinformationen folgen die Seiten mit den Indexeinträgen. Einige Seiten innerhalb des Indexfiles können aber leer sein. Diese leeren Seiten werden durch eine verkettete Liste markiert. Der 4-Byte-Zeiger »empty page« definiert dabei den Offset (in Byte) vom Beginn der Datei zur ersten leeren Seite. Die ersten 4 Bytes einer leeren Seite enthalten wieder einen Zeiger auf die nächste unbenutzte Seite. Das Ende dieser Liste wird durch den Wert 00 00 00 00 in den ersten 4 Bytes der Seite markiert. Ist im Feld »empty page« der Wert 0 eingetragen, existieren keine freien Seiten mehr. (Anmerkung: Durch diese Technik lassen sich unbelegte Seiten wieder benutzen. dBASE kennt diese Speicher-

verwaltung nicht, so daß hier leere Seiten nicht mehr verwendbar sind. Eine leere Seite tritt auf, falls alle Indexeinträge der Seite aus der Datenbank gelöscht werden).

Das Feld »item size« (Offset 0CH) umfaßt 2 Byte und definiert die Größe eines Eintrages pro Schlüssel in einer Indexseite. Der Wert wird nach der Formel

```
item size = index_key_len + 2*4 Byte
```

bestimmt. Neben dem Speicherbedarf für den eigentlichen Index sind noch zwei Zeiger (Seite, Satznummer) mit je 4 Byte zu addieren. Dieser Wert entspricht der Schrittweite zum Zugriff auf die einzelnen Einträge innerhalb einer Indexseite.

Das Wort »key size« (Offset 0EH) definiert die Größe des eigentlichen Schlüssels, aus dem der Index aufgebaut wird. Umfaßt das Schlüsselfeld zum Beispiel 10 Zeichen (C*10), wird »key size« ebenfalls auf den Wert 10 gesetzt. Der Wert ist 8 Byte kleiner als der Inhalt des Feldes »item size«.

Das Feld »key dec« (Offset 10H) umfaßt 2 Byte und gibt bei numerischen Schlüsseln die Zahl der Dezimalstellen an. Bei alphanumerischen Schlüsseln wird das Feld nicht verwendet.

Mit dem Inhalt des Feldes »max key« (Offset 12H) wird die Indexseite verwaltet. Pro Indexseite lassen sich *n* Indexeinträge (Schlüsselbegriffe + zugehörige Zeiger) abspeichern. Die Zahl der Einträge hängt von der Länge des Index ab, da die Seite 1024 Byte umfaßt. Das 2-Byte-Feld »max key« gibt die maximale Zahl der Indexeinträge pro Indexseite an.

Das 2-Byte-Feld »half page« spezifiziert die Zahl der Schlüsselbegriffe pro Seite, dividiert durch 2. Dieser Wert bestimmt, wie viele Einträge mindestens in einer Seite des B-Baumes abzulegen sind. Diese Angabe ist beim Aufbau des B-Baumes wichtig, da dieser möglichst ausbalanciert werden soll.

Das folgende Feld »key expr« (Offset 16H) enthält den Schlüsselbegriff als ASCIIZ-String. Es ist eine variable Länge mit maximal 256 Zeichen vorgesehen. Der String wird mit dem Zeichen 00H abgeschlossen.

»Unique« ist ein als boolean definiertes Byte-Flag (ab Offset 272H), welches den Status des »Unique Flags« zur Zeit der Indexerzeugung aufnimmt. Der Wert 1 definiert, daß »UNIQUE ON« war, bei 0 war der Wert auf »UNIQUE OFF« gesetzt.

Der Rest des 1024-Byte-Headers ist für Füllbytes reserviert.

Der Aufbau der Indexseiten

An diesen Header schließen sich die Indexseiten an. Jede 1024-Byte-Seite umfaßt eine Anzahl von Einträgen (keys), die auf die Datensätze der DBF-Dateien verweisen. Tabelle 2.8 gibt die Strukturen innerhalb der Indexseite wieder.

Offset	Byte	Bemerkungen
00H	2	Zahl der Einträge (count)
02H	x * 2	Zeigerfeld (ref[maxitem+1]) auf die folgenden *n*-Einträge (Items)
xxH	 4 4 n	*n*-Elemente mit folgenden Feldern: Offset linker Seite (page) Satznummer (record number) Schlüssel (key)

Tabelle 2.8 Aufbau der Struktur der NTX-IndexseiteAufbau der Struktur der NTX-Indexseite

Der erste Eintrag der Indexseite umfaßt 2 Byte. (count+1) spezifiziert die Anzahl der belegten Einträge der Seite (Items). Der Wert muß zwischen den Vorgaben für »half page« und »max item« liegen. Lediglich in der ersten aufgebauten Indexseite (root page) darf der Wert zwischen 1 und »max item« liegen.

Ab Offset 02H beginnt ein 2-Byte-Feld mit Zeigern. Diese Zeiger verweisen zu den Schlüsselrecords (Items) innerhalb der Seite. Diese Schlüsselrecords bestehen gemäß Tabelle 2.8 jeweils aus drei Elementen (page, record number, key), deren Bedeutung weiter unten beschrieben wird. Für das Zeigerfeld werden immer »max item+1« Worte reserviert und (count+1) Worte belegt. Der Wert 0 signalisiert, daß kein zugehöriger Schlüsselrecord in der Seite vorliegt. Andernfalls gibt ein positiver Wert den Offset vom Seitenanfang zum Schlüsselrecord innerhalb der Seite an.

An das Feld mit den Zeigern schließen sich dann die eigentlichen Schlüsselrecords (Items) an. Deren Anzahl wird im ersten Wort (count+1) der Indexseite festgehalten. Jeder Schlüsselrecord (Item) besteht aus drei Elementen (page, record number, key). Das erste Element mit dem Namen »page« enthält einen 4-Byte-Zeiger auf die logisch vorhergehende Indexseite (Offset in Byte vom Dateianfang). Das Feld »record number« enthält gegebenenfalls einen 4-Byte-Zeiger auf den Datensatz der DBF-Datei (Offset in Byte vom Beginn der DBF-Datei bis zum Datensatz). Das Feld »key« besitzt eine variable Länge und dient zur Aufnahme des Schlüssels. Die Länge des Feldes ist im Header der NTX-Datei im Feld »key size« (Offset 0EH) definiert. Das Feld zu Beginn einer Indexseite enthält dann jeweils die Zeiger auf die einzelnen Schlüsselrecords.

Beim Vergleich des Suchbegriffs mit dem Eintrag im Feld »key« können nun drei Situationen auftreten:

- Der Suchbegriff stimmt mit dem eingetragenen Index (key) überein. Dann findet sich im folgenden Feld »record number« die Recordnummer (Offset in Byte) der DBF-Datei.
- Der Suchbegriff kommt logisch vor dem eingetragenen Index (key). In diesem Fall muß die Suche in einem anderen Knoten des Baumes fortgesetzt werden. Der Inhalt des Feldes »page« gibt dann den 4-Byte-Offset in die NTX-Datei an, an der die nächste (logisch vorhergehende) Indexseite folgt. Alle Einträge in den Indexseiten des Teilbau-

mes, auf die der Inhalt von »page« verweist, sind also kleiner als der aktuelle Index im Feld »key«.

- Der gesuchte Schlüssel kommt logisch nach dem eingetragenen Index (key). Der Inhalt von »page« ist hier nicht benutzbar, da er auf vorhergehende Knoten verweist. Hier muß (über das Zeigerfeld am Seitenanfang) auf den nächsten Schlüsselrecord in der Indexseite zugegriffen werden. Dann ist dieser Record gemäß der oben beschriebenen Regeln zu analysieren.

Wird das Ende der Seite erreicht, ohne daß der Schlüssel übereinstimmt, enthält die Indexdatei den Begriff nicht. Mit diesem Verfahren kann der B-Baum recht schnell abgesucht werden.

Das Feld »key« enthält den Schlüsselwert des betreffenden Datensatzes als ASCII-String. Dies gilt auch für numerische Schlüssel. Die Länge wird im Header der NTX-Datei im Feld »key size« definiert. Ein Datumsindex (z. B. 1.12.1991) wird dann als 8-Byte-Zeichenkette (19911201) gespeichert. Numerische Indexwerte werden ebenfalls als String behandelt (weil DBF-Felder ebenfalls als Zeichenstring gespeichert werden). Die Zahl der Nachkommastellen wird durch das Feld »key dec« im NTX-Header definiert. Zu beachten ist aber, daß der Dezimalpunkt mit im Schlüssel steht (z. B. 999.99), d. h. die Zahl der Dezimalstellen ist um 1 niedriger als in »key size« definiert. Sind die Zahlen kleiner als der Platz im Indexfeld, werden führende Stellen mit Leerzeichen aufgefüllt.

Dieser Sachverhalt wird nochmals graphisch in Bild 2.3 wiedergegeben.

Die einzelnen Seiten besitzen eine Länge von 1024 Byte. Am Anfang der Seite befindet sich jeweils ein Zeigerfeld auf die Einträge in der Indexseite. Diese Einträge enthalten die Schlüsselbegriffe (z. B. A, E, L etc.), sowie die Zeiger auf die Folgeseiten. Weiterhin verweist ein Zeiger direkt auf den zugehörigen Datensatz in der DBF-Datei. Der Eintrag »3:E« bedeutet zum Beispiel, daß zum Schlüssel »E« direkt der dritte Record in der DBF-Datei gehört. Die Zahl 3 symbolisiert dabei die Recordnummer. Die logisch vorhergehende Seite enthält alle Schlüssel, die in der Sortierreihenfolge vor »E« kommen. (Der Index wurde in obigem Beispiel auf 1 Zeichen begrenzt). Alle Schlüssel, die größer als »E« sind, finden sich entweder in einem der folgenden Einträge der Seite oder auf einer der Folgeseiten. Der Aufbau der Indexseiten ist mit dieser Organisation wesentlich effizienter als bei den NDX-Seiten von dBASE III.

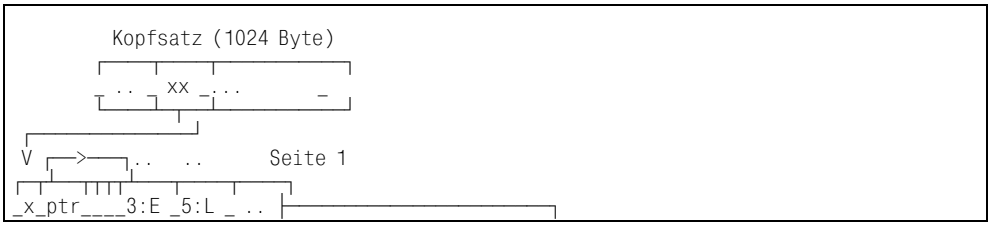


Abbildung 2.3 Schema des NTX-B-Baumes

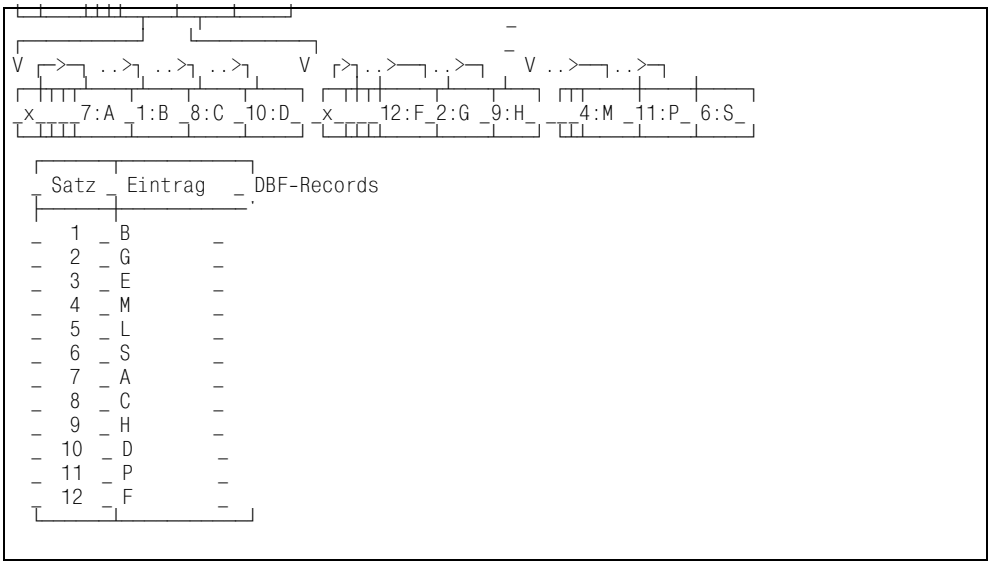


Abbildung 2.3 Schema des NTX-B-Baumes

MEM-Dateiformat in dBASE III

In dBASE III und dBASE III+ ist eine Sicherung der aktuellen Variablen in eine Datei möglich. Diese Datei enthält für jede gespeicherte Variable einen Record, bestehend aus einem 32-Byte-Header, gefolgt vom Inhalt der Variablen. Der Header besitzt die Struktur aus Tabelle 2.9.

Bytes	Bemerkungen
1-11	Name der Variablen als ASCIIZ-String
12	Type der Variablen
13-16	4 Füllbytes (unbenutzt)
17	Länge der Variablen in Byte
18	Zahl der gültigen Dezimalstellen
19-32	Füllbytes (unbenutzt)
32-*n*	Wert der Variablen

Tabelle 2.9 Format einer dBASE III-MEM-DateiFormat einer dBASE III-MEM-Datei

Die ersten 11 Byte enthalten den Namen der Variablen als ASCIIZ-String, d.h. das letzte Byte wird mit 00H abgeschlossen. Daran schließt sich der Typ der Variablen an.

Es lassen sich Variablen vom Typ

C = Character
D = Date

L = Logical
N= Numerisch

speichern. Bei der Kodierung wird der ASCII-Code für den Typ (z. B. C = 43H) benutzt. Allerdings wird immer das oberste Bit gesetzt. Aus dem Zeichen C wird dann der Wert

```
C =>43H OR 80H => C3H
```

ermittelt und als Code gespeichert. Analog wird mit den restlichen Variablen verfahren. Die nächsten 4 Byte sind unbenutzt und dienen als Füllbytes. Ab Offset 17 (11H) folgt ein Byte mit der Längenangabe für die Variable. Im Byte ab Offset 18 (12H) findet sich die Zahl der Nachkommastellen für numerische Werte. Die restlichen 14 Byte des Headers sind unbelegt. Daran schließen sich *n* Bytes mit dem Wert der Variablen an. Bei Zeichenvariablen wird der Inhalt als ASCIIZ-String abgespeichert. Ist der Text kürzer als das reservierte Feld, werden die folgenden Stellen mit Nullbytes aufgefüllt. Bei Logical Variablen reserviert dBASE III ein Byte für den Wert und belegt das Byte mit dem Wert 00 (false) oder 01 (true). Bei numerischen Werten erfolgt die Kodierung in einer dBASE III-internen Notation (8 Byte Fließkommazahl). Datumsvariablen werden ebenfalls als Fließkommazahlen behandelt. Das Ende des gültigen Speicherbereichs (EOF) wird durch den Code 1AH markiert.

DBT-Dateien in dBASE III (Memo-Dateien)

Ab dBASE III wurden erstmals Memo-Dateien zur Aufnahme von Texten eingeführt. In den eigentlichen Datenbankdateien (DBF-Dateien) findet sich dann nur ein Feld mit einem Zeiger auf die eigentliche Memo-Datei bzw. auf einen Textblock innerhalb der Datei (Bild 2.4).

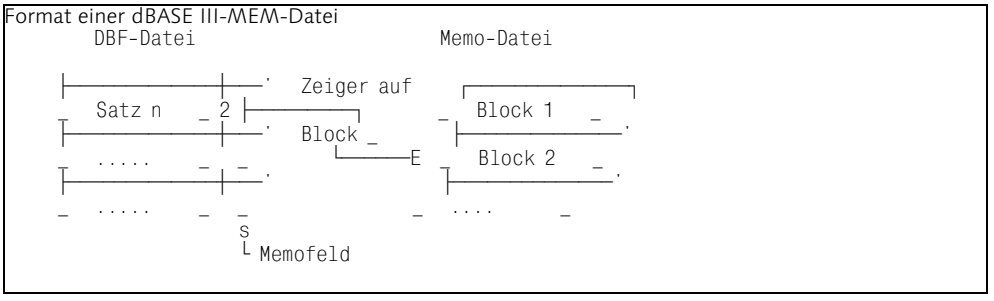


Abbildung 2.4 Referenz auf einen Block der Memo-Datei

Der Zeiger im Memofeld der DBF-Datei ist für den Anwender unsichtbar. Falls kein Textblock für diesen Satz vorliegt, besitzt die DBF-Datei für diesen Satz einen Leereintrag im Memofeld. Andernfalls steht dort ein 10 Byte langer Zeiger, der als ASCII-Zahl interpretiert wird. dBASE III legt neben der DBF-Datei eine zweite Datei gleichen Namens, je-

doch mit der Erweiterung DBT ab. Hier sind die Texte des jeweiligen Memofeldes gespeichert. Die Memo-Datei wird in Sätze zu je 512 Byte unterteilt. Im Kopfsatz sind nur die ersten 4 Bytes belegt – sie geben den nächsten freien Satz der Memo-Datei an (Bild 2.5).

Aus Bild 2.5 ist ersichtlich, daß der Zeiger im Kopf der Memo-Datei immer auf das Ende der Datei gerichtet ist.

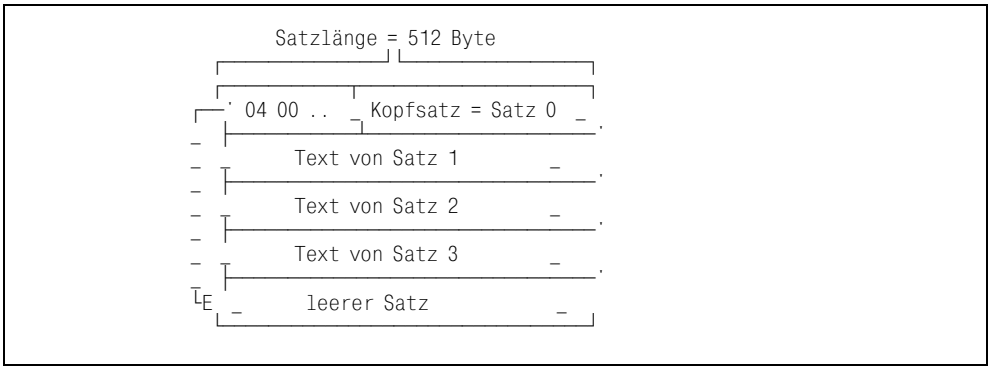


Abbildung 2.5 Satzaufbau einer Memo-Datei

Soll ein neuer Text zu einem Memofeld gespeichert werden, liest dBASE den Kopfzeiger der Memo-Datei und trägt den Wert in das entsprechende Memofeld der DBF-Datei ein. Dann wird der Text einfach an das Ende der bisherigen Memo-Datei angefügt. Ist der Text länger als 512 Byte, wird einfach ein Vielfaches davon angehängt. Das Ende des Textes in einem Memo-Feld wird durch den Code 1AH markiert. Der Satz wird dann bis zur 512-Byte-Grenze mit Füllbytes ergänzt. Der Textzeiger wird in diesem Fall auf den nächsten freien Satz berechnet und im Kopf der Memo-Datei gespeichert.

Bei Änderungen in Memo-Texten macht sich eine gravierende Schwäche der Memo-Dateiverwaltung bemerkbar: Dadurch, daß der geänderte Text einfach an das Ende der Datei angefügt und der neue Zeiger im Memofeld der DBF-Datei eingetragen wird, bleibt der alte Text in der Memo-Datei erhalten – jedoch ohne Zeiger, was zur Folge hat, daß er nicht mehr gefunden werden kann. Dazu kommt, daß dieses Verfahren bei häufigen Textänderungen die Memo-Dateien stark vergrößert. Abhilfe schafft erst das dBASE III-Kommando COPY ....., über das sich unbenutzte Texte aus der Memo-Datei entfernen lassen.

Bei Texten mit mehr als 512 Byte (max. 64 Kbyte) belegt dBASE III einfach einen weiteren 512 Byte großen Satz. Dies wird so lange wiederholt, bis alle Bytes des Textes gespeichert sind. In der DBF-Datei findet sich dann der Zeiger auf den Anfang des Textblocks. Die folgenden Bytes bis zum Ende des 512-Byte-Satzes sind damit undefiniert. Das Textende wird durch zwei 1AH-Zeichen abgeschlossen, und im Header der Memo-Datei findet sich abschließend der Zeiger auf den nächsten freien Satz. Der Zeiger im Memofeld der

DBF-Datei spezifiziert damit den DBT-Satz (512-Byte-Record), ab dem der zugehörige Text beginnt.

FRM-Dateien in dBASE III

In dBASE III lassen sich Formatvorgaben für Reports in FRM-Dateien speichern. Das Format dieser FRM-Dateien wird nachfolgend kurz beschrieben.

Die FRM-Datei besitzt eine Datenstruktur gemäß Tabelle 2.10.

Offset	Byte	Bemerkungen
00H	2	Signatur (sign)
02H	2	Zeiger auf Ende Ausdruck (exp_end)
04H	55*2	Feld (55) Länge Ausdruck (exp_length)
72H	55*2	Feld (55) Index (exp_index)
E0H	1440	String mit den Ausdrücken (exp_area)
680H	 2 2 1 1 2 2 2	Datenstruktur 25 mal FRM-FIELD: width pad1 pad2 total dec exp_contents exp_header
7ACH	2	title_exp_num
7AEH	2	grp_on_exp_num
7B0H	2	sub_on_exp_num
7B2H	2	grp_head_exp_num
7B4H	2	sub_head_exp_num
7B6H	2	page_width
7B8H	2	line_per_page
7BAH	2	left_margin
7BCH	2	right_margin
7BEH	2	num_of_cols
800H	1	dbl_space
801H	1	summary
802H	1	eject
803H	1	plus_bytes
804H	2	sign2

Tabelle 2.10 Tabelle 2.10 – Die Struktur einer FRM-Datei

Das 2-Byte-Feld »sign« enthält die Signatur 0002H für gültige FRM-Dateien. In der Datei ergibt sich wegen der Speicherung der Daten dann die Bytefolge 02 00.

Die Ausdrücke (Expressions) für die Reports werden in einem eigenen Datenbereich, der »exp_area«, innerhalb der Datei abgelegt. Dies ist nichts anderes als ein 1440 Byte langer Textstring. Ausdrücke können dabei statische Texte oder Formeln mit Variablen etc. sein. Formeln werden dann zur Laufzeit durch dBASE ausgewertet. In dem 2-Byte-Feld »exp_end« (Offset 02H) findet sich ein Zeiger auf das erste freie Zeichen im Bereich mit den Ausdrücken (exp_area).

Das Feld »exp_length[55]« umfaßt 55 Einträge à 2 Byte und enthält für jeden Ausdruck innerhalb des Textbereiches dessen Länge in Byte. Das folgende Feld »exp_index[]« gibt dann den Index des jeweiligen Ausdrucks an.

Im Feld »exp_index[55]« findet sich für jeden Ausdruck ein Zeiger auf den Beginn des Textes mit dem Ausdruck. Weiterhin gibt dieses Feld implizit die Reihenfolge der auszuwertenden Ausdrücke an.

An das Feld mit den Indizes schließt sich ein 1440 Byte langer Bereich an, in dem die eigentlichen Ausdrücke (statische Texte oder Berechnungsvorschriften) als Texte abgespeichert werden. Das Ende des belegten Bereiches wird im Feld »exp_end« (Offset 02H) angegeben. Der Anfang der einzelnen Ausdrücke und deren Länge in Byte wird in den beiden Feldern »exp_index[]« und »exp_length[]« gespeichert.

An den Bereich mit den Ausdrücken schließt sich eine Datenstruktur mit 25 Elementen an, die als »FRM_FIELD[25]« bezeichnet wird. Diese Datenstruktur enthält für jedes im Report benutzte Feld einen Eintrag. Allerdings bleibt das erste Feld (Index 0) unbelegt. Jedes Element von »FRM_FIELD« besitzt folgende Variable:

width

Diese 2-Byte-Variable definiert die Druckbreite (Zahl der Zeichen) für den auszugebenden Wert des Feldes.

plan1, plan2

Dies sind Variablen, die als Füllmuster dienen. Pad1 belegt dabei 2 Byte, während pad2 ein Byte umfaßt.

total

Dieses Byte definiert, ob ein numerisches Feld als Summe (total) auszugeben ist (Y oder N).

dec

Bei numerischen Feldern gibt diese Variable die Zahl der Dezimalstellen an.

exp_contents

Bei verschiedenen Ausgaben kann das Ergebnis aus einer Berechnung kommen. Dann gibt die Variable »exp_contents« die Nummer des Ausdrucks der Berechnung an.

exp_header

Diese Variable enthält die Nummer des Textstrings (aus dem Feld Expressions), der dem Feld zugeordnet wurde.

Damit ist die Beschreibung der Elemente aus FRM_FIELD abgeschlossen. Die folgenden Ausführungen beziehen sich wieder auf die Einträge der Dateistruktur.

Das Feld »title_exp_num« umfaßt 2 Byte und enthält die Nummer des Ausdrucks (Expression) für die Titelzeile des Reports. Bei diesem Ausdruck handelt es sich um einen einfachen String.

Das 2-Byte-Feld »grp_on_exp_num« enthält die Nummer des GROUP ON-Ausdrucks. Das 2-Byte-Feld »sub_on_exp_num« enthält dagegen die Nummer des SUB GROUP ON-Ausdrucks. Das 2-Byte-Feld »grp_head_exp_num« enthält die Nummer des GROUP ON-Kopftextes (wird als Ausdruck angegeben). Das 2-Byte-Feld »sub_head_exp_num« enthält die Nummer des SUB GROUP ON-Kopftextes (Angabe als Ausdruck).

Die folgenden Felder umfassen alle 2 Byte und beziehen sich auf die Formatierung der Druckseite. Im Feld »page_width« wird die Zahl der Zeichen pro Zeile (Seitenbreite) angegeben. Daran schließt sich das Feld »line_per_page« an, welches die Zahl der Zeilen einer Druckseite definiert. In »left_margin« wird die Breite des linken Randes (in Zeichen) eingestellt. Das gleiche gilt für den rechten Rand, dessen Breite im Feld »right_margin« steht. Die letzte Angabe »num_of_cols« gibt die Zahl der Spalten im Report an. Dies entspricht auch der Zahl der verwendeten Felder im Ausdruck.

Die nächsten Felder besitzten jeweils nur ein Byte und steuern die Ausgabe. Mit »dbl_space« wird selektiert, ob die Zeichen gesperrt (double spacing) ausgegeben werden. Steht im Feld ein »Y«, wird der Modus »double spacing« eingeschaltet. Mit »N« wird der Modus ausgeschaltet. Im nächsten Feld »summary« wird mit dem Eintrag »Y« signalisiert, daß eine Summe unter der Ausgabespalte zu bilden ist. Mit »N« wird keine Summe unterhalb der Spalte ausgegeben.

Das Feld »eject« definiert, ob ein Seitenvorschub nach der Ausgabe einer Gruppe durchzuführen ist. Mit »Y« erfolgt ein Seitenvorschub, während mit »N« dieser Vorschub unterbleibt.

Das Feld »plus_bytes« wird in allen dBASE-Versionen vor III+ nicht benutzt. Ab der Version dBASE III+ enthält es drei Bits, die die Ausgabe eines Reports steuern:

Bit 0:Seitenvorschub vor dem Report
Bit 1:Seitenvorschub nach dem Report
Bit 2:einfacher Report (plain report) ohne Vorschub

Um eine Option einzuschalten, wird das betreffende Bit auf 1 gesetzt.

Die Datei wird durch ein Wort mit der Signatur 0002H (Bytefolge 02 00) abgeschlossen.

LBL-Dateien in dBASE III

In dBASE III lassen sich Formatvorgaben für Labels in LBM-Dateien speichern. Die LBL-Datei besitzt eine Datenstruktur gemäß Tabelle 2.11.

Offset	Byte	Bemerkungen
00H	1	Signatur (sign)
01H	60	Bemerkungen (remarks)
3DH	2	Höhe (height)
3FH	2	Breite (width)
41H	2	linker Rand (left margin)
43H	2	Labelzeile (label line)
45H	2	Zwischenraum Zeilen (label space)
47H	2	Label pro Druckreihe (label across)
49H	16*60	Labeltext (info[16][60])
409H	1	Signatur 2 (sign2)

Tabelle 2.11 Tabelle 2.11 – Die Struktur einer LBL-Datei

Das 1-Byte-Feld »sign« enthält die Signatur 02H für gültige LBL-Dateien. Daran schließt sich ein maximal 60 Zeichen umfassender Kommentartext an, der die vordefinierte Größe des Labels spezifiziert. Dieser Text ist meist mit Leerzeichen gefüllt.

Das Feld »height« enthält die Zahl der Zeilen im Label, während das Feld »width« die Druckbreite der einzelnen Zeilen des Labels definiert. In »left_margin« wird die Zahl der Leerzeichen für den linken Rand des Labels angegeben.

Die folgenden Parameter beziehen sich auf die Druckersteuerung, falls sich auf einem Ausgabebogen mehrere Labels befinden, die gleichzeitig bedruckt werden. Das Feld »label_line« definiert den Zwischenraum (in Zeilen) zwischen einzelnen Labelreihen. Der Parameter »label_space« definiert die Zahl der Leerzeichen zwischen einzelnen Labels einer Zeile. Diese Zahl ist für den Vorschub des Druckkopfes zum Anfang des nächsten (rechts stehenden) Labels wichtig. Der Parameter »label_across« gibt an, wie viele Labels sich in einer Reihe auf dem Druckbogen befinden.

Ab Offset 49H beginnt ein Bereich mit dem Text des Labels. Das Label darf dabei 16 Zeilen Text mit einer Länge von 60 Zeichen pro Zeile enthalten. Der Textbereich ist deshalb auch als Feld mit 16 Zeilen zu 60 Zeichen organisiert. In diesen Strings stehen dann die Ausdrücke zur Erzeugung der Druckzeile.

Die Datei wird ab Offset 409H mit der zweiten Signatur 02H (1 Byte) abgeschlossen.

Das Format der Datei DBPRINT.PTB

Ein leidiges Thema ist die Anpassung von Druckern unter dBASE III: Oft steht der Anwender vor dem Problem, daß bestimmte Umlaute oder Sonderzeichen auszugeben sind. Zur Hilfestellung existiert hier in dBASE III+ zwar eine Datei DBPRINT.PTB, Funktion und Aufbau dieser Datei sind jedoch weitgehend undokumentiert.

Sehen wir uns die Sätze dieser Datei genauer an (Tabelle 2.12):

Byte	Code	Bedeutung
1	00	Header Satzanfang
2	xx	erstes Codebyte (dBase-Zeichen)
3	xx	zweites Codebyte (Druckercode) optional

Tabelle 2.12 Tabelle 2.12 – Satzaufbau von DBPRINT.PTB

Der Inhalt der Datei dient zur Anpassung des angeschlossenen Druckers an die auszugebenden Zeichen. Dies ist zum Beispiel immer dann erforderlich, wenn der Drucker nicht direkt auf den Zeichensatz der MS-DOS-Personal-Computer abgestimmt ist. Der Ausdruck von Sonderzeichen und Umlauten funktioniert hier meist nicht. So erhält der Benutzer anstelle des deutschen Umlautes »Ä« im Ausdruck eine eckige Klammer (»[«); das deutsche »ß« gibt es überhaupt nicht. Die Ursache dieses Problems liegt darin, daß alle Zeichen über ASCII 128 in der ASCII-Tabelle nicht genormt sind. So kann es durchaus vorkommen, daß der Rechner den Code für das Zeichen »ß« zwar an den Drucker sendet und auch auf dem Bildschirm korrekt darstellt, weil es mit dem Zeichensatz des Rechners übereinstimmt. Ein Drucker mit abweichender Belegung wird jedoch für den empfangenen Code ein eigenes Zeichen ausgeben. Um das Zeichen, das in der Zeichentabelle des Druckers einem anderen Code zugeordnet ist, dennoch darstellen zu können, muß per Umsetzungstabelle das dBase-Zeichen in diesen Code umgewandelt werden – und genau diese Umwandlungstabelle ist in der Datei DBPRINT.PTB abgespeichert. Ist diese Datei beim Start von dBASE III+ vorhanden, wird sie geladen und dient zur Umkodierung der Druckerausgaben.

Die Datei besteht in der Regel aus mehreren Bytes, die in Sätzen variabler Länge abgespeichert sind. Jeder Satz beginnt mit einem 1 Byte langen Header, der den *Satztyp* spezifiziert. Insgesamt sind 2 verschiedene Headertypen zulässig:

Header	Bedeutung
00	Datensatz mit 2-3 Byte (einschließlich Header)
08	Kommentarsatz mit *n* Byte

Tabelle 2.13 – Headertypen – Headertypen

Sobald in der Datei ein weiterer Header (00H oder 08H) auftritt, wird der aktuelle Satz beendet. Das Ende der Datei ist durch einen Leersatz mit zwei Nullbytes (00 00) markiert. Die Datei beginnt meist mit dem Header 08H, gefolgt von einem Kommentartext mit Hinweisen zur Druckeranpassung (Bild 2.6).

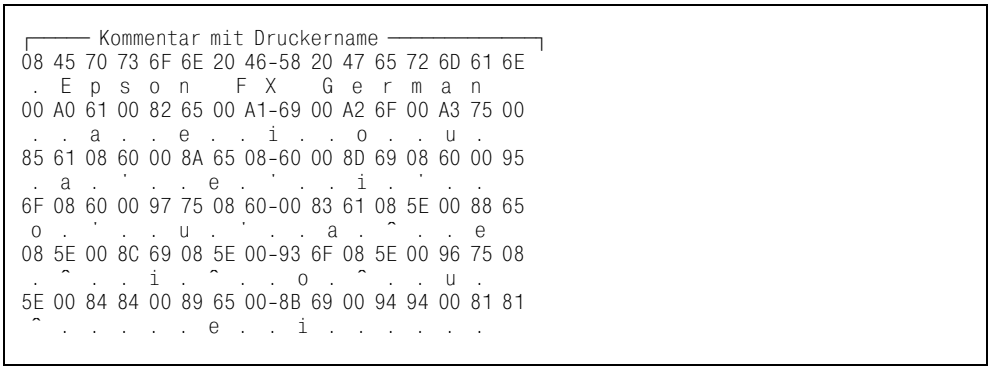


Abbildung 2.6 Ausschnittsdump der Datei DBPRINT.PTB

An den Kommentartext schließen sich die eigentlichen Sätze mit den Druckercodes an, und zwar mit dem in Tabelle 2.12 gezeigten Aufbau. Nach dem Header (00) wird im zweiten Byte der ASCII-Code des zu ersetzenden Zeichens abgelegt (beispielsweise der Wert 41H für den Ersatz des Buchstabens »A«). Im dritten Byte folgt dann der ASCII-Code des Ersatzbuchstabens. Wird hier zum Beispiel der Wert 40H eingetragen, gibt dBASE bei allen »A«-Zeichen den Buchstaben mit dem Code 40H (@, das sogenannte »Klammeräffchen«) auf dem Drucker aus.

Falls ein auszugebendes Zeichen im Zeichensatz des Druckers fehlt oder bei der Ausgabe unterdrückt werden soll, muß dessen Code in die Tabelle mitaufgenommen werden. Für das Zeichen wird ein Satz in der Tabelle angelegt, in dem nur der Header und das zweite Byte eingetragen werden. Da das dritte Byte fehlt, unterdrückt dBase das Zeichen bei der Ausgabe auf dem Drucker.

Einträge aus der Datei DBPRINT.PTB zu entfernen, ist recht einfach, wenn das Headerbyte anstelle von 00H mit dem Wert 08H belegt wird. dBASE interpretiert diesen Eintrag dann als Kommentar, und eine spätere Aktivierung ist durch Änderung des Headerbytes möglich.

3 Dateiformate in dBASE IV

Als Nachfolger von dBASE III+ entwickelte Ashton Tate dBASE IV, das viele Einschränkungen seiner Vorgänger behob. Die folgenden Seiten beschreiben die Struktur der DBF-Dateien dieser Programmversion.

DBF-Dateiformat in dBASE IV

Der Aufbau dieser Dateien lehnt sich an die Struktur von dBASE III an, wenn auch die Leistungen der neuen Version erweitert wurden. Jede DBF-Datei in dBASE IV besteht wie bei den älteren Versionen aus drei Teilen: dem Header, der Satzbeschreibung und den eigentlichen Daten.

Die Belegung des Kopfsatzes mit dem Header und der Datensatzbeschreibung hängt von der Konfiguration des Programms ab. Seine Struktur ist in Tabelle 3.1 dargestellt.

Offset	Bytes	Bemerkungen
0	1	Nummer der dBASE-Version Bit 0-2 dBASE-Version (1=dBASE IV) Bit 3 Memofeldindikator Bit 4-6 reserviert für SQL Bit 7 Flag für dBASE III+ Memodateien
1	3	Datum des letzten Schreibzugriffs im Binärformat (JJMMTT)
4	4	Zahl der Datensätze in der Datei
8	2	Länge des Headers in Byte
10	2	Länge eines Datensatzes in Byte
12	2	reserviert
14	1	Flag für nichtbeendete Transaktionen
15	1	Flag für Verschlüsselung
16	12	reserviert für die Netzwerkversion
28	1	Markierung für den Arbeitsindex 01H = MDX-Datei vorhanden 00H = kein Multikey-Index vorhanden
29	3	reserviert
32	32 * N	32 Byte pro Feld mit der Beschreibung des Aufbaus
32*N+1	1	Wert 0DH als Markierung Header Ende

Tabelle 3.1 Format eines dBASE IV-DBF-Headers

Die Informationen sind ähnlich wie in dBASE III gemischt im ASCII- und Binärformat gespeichert.

Das erste Byte dient zur Identifizierung der dBASE-Version. Bei dBASE II wurde hier der Wert 02 abgelegt, ab dBASE III findet sich im unteren Nibble (Bit 0 ... 2) der Wert 3H. Das oberste Bit (7) zeigt an, ob die Datei Memofelder enthält. In diesem Fall ist der DBF-Datei eine DBT-Datei mit den Memotexten zugeordnet, und das Byte enthält demnach den Code 83H. In allen anderen Fällen findet sich der Wert 03H im ersten Byte. Findet dBASE IV einen anderen Wert, lehnt es einen Zugriff ab, da es sich dann nicht um eine DBF-Datei handeln kann. Die Bits 4 bis 6 sind zur Zeit noch für Dateien im SQL-Format reserviert.

Anmerkung: Es gibt Hinweise, daß dBASE IV-Versionen den Wert 7BH für DBF-Dateien mit Memofeldern verwenden. Dann wird als Versionsnummer der Wert 01 benutzt.

Das nächste Feld umfaßt 3 Byte mit dem im Binärformat kodierten Datum des letzten Schreibzugriffs in der Form JJMMTT – im ersten Byte steht also wieder das Jahr (0...99).

Das folgende Feld umfaßt 4 Byte, in denen die Zahl der Datensätze in der DBF-Datei geführt wird. Die Bytes werden als vorzeichenlose 32-Bit-Zahl interpretiert, wobei die üblichen Intel-Konventionen zur Speicherbelegung (niederwertiges Byte der Zahl auf der untersten Adresse) gelten. Der Wert umfaßt alle Sätze, also auch solche, die bereits zum Löschen markiert sind.

Das nächste Feld umfaßt eine vorzeichenlose 16-Bit-Zahl, in der die Länge des Headers in Bytes steht. Der Header enthält damit 32 Byte plus *n* Sätze à 32 Byte mit der Feldbeschreibung sowie das Abschlußbyte mit dem Code 0DH. Diese Headerlänge wurde von dBASE III+ übernommen.

Die Länge eines Datensatzes wird ab Byte 10 als vorzeichenlose 16-Bit-Zahl geführt. Dieser Wert ist immer um ein Byte höher als die rechnerische Summe der einzelnen Feldlängen, was darin begründet ist, daß am Anfang eines Datensatzes immer ein Byte zur Markierung gelöschter Sätze reserviert wird.

Ab Byte 12 folgt ein 20 Byte großer reservierter Bereich für die interne Verwendung. Ab Offset 14 verwaltet dBASE IV hier beispielsweise ein Flag, das die erfolgreiche oder erfolglose Durchführung einer Transaktion anzeigt. Bei unvollständigen Transaktionen bleibt das Flag gesetzt. Der Befehl BEGIN TRANS-ACTION setzt den Wert des Flags auf 01H. Die Befehle END TRANSACTION und ROLLBACK löschen das Flag wieder.

Mit der dBASE IV-Funktion ISMARKED() läßt sich der Status dieses Flags prüfen.

Byte 15 markiert, ob die Daten innerhalb der Datei durch dBASE verschlüsselt wurden. Der Wert 0 steht für unverschlüsselte Daten, während mit 1 die Daten verschlüsselt gespeichert werden. Ein Zurücksetzen des Wertes von 1 auf 0 verursacht jedoch keine Entschlüsselung dieser Daten, da dies nur durch dBASE selbst erfolgen kann. Eine Verschlüsselung ist grundsätzlich erst ab dBASE IV möglich.

Das in dBASE III reservierte Byte 28 wird in der Version IV zur Markierung von Multikey-Indexdateien benutzt. Wurde eine solche Datei durch dBASE aufgebaut, enthält das Byte den Wert 01H; sonst enthält das Byte den Wert 00H.

Die Informationen über den Aufbau der Datensätze schließen sich wie in dBASE III an den Header an. Auch hier gibt es wieder einzelne Feldbeschreibungen, wobei in dBASE IV bis zu 255 Felder definierbar sind. Für jedes Feld der Datenbank findet sich ein Satz mit 32 Byte, der das in Tabelle 3.2 gezeigte Format besitzt.

Die ersten 11 Byte der Feldbeschreibung enthalten den Feldnamen (10 Zeichen + 00H), der als ASCIIZ-Text abgelegt wird. dBASE lehnt sich hier stark an die Sprache C an, die Zeichenketten ebenfalls mit einem Nullbyte abschließt. Umfaßt der Feldname keine 11 Zeichen, werden die restlichen Bytes mit dem Wert 00H aufgefüllt.

Offset	Bytes	Bemerkungen
0	11	Name des Feldes in ASCII-Zeichen
11	1	Feldtyp in ASCII (C, N, L, D, M)
12	4	Datenadresse des Feldes im Speicher
16	1	Feldlänge in Byte als Binärzahl
17	1	Zahl der Nachkommastellen in Byte
18	2	reserviert für Mehrbenutzerbetrieb
20	1	ID für Arbeitsbereich
21	11	reserviert
31	1	Flag für Arbeitsindex-Feld (MDX) 01, falls MDX einen Subindex besitzt

Tabelle 3.2 Die DBF-Feldbeschreibung in dBASE IV

Im nächsten Byte steht das ASCII-Zeichen für den Feldtyp (Tabelle 3.3 zeigt die Kodierung der gültigen Feldtypen für dBASE IV). Gegenüber dBASE III+ gibt es hier bei dBASE IV nun ein erweitertes Fließkommaformat (F).

Zeichen	Feldtyp	erlaubte Zeichen
C	Character	ASCII-Zeichen
N	Numerisch 1	- 0...9
F	Numerisch 2	- 0...9
L	logisch	JjNnTtFf ?
D	Datum	JJJJMMTT
M	Memofeld	DBT-Blocknummer

Tabelle 3.3 Kodierung der Feldtypen in dBASE IVKodierung der Feldtypen in dBASE IV

An den Feldtyp schließt sich ein 4 Byte langer Vektor an, der von dBASE intern zur Speicherung der Feldadresse benutzt wird. Dieser Wert hat für externe Zwecke keine Bedeutung.

Die Feldlänge findet sich ab Offset 16 und ist in einem Byte als Binärcode abgelegt. Damit kann ein Feld maximal 255 Zeichen umfassen, was aber nur bei Zeichenfeldern (max. 254 Zeichen) ausgenutzt wird. Bei numerischen Feldern gibt der Wert die Zahl der Dezimalstellen einschließlich des Dezimalkommas an.

dBASE bietet als Neuerung ab Version IV zwei Möglichkeiten zur Darstellung von Fließkommazahlen: Mit dem neuen F-Format werden die Daten intern in einer Gleitkommadarstellung mit Binärarithmetik bearbeitet, was zu größerer Genauigkeit führt. Alternativ lassen sich die Daten im Dezimalmodus in dem bereits von dBASE II bekannten N-Format darstellen. Hier ist die Genauigkeit auf ca. 15 Stellen begrenzt.

Bei Memofeldern beträgt die Feldlänge immer 10 Byte, da dort die Blocknummer für den Satz in der zugehörigen DBT-Datei gespeichert wird (weitere Erläuterungen hierzu bei der Beschreibung des DBT-Formates). Logische Felder besitzen die Länge 1, während bei Datumsfeldern immer 8 Byte reserviert werden.

Bei numerischen Feldern spezifiziert das Folgebyte dann die Zahl der Nachkommastellen. Bei allen anderen Feldtypen besitzt der Eintrag den Wert 00H. Wichtig ist, daß die Zahl der Nachkommastellen immer kleiner als die Feldlänge ist.

Die restlichen 14 Byte sind für interne Zwecke reserviert. Sie müssen lediglich beim Zugriff auf die Feldbeschreibung überlesen werden. Auch das SET FIELDS-Byte ist nicht weiter relevant, da DBASE IV diesen Eintrag offensichtlich nur im Speicher benutzt.

Jedes definierte Feld der Datenstruktur besitzt einen eigenen 32-Byte-Satz im Kopf der DBF-Datei. Das Ende der Felddefinition wird durch das Zeichen 0DH markiert. In dBASE IV lassen sich bis zu 255 Felder pro Satz mit einer Recordlänge von maximal 4000 Byte definieren. Das letzte Feld wird mit dem Zeichen 0DH (*Carriage Return*) abgeschlossen. Falls nicht alle Felder definiert sind, steht das Zeichen hinter der letzten Felddefinition.

Die eigentlichen Datensätze werden wie bei dBASE III an den Definitionsteil angehängt. Die Recordlänge richtet sich nach der Länge der jeweiligen Felder und wird im Header der Datei geführt. Der Wert ist immer um ein Byte größer als die Summe der Feldlängen, da vor jedem Record ein Byte für die DELETE-Markierung reserviert wird. Ein Datensatz wird im reinen ASCII-Format ohne Trennzeichen gespeichert. Damit ist der Import und Export von Daten über die Option SDF natürlich recht einfach. Weiterhin lassen sich alle Felder typunabhängig als ASCII-Text bearbeiten.

Sobald über APPEND BLANK ein neuer Satz an die Datei angehängt wird, füllt ihn dBASE mit Blanks auf. Das erste Byte im Satz dient zur Markierung gelöschter Daten, und da beim neuen Satz ein Blank eingetragen ist, kann er nicht gelöscht werden. Erst wenn das

erste Byte das Zeichen »*« enthält, wird der Satz beim nächsten PACK-Befehl aus der DBF-Datei entfernt. Dies führt zu sehr schnellen DELETE-Operationen und ermöglicht sogar ein UNDELETE ohne großen Aufwand. Die Sätze stehen jedoch nach wie vor in der Datenbank, so daß diese durchaus mehrere hundert Sätze enthalten kann, von denen keiner gültig ist. Zugriffe ohne Index werden dann natürlich recht langsam, da alle Sätze (gelöscht oder ungelöscht) zu lesen sind. Um diesen Nachteil zu korrigieren, sollten gelöschte Sätze möglichst häufig über PACK aus der DBF-Datei entfernt werden. Records, die zum Löschen markiert sind, werden dann durch nachfolgende gültige Sätze überschrieben; der Eintrag im Header wird auf die Zahl der ungelöschten Records reduziert.

Der Abschluß des gültigen Datenbereiches wird durch das Zeichen 1AH markiert. Im Gegensatz zu früheren dBASE-Versionen verändert dBASE IV mit dem PACK-Befehl auch die physikalische Dateigröße. Nach dieser Operation befindet sich die DOS-EOF-Marke direkt hinter dem Zeichen 1AH, womit die gelöschten Sätze endgültig entfernt sind.

Bild 3.1 gibt einen Ausschnitt aus einer DBF-Datei in dBASE IV als Hexdump wieder.

DBT-Dateiformat in dBASE IV

Die Memo-Dateien dienen zur Aufnahme von Texten. In den eigentlichen Datenbankdateien (DBF-Dateien) findet sich dann nur ein Feld mit einem Zeiger auf die eigentliche Memo-Datei. Dabei ergibt sich die Struktur aus Bild 3.2.

Der Wert des jeweiligen MEMO-Feldes in der DBF-Datei ist als Zeiger (Blocknummer) auf einen Eintrag in der zugehörigen DBT-Datei zu verstehen. Die DBT-Datei besitzt den gleichen Namen wie die DBF-Datei und unterscheidet sich lediglich in der Erweiterung. DBT-Dateien werden dann in Records zu *n* Bytes unterteilt, wobei sich die Recordlänge in dBASE IV mit dem Befehl SET BLOCKSIZE festlegen läßt. Der Zeiger in der DBF-Datei gibt den Offset in Records in der DBT-Datei an. Falls in der Memodatei kein Text gespeichert ist, enthält das zugehörige Feld in der DBF-Datei 10 Blanks. In dBASE IV wird ein gelöschter Text in der MEMO-Datei wieder freigegeben und kann erneut belegt werden. Die Struktur der Datei lehnt sich an dBASE III an und ist folgendermaßen definiert:

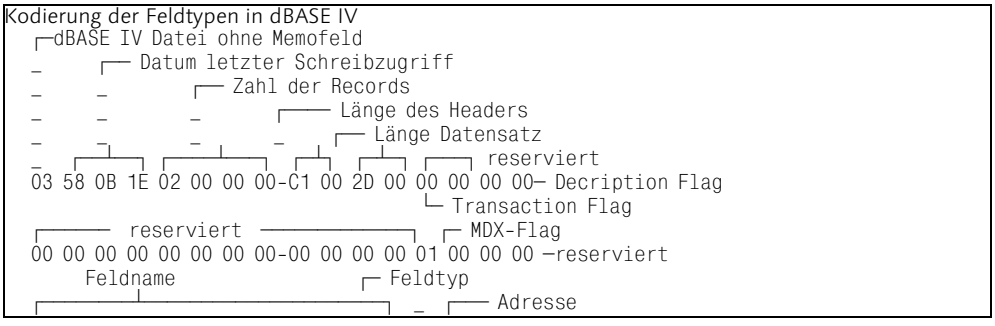


Abbildung 3.1 Dump der dBASE IV-Datenbank TEST.DBF

```
46 45 4C 44 31 00 00 00-00 00 00 43 11 00 A0 47
 F  E  L  D  1  .  .  .
 ┌─ Feldlänge in Bytes
 │  ┌─ Nachkommastellen = 0
14 00 00 00 01 00 00 00-00 00 00 00 00 00 00 00
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
46 45 4C 44 32 00 00 00-00 00 00 4E 25 00 A0 47
 F  E  L  D  2  .  .  .  .  .  .  N  .  .  .  G
05 02 00 00 01 00 00 00-00 00 00 00 00 00 00 00
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
46 45 4C 44 33 00 00 00-00 00 00 44 2A 00 A0 47
 F  E  L  D  3                 D  .  .  .  G
08 00 00 00 01 00 00 00-00 00 00 00 00 00 00 00
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
46 45 4C 44 34 00 00 00-00 00 00 4C 32 00 A0 47
 F  E  L  D  4  .  .  .  .  .  .  L  2  .  .  G
01 00 00 00 01 00 00 00-00 00 00 00 00 00 00 00
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
46 45 4C 44 35 00 00 00-00 00 00 4D 33 00 A0 47
 F  E  L  D  5  .  .  .  .  .  .  M  3  .  .  G
0A 00 00 00 01 00 00 00-00 00 00 00 00 00 00 00
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
0D 20 20 20 20 20 20 20-20 20 20 31 20 20 20 20
    └─ Markierung Datensatzanf.   1
 └─ Ende Header
20 20 20 20 20 20 20 31-2E 30 30 31 39 38 38 30
                    1  .  0  0  1  9  8  8  0
33 31 32 54 20 20 20 20-20 20 20 20 20 20 20 20
 3  1  2  T
20 20 20 20 20 20 20 20-32 20 20 20 20 20 20 20
                       2
20 20 20 20 32 2E 30 30-31 39 33 33 30 33 31 32
           2  .  0  0  1  9  3  3  0  3  1  2
46 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
 F
20 20 20 20 20 33 20 20-20 20 20 20 20 20 20 20
 .....
1A 61
```

Abbildung 3.1 Dump der dBASE IV-Datenbank TEST.DBF

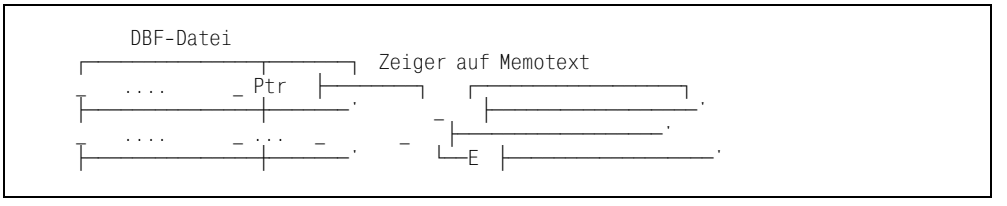


Abbildung 3.2 Zeigerstruktur auf Memotexte in DBT-Dateien

Offset	Bytes	Bemerkungen
00H	4	Zeiger auf ersten freien Block
04H	4	unbelegt

Tabelle 3.4 Header (Block 0) der DBT-Datei (dBASE IV)

Offset	Bytes	Bemerkungen
08H	8	DBF-Dateiname (ohne Extension)
10H	1	Flag: 03H dBASE III Header, sonst 00H
11H	3	reserviert
14H	2	Blocklänge in Byte
16H	n	Füllbytes 00H bis Blockende

Tabelle 3.4 Header (Block 0) der DBT-Datei (dBASE IV)

Die DBT-Datei besteht aus *n* Byte langen Blöcken in fortlaufender Reihenfolge. Die Blocklänge läßt sich mit SET BLOCKSIZE TO definieren und wird in Block 0 ab Offset 14H gespeichert. Zu beachten ist aber, daß dBASE III-DBT-Dateien (aus Kompatibilitätsgründen) die Blocklänge 1 erhalten.

An Block 0 schließen sich *n* Blöcke mit fester Länge an, die belegt oder frei sein können. Belegte Blöcke werden durch die Einträge in der DBF-Datenbank adressiert und besitzen die Struktur aus Tabelle 3.5.

Offset	Bytes	Bemerkungen
00H	4	reserviert (FF FF 08 00H)
04H	4	Länge Memofeldeintrag in Byte
08H	n	Inhalt des Memofeldes

Tabelle 3.5 Header eines belegten DBT-Blocks (BASE IV)

Die Länge der Memofelddaten wird im Kopf des Blocks ab Offset 04H gespeichert. Dadurch entfällt die aus dBASE III bekannte Endemarke (1AH). Im Text werden Absätze mit den Codes 0DH, 0AH abgeschlossen. Ein Zeilenumbruch wird mit 8DH, 0AH markiert.

In Block 0 findet sich ab Offset 00 ein Zeiger auf den ersten freien Block der DBT-Datei. In dBASE IV lassen sich freie DBT-Blöcke wieder belegen. Hierzu werden die freien Blöcke über eine Zeigerstruktur (Tabelle 3.6) verwaltet.

Offset	Bytes	Bemerkungen
00H	4	Zeiger nächster freier Block
04H	4	Zeiger nächster belegter Block
08H	n	Füllbytes bis Blockende

Tabelle 3.6 Verkettung leerer DBT-Blöcke (dBASE IV)

Ausgehend vom DBT-Kopf lassen sich die Blockzahl und die Länge der Blocks ermitteln.

4 Dateiformate in FoxPro

Das mittlerweile von Microsoft vertriebene Produkt FoxPro setzt ebenfalls auf dem DBF-Dateiformat auf. Allerdings wurden mit den verschiedenen Versionen einige Erweiterungen eingeführt, die nachfolgend beschrieben werden.

FoxPro – Format der DBF-Dateien

In FoxPro werden verschiedene Versionen unterschieden:

- FoxPro 1.0
- FoxPro 2.0
- FoxBase+
- FoxPro 2.5 und FoxPro für Windows 2.5a/2.6
- Visual FoxPro 3.0

Der Aufbau der DBF-Dateien basiert auf der Struktur der dBASE III-DBF-Dateien. Jede DBF-Datei besteht gemäß Bild 4.1 aus drei Teilen, dem Header, der Satzbeschreibung und den eigentlichen Daten.

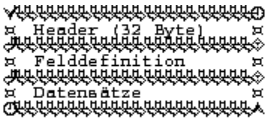


Abbildung 4.1 Struktur einer DBF-Datei in FoxPro

Innerhalb der Datei liegen die Daten gemischt als ASCII-Text und im Binärformat vor. Der Header besteht aus Binärdaten, während die Datensätze als reiner ASCII-Text gespeichert werden.

Der DBF-Header

Der Header umfaßt immer 32 Byte mit der Struktur aus Tabelle 4.1.

Offset	Bytes	Bemerkungen
0	1 03H 03H 83H 8BH F5H 30H	Versionskennung DBF-Datei dBASE III+/FoxBase+ ohne Memo dBASE IV / FoxPro 2.x ohne Memo dBASE III+/FoxBase+ mit Memo dBASE IV mit Memo FoxPro 2.x mit Memo Visual FoxPro 3.0
1	3	Datum der letzten Änderung im Binärformat (JJMMTT)

Tabelle 4.1 Format eines FoxPro-DBF-Header

Offset	Bytes	Bemerkungen
4	4	Zahl der Datensätze in der Datei
8	2	Zeiger auf den 1. Datensatz (Offset in Byte)
10	2	Länge eines Datensatzes einschließlich der Löschmarkierung
12 .. 31	20	reserviert
32	32 * N	32 Byte pro Feld mit der Beschreibung des Feldaufbaus
N+1	1	Wert 0DH als Markierung Header Ende

Tabelle 4.1 Format eines FoxPro-DBF-Header

Das erste Byte dient zur Identifizierung der Version des Programmes, welches die DBF-Datei erzeugt hat. Bei dBASE III wurde hier der Wert 03H abgelegt. Das oberste Bit (7) dient in dBASE III zur Markierung, ob die Datei Memofelder enthält. In FoxBase+ wurde die dBASE III-Notation übernommen. Das gleiche gilt für FoxPro-Dateien, die keine Memofelder enthalten. Sobald eine FoxPro-DBF-Datei DBF-Memofelder enthält, trägt FoxPro die Signatur F5H im ersten Byte ein. In diesem Fall ist der DBF-Datei eine FPT-Datei mit den Memotexten (oder Bilddaten) zugeordnet. Findet FoxPro einen anderen Wert als in Tabelle 4.1 aufgeführt, lehnt es einen Zugriff ab, da es sich dann nicht um eine DBF-Datei handeln kann.

Das nächste Feld umfaßt drei Byte mit dem im Binärformat kodierten Datum des letzten Schreibzugriffs (Format JJMMTT). Dieses Format enthält damit im ersten Byte das Jahr (0 .. 99).

Das folgende Feld ab Offset 4 umfaßt 4 Byte mit der Zahl der Datensätze der DBF-Datei. Der Wert wird als vorzeichenlose 32-Bit-Zahl interpretiert, wobei die üblichen Intel-Konventionen zur Speicherbelegung (Low Byte der Zahl auf der untersten Adresse) gelten. Der Wert umfaßt alle Sätze, also auch solche, die bereits zur Löschung markiert sind.

Das nächste Feld umfaßt eine vorzeichenlose 16-Bit-Zahl, in der die Länge des Headers in Byte steht. Die Länge des Headers variiert insofern, als eine DBF-Datei eine variable Anzahl von Feldefinitionen (siehe unten) enthalten kann. Die Längenangabe des Headers stellt daher einen Zeiger (Offset in Byte) auf den ersten Datensatz dar.

Die Länge eines Datensatzes wird ab Offset 10 (0AH) als vorzeichenlose 16-Bit-Zahl geführt. Dieser Wert umfaßt auch das Byte mit der Löschmarkierung (Blank oder *). Der Wert muß der Summe der einzelnen Feldlängen+1 entsprechen.

Ab Offset 12 (0CH) folgt ein reservierter Bereich mit 20 Byte, der für eine interne Verwendung vorgesehen ist. dBASE IV nutzt den Bereich zum Beispiel zur Netzwerkverwaltung. FoxPro 2.5a (für Windows) speichert im Byte ab Offset 31 die Kennung für die verwendete Codepage. Damit lassen sich in FoxPro die Inhalte von Text- und Datumsfeldern

in unterschiedlichen Zeichensätzen darstellen. Weiterhin erfolgt die Sortierung in Abhängigkeit von den jeweiligen Codepages. Die Belegung ist nicht offiziell dokumentiert. Tabelle 4.2 enthält jedoch eine Aufstellung der bisher verwendeten Codes.

Code	Codepage	Bemerkung
01H	437	US MS-DOS
02H	850	internatial MS-DOS
03H	1251 1252	Russian Windows Windows ANSI-Codes
64H	852	EE MS-DOS
65H	866	Russian MS-DOS
66H	865	Nordic MS-DOS

Tabelle 4.2 Kodierung der Codepage in FoxPro 2.5a

In der FoxPro 2.5b Version werden weitere Codepages unterstützt. Die Markierung für diese Codepages ist zur Zeit jedoch nicht bekannt.

Die Feldbeschreibung

Ab Offset 32 (20H) schließen sich die Felddefinitionen an. Diese lehnen sich an die Vorgaben von dBASE III an, wobei allerdings neue Feldtypen eingeführt wurden. Jede Feldbeschreibung umfaßt 32 Byte, wobei in FoxPro bis zu 255 Felder pro Datensatz zulässig sind. Der Aufbau der Feldbeschreibung wird in Tabelle 4.3 gezeigt.

Offset	Bytes	Bemerkungen
0	11	Feldname in ASCII-Zeichen
11	1	Feldtyp in ASCII (C,N,L,D,M,G,F,P)
12	4	Feldposition im Datensatz
16	1	Feldlänge in Byte (Binärzahl)
17	1	Zahl der Nachkommastellen in Byte
18..31	14	reserviert

Tabelle 4.3 Die FoxPro-DBF-Feldbeschreibung

Die ersten 11 Byte der Feldbeschreibung enthalten den Feldnamen als ASCIIZ-Text, d.h. der Name wird immer mit einem Nullbyte 00H abgeschlossen. Umfaßt der Feldname keine 10 Zeichen, werden die restlichen Byte mit dem Wert 00H aufgefüllt.

Im nächsten Byte steht das ASCII-Zeichen für den Feldtyp. In Tabelle 4.4 findet sich die Kodierung der gültigen Feldtypen für FoxPro 2.x. Gegenüber dBASE III gibt es zusätzlich Objekt-, Gleitpunkt- und Bildfelder.

Zeichen	Feldtyp	erlaubte Zeichen
C	Character	ASCII – Zeichen
N	Numerisch	- . 0 .. 9
L	Logical	JjNnTtFf ?
M	Memofeld	DBT Blocknr.
G	Objekt	FPT Blocknr.
D	Datum	JJJJMMTT
F	Gleitpunkt	- . 0 .. 9
P	Bild	FPT Blocknr.

Tabelle 4.4 Kodierung der FoxPro 2.x-Feldtypen

Die Felddefinition in FoxPro enthält neben den zusätzlichen Feldtypen eine weitere Abweichung zu dBASE. Während dBASE ab Offset 12 (0CH) die interne Datenadresse des Feldes sichert, speichert FoxPro die Feldposition im Datensatz. Damit muß die Reihenfolge der Felddefinitionen nicht mehr mit der Feldreihenfolge im Datensatz übereinstimmen. Weiterhin sind die Bytes ab Offset 18 (12H) reserviert. Die Feldlänge findet sich ab Offset 16 (10H) und ist in einem Byte als Binärcode abgelegt. Damit kann ein Feld maximal 256 Zeichen umfassen. Diese Länge wird aber nur bei Characterfeldern ausgenutzt. Bei numerischen Feldern gibt der Wert die Zahl der Stellen einschließlich des Dezimalpunktes an. In FoxPro lassen sich Zahlen aber nur mit einer Genauigkeit von ca. 15 Stellen verarbeiten. Bei Memo-, Objekt- und Bildfeldern beträgt die Feldlänge immer 10 Byte, da dort die Blocknummer für den Satz in der zugehörigen FPT-Datei (oder DBT-Datei in FoxPro 1.0) gespeichert wird. Weitere Erläuterungen finden sich bei der Beschreibung der FPT- und DBT-Formate. Logical Felder besitzen die Länge 1, während bei Datumsfeldern immer 8 Byte reserviert werden. Bei numerischen Feldern spezifiziert das Folgebyte die Zahl der Nachkommastellen. Bei allen anderen Feldtypen besitzt der Eintrag den Wert 00H. Wichtig ist, daß die Zahl der Nachkommastellen immer kleiner als die Feldlänge ist. Die restlichen 14 Byte sind für interne Zwecke reserviert.

Jedes definierte Feld der Datenstruktur besitzt einen eigenen 32-Byte-Satz im Kopf der DBF-Datei. Das Ende des Bereiches mit den Felddefinitionen wird durch das Zeichen 0DH markiert.

Die DBF-Datensätze

Die eigentlichen Datensätze werden wie bei dBASE hinter die Felddefinition angehängt. Die Satzlänge richtet sich nach der Länge der jeweiligen Felder und wird im Header der Datei geführt. Der Wert ist immer um 1 Byte größer als die Summe der Feldlängen, da vor jedem Satz ein Byte für die Löschmarkierung reserviert wird. Die Daten werden im reinen ASCII-Format und ohne Trennzeichen gespeichert. Ein ungelöschter Satz enthält im ersten Byte immer ein Leerzeichen (Code 20H). Als gelöscht markierte Sätze enthal-

ten das Zeichen * (Code 2AH) im ersten Byte. Damit wird der Import und Export von ASCII-Daten (z. B. mit der SDF-Option) recht einfach.

Wird ein neuer Satz mit dem *APPEND BLANK*-Befehl an die Datei angehängt, füllt FoxPro diesen Satz mit Leerzeichen auf. Das erste Byte im Satz dient zur Markierung gelöschter Daten. Wird ein Leerzeichen eingetragen, kann der Satz nicht gelöscht werden. Erst wenn das erste Byte das Zeichen '*' enthält, wird der Satz beim nächsten PACK-Befehl aus der DBF-Datei entfernt. Dies führt zu sehr schnellen *DELETE*-Operationen und ermöglicht sogar ein *UNDELETE* ohne großen Aufwand.

Der Abschluß des gültigen Datenbereiches wird durch das Zeichen 1AH markiert. Diese *EOF*-Marke wird nicht durch DOS, sondern durch FoxPro verwaltet.

Die Struktur einer FoxBase+ DBT-Datei (Memodatei)

In FoxBase+ werden Memodaten in DBT-Dateien gespeichert. Deren Struktur wurde von dBASE III abgeleitet, und die Speicherung ist nicht so effektiv wie bei den (im nächsten Absatz beschriebenen) FoxPro-Memodateien (FPT). In der DBF-Datei findet sich im betreffenden Feld des Datensatzes nur ein Zeiger auf den Block der Memodatei (Bild 4.2)

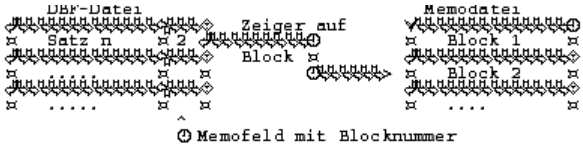


Abbildung 4.2 Zeiger im Memofeld auf den Memoblock

Der Zeiger im Memofeld der DBF-Datei ist für den Anwender unsichtbar. Falls kein Textblock für diesen Satz vorliegt, besitzt der Satz einen Leereintrag im Memofeld.

Die FoxBase+ Memodatei wird in Blöcke zu je 512 Byte unterteilt. Der erste Block enthält in den ersten 2 Byte die Nummer des nächsten freien Blocks in der Memodatei (Bild 4.3).

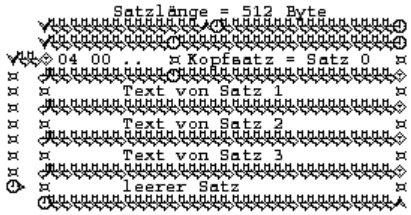


Abbildung 4.3 Satzaufbau einer FoxBase+ Memodatei

Der Offset auf den Anfang des freien Blocks wird zu:

```
Offset = Blocknummer * 512
```

berechnet. Beachten Sie, daß die Blocknummer im Intelformat (Low Byte first) vorliegt. Soll ein neuer Text zu einem Memofeld gespeichert werden, liest FoxBase+ den Kopfzeiger der Memodatei und trägt den Wert in das entsprechende Memofeld der DBF-Datei ein. Dann wird der Text einfach an das Ende der bisherigen Memodatei angefügt. Ist der Text länger als 512 Byte, werden mehrere 512-Byte-Blöcke benutzt. Anschließend wird die Nummer des nächsten freien Blocks berechnet und im Kopf der Memodatei gespeichert.

Bei Änderungen in MEMO-Texten werden diese einfach an das Ende der Memodatei angefügt und der neue Zeiger im Memofeld der DBF-Datei eingetragen. Damit bleibt der alte Text in der Memodatei erhalten. Dies bläht die Memodateien stark auf, falls häufig Texte geändert werden. Erst ein COPY-Kommando sorgt dafür, daß die unbenutzten Texte aus der Memodatei entfernt werden.

Ist ein Text länger als 512 Byte, belegt dBASE III einfach einen weiteren 512-Byte-Satz. Dies wird solange wiederholt, bis alle Bytes des Textes gespeichert sind. In der DBF-Datei findet sich dann der Zeiger auf den Anfang des Textblocks. Das Ende des Textes wird durch das Zeichen 1AH abgeschlossen. Die folgenden Bytes bis zum Ende des 512-Byte-Blocks sind damit undefiniert.

Die Struktur der FoxPro FPT-Dateien (Objekt- und Memodateien)

Ab FoxPro 2.0 existieren FPT-Dateien zur Aufnahme von Memotexten und Binärdaten (Bilder). FoxPro 2.5 erlaubt zusätzlich, Objekte in FPT-Dateien zu speichern. Die Daten können dabei das Objekt selbst oder lediglich eine Referenz auf das zugehörige Objekt (Anwendung mit Daten) umfassen.

Die FPT-Dateien besitzen in FoxPro 2.x einen einheitlichen Aufbau und bestehen aus einem Dateikopf und *n* Datensätzen. Die Länge des Dateikopfes beträgt stets 512 Byte. Der Rest der Datei wird in Blöcke fester Länge eingeteilt. Deren Länge wird auf 512 Byte voreingestellt, läßt sich aber mit dem Befehl *SET BLOCKSIZE* variieren.

Der Kopf einer FPT-Datei

Der Dateikopf besitzt den Aufbau gemäß Tabelle 4.5.

Offset	Bytes	Bemerkung
00H	4	Position freier Block
04H	2	unbelegt
06H	2	Blockgröße in Byte

Tabelle 4.5 Kopf einer FoxPro FPT-Datei

Offset	Bytes	Bemerkung
08H ... 1FFH	504	unbelegt

Tabelle 4.5 Kopf einer FoxPro FPT-Datei

Die ersten 4 Byte enthalten einen Zeiger mit dem Offset auf den ersten freien Block der Memodatei. Der Zeiger ist dabei nicht im Intel-, sondern in der Motorolaformat (Low Byte last) gespeichert. Der Offset 2000H wird dann als Bytefolge 00 00 20 00 gespeichert. Die zwei Folgebytes ab Offset 04H sind unbelegt. Ab Offset 06H folgt die Angabe über die Blockgröße in Byte. Auch dieser Wert ist als Hexadezimalzahl von links nach rechts (02 00 entspricht 200H) zu lesen. Dieser Wert definiert dann die Länge der folgenden Datenblöcke und wird über den FoxPro-Befehl *SET BLOCKSIZE* eingestellt. Die restlichen Byte des ersten Blocks sind unbelegt.

Der Datenbereich der FPT-Datei

Ab Offset 512 (200H) folgen die eigentlichen Datenblöcke mit den Memotexten oder Objektdaten. Die Datenblöcke besitzen eine feste Länge, die im Kopf der FPT-Datei ab Offset 06H festgehalten wird. Jeder Block muß an einer geraden Adresse beginnen. Die Position des Blockanfangs läßt sich durch Multiplikation der Blocknummer (aus der DBF-Datei) mit der Blockgröße (aus dem Kopf der FPT-Datei) errechnen:

```
Offset Block = Blocknummer * Blockgröße
```

Falls kein Eintrag für einen Satz vorliegt, besitzt die DBF-Datei für diesen Satz einen Leereintrag im Memo- oder Objektfeld. Andernfalls steht dort die 10 Byte lange Blocknummer, die als ASCII-Zahl zu interpretieren ist. Bild 4.4 zeigt Auszüge aus einer FPT-Datei als Hexdump.

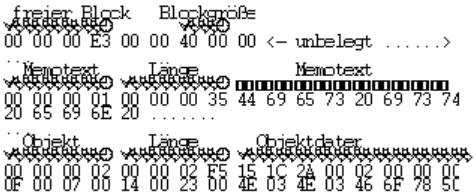


Abbildung 4.4 Auszug aus einer FPT-Datei

Die Daten (Memofelder, Bilder, Objekte etc.) werden satzweise auf die einzelnen Datenblöcke abgebildet. Ein Satz beginnt immer an einem Blockanfang und damit an einer geraden Adresse. Der Satz kann durchaus mehrere Blöcke belegen. Wichtig ist lediglich, daß der Folgesatz am Beginn des nächsten Blocks beginnt. Die Bytes ab dem Satzende bis

zum Blockende bleiben in diesem Fall undefiniert. Ein Datensatz besitzt die Struktur aus Tabelle 4.6.

Offset	Bytes	Bemerkung
00H	4	Satzkennung 0: Feldtyp Bild 1: Feldtyp Memo 2: Feldtyp Objekt
04H	4	Länge Memofeld in Byte
08H..n	n	Memotext mit n Byte

Tabelle 4.6 Struktur eines Datenblocks

Die ersten 4 Byte enthalten eine Kennung für den Typ der Daten. Hierbei wird zwischen Text (Memo) und Binärdaten (Bild oder Objekt) unterschieden. Bei Texten (Memofeldern) wird die Kennung 1 eingetragen. Beachten Sie dabei, daß die Werte im Motorolaformat als Hexzahlen (z. B. 00 00 00 01 für die Kennung 1) vorgegeben werden. Im Datenbereich finden sich dann ASCII- (oder ANSI-) Texte. Wenn die Kennung 0 auftritt (00 00 00 00), enthält der Datenbereich die Binärdaten eines Bildes. Dies ist nach meinen Erfahrungen aber nur auf dem Macintosh möglich, da FoxPro auf DOS-Maschinen den Feldtyp Objekt verwendet. Bei Objekten wird aber nach meinen Informationen die Kennung 2 (00 00 00 02) verwendet. Hierbei kann es sich um ein Bild, einen Klang oder ein eingebettetes Objekt handeln. Die Struktur des Datenbereiches richtet sich nach dem eingebundenen Anwendungsobjekt (Excel Tabelle, Paintbrush Bild etc.), ist mir aber nicht bekannt.

Die Länge des Datenbereiches wird ab Offset 04H als 4-Byte-Zeiger im Kopf des Satzes geführt. Auch dieser Wert ist von links nach rechts als Hexadezimalzahl zu lesen.

Ab Offset 08H schließt sich der Datenbereich mit einer variablen Länge an. Der Datenbereich kann dabei mehrere Blöcke der FPT-Datei umfassen. Im letzten Block bleibt der Bereich zwischen dem Ende der Daten und dem Blockende undefiniert, da der Folgesatz auf einer Blockgrenze liegen muß. In einigen Fällen trägt FoxPro bei Memofeldern hier die Kennung Memofeld ein.

Die Struktur unkomprimierter IDX-Indexdateien

FoxPro unterstützt verschiedene Indexdateien:

- unkomprimierte IDX-Dateien
- komprimierte IDX-Dateien
- Multiindex CDX-Dateien

Nachfolgend werden die Strukturen der unkomprimierten IDX-Dateien behandelt.

Unkomprimierte Indexdateien (IDX) werden als Baum aufgebaut. Ausgehend von einem Startknoten werden die Schlüssel über verschiedene Indexknoten verteilt (Bild 4.5).

n. gef. H:\Galileo\Born\45

Abbildung 4.5 Baumstruktur einer IDX-Datei

Beginnend ab der Wurzel (Startknoten) werden die Indexeinträge auf die Knoten (Indexknoten) des Baumes aufgeteilt. Die Blätter des Baumes (Endknoten) enthalten dann die Verweise auf die eigentlichen Datensätze. Um einen Ausdruck innerhalb des Baumes zu suchen, muß der Baum vom Startknoten bis zum Endknoten einen eindeutigen Pfad aufweisen. Die Einträge des Knotens werden dann sukzessive mit dem Suchbegriff verglichen. Ist der Suchschlüssel kleiner als der Schlüssel im Knoten (z. B. Suchbegriff = F, Eintrag = G), muß in den Teilbaum des nächst tieferen Knotens verzweigt werden. Wird zum Beispiel der Begriff *K* im Baum gesucht, sind folgende Operationen durchzuführen:

- Vergleiche den Begriff *K* mit dem ersten Eintrag *F* im Startknoten.
- Da der Suchbegriff größer als der Eintrag ist, muß der nächste Eintrag analysiert werden.
- Auch dieser Eintrag ist kleiner als der Suchbegriff.
- Da kein weiterer Eintrag mehr vorliegt, ist der Suchbegriff nicht im Baum vorhanden.

Wird dagegen der Buchstabe *C* im Baum gesucht, ergibt der erste Vergleich, daß der Suchbegriff kleiner als der Eintrag ist. Damit wird zum nächst tieferen Knoten mit dem Teilbaum verzweigt. Dort ergibt bereits der erste Vergleich einen Treffer, da der Eintrag gleich dem Suchbegriff ist. Nun wird zum nächst tieferen Knoten (Endknoten) verzweigt. Beim dritten Zugriff auf die Daten ergibt sich, daß der Suchbegriff gleich dem Eintrag ist. Damit liegt die Satznummer der Datenbank vor.

Die Suche über den Indexbaum erlaubt einen schnellen Zugriff auf die Daten der Datenbank. Zur Optimierung der Suchvorgänge im Baum enthält jeder Knoten noch zwei Zeiger auf die direkten Nachbarknoten der gleichen Ebene. Damit muß bei der Suche in Indexbereichen nicht zu den jeweiligen Verzweigungsstellen des Baumes zurückgegangen werden. Falls der benachbarte Knoten nicht existiert, wird der Zeiger mit dem Wert -1 (FF FF FF FF) belegt.

Nach diesen Vorüberlegungen möchte ich die Struktur der Indexdatei erläutern. Diese besteht aus einem Kopfsatz und beliebig vielen Knoten. Der Kopfsatz und die einzelnen Knoten besitzen immer eine Länge von 512 Byte.

Der Kopfsatz der IDX-Datei

Der Kopfsatz einer unkomprimierten IDX-Datei enthält alle Informationen über den Startknoten, die aktuelle Dateigröße, die Länge des Schlüssels etc. Tabelle 4.7 enthält die Struktur des Kopfsatzes.

Offset	Bytes	Bemerkungen
0	4	Zeiger auf den Startknoten
4	4	Zeiger auf die Liste der freien Knoten (-1 -> keine freien Knoten)
8	4	Zeiger auf das Dateiende
12	2	Schlüssellänge in Byte
14	1	Indexoptionen 1: eindeutiger Index 8: Index mit einer FOR-Klausel
15	1	Indexkennung (für spätere Verwendung)
16	220	Schlüsselausdruck als Zeichenkette
236	220	FOR-Ausdruck als Zeichenkette
456	56	unbelegt

Tabelle 4.7 Aufbau des IDX-Kopfsatzes

Die ersten 4 Byte enthalten einen Zeiger (im Intelformat) auf den Startknoten. Der Wert definiert den Offset ab dem Dateianfang auf den Anfang des Startknotens und muß ein Vielfaches von 512 Byte sein.

Beim Bearbeiten der Indexdatei werden neue Einträge in Knoten hinzugefügt und bestehende Einträge gelöscht. Daher kann es vorkommen, daß einzelne Knoten keine Einträge aufweisen. Ab Offset 4 findet sich deshalb ein Zeiger auf die Liste der freien Indexknoten. Der Zeiger ist als vorzeichenlose Zahl im Intelformat zu interpretieren. Existiert keine Liste mit leeren Indexknoten, wird der Zeiger mit -1 (FF FF FF FF) belegt.

FoxPro führt die Größe der Indexdatei im Kopfsatz ab Offset 8. Das Wort ab Offset 12 (0CH) definiert die Länge des Schlüsselbegriffs in Byte.

FoxPro kann eine Indexdatei über einen eindeutigen Schlüssel aufbauen. Alternativ besteht die Möglichkeit, Suchbereiche über eine FOR-Klausel anzugeben. Diese beiden Suchverfahren benutzen unterschiedliche Schlüsselbegriffe. Deshalb enthält das Byte ab Offset 14 (0EH) einen Eintrag für die betreffende Indexoption. Der Wert 1 signalisiert, daß die Indexdatei über einen eindeutigen Schlüssel aufgebaut wurde. Mit dem Wert 9 wurde der Index über eine FOR-Klausel erzeugt.

Das Byte ab Offset 14 (0FH) ist für die Indexkennung vorgesehen, wird bei unkomprimierten IDX-Dateien aber nicht benutzt. Der Schlüsselausdruck für eindeutige Indizes wird ab Offset 16 (10H) gespeichert. FoxPro wandelt dabei numerische Indexfelder in

Zeichenketten. Dadurch lassen sich die gleichen Such- und Sortieralgorithmen benutzen. Numerische Werte werden in die IEEE-Zahl konvertiert, in das Motorolaformat umgesetzt und bei negativen Werten in den Betrag umgerechnet. Dieser Wert wird dann als Schlüssel benutzt. Der Schlüssel darf dabei bis zu 220 Zeichen umfassen, wobei die Länge im Kopf ab Offset 12 (0CH) gespeichert ist.

Die Datei kann auch über einen FOR-Ausdruck aufgebaut werden. In diesem Fall ist der FOR-Ausdruck ab Offset 236 mit einer maximalen Länge von 220 Zeichen gespeichert.

Die restlichen Byte ab Offset 456 bis 511 sind unbenutzt.

Der Aufbau der Knotensätze

An den Kopfsatz schließen sich die einzelnen Knoten mit einer Länge von jeweils 512 Byte an. Jeder dieser Datensätze enthält ein Attribut, die Zahl der gespeicherten Schlüssel, die Zeiger auf die Nachbarknoten und die eigentlichen Indexeinträge. Tabelle 4.8 gibt die genaue Struktur eines Indexknotens wieder.

Offset	Bytesv	Bemerkungen
0	2	Attribut des Knotens 0: Indexknoten 1: Startknoten 2: Endknoten
2	2	Zahl der eingetragenen Schlüssel
4	4	Zeiger auf den linken Knoten der gleichen Ebene (-1 -> kein Knoten)
8	4	Zeiger auf den rechten Knoten der gleichen Ebene (-1 -> kein Knoten)
12	500	Bereich mit den Schlüsseleinträgen jeder Eintrag besteht aus dem Schlüssel und einem 4-Byte-Zeiger

Tabelle 4.8 Aufbau eines Indexknotens in unkomprimierten IDX-Dateien

Die ersten beiden Byte enthalten das Attribut des jeweiligen Knotens. Dabei darf der Wert auch aus der Summe der Einzelattribute (0: Indexknoten, 1: Startknoten, 2: Endknoten) gebildet werden. Die Zahl wird im Intelformat gespeichert (z. B. 03 00).

Ab Offset 2 findet sich die Anzahl der Schlüsseleinträge im Knoten. Der Wert wird ebenfalls im Intelformat gespeichert. Daran schließen sich die Zeiger auf die Nachbarknoten der gleichen Ebene an. Diese Zeiger umfassen jeweils 4 Byte und werden ebenfalls im Intelformat gesichert. Existiert der Nachbarknoten nicht, trägt FoxPro den Wert -1 ein (FFFF FFFFH). Der Zeiger auf den linken Knoten beginnt ab Offset 4 und der Zeiger auf den rechten Knoten ab Offset 8.

Die 500 Byte ab Offset 12 (0CH) bis zum Ende des Knotens nehmen die Indexeinträge auf. Jeder Eintrag besteht dabei aus dem eigentlichen Schlüsselbegriff, gefolgt von einem 4-Byte-Zeiger (Bild 4.8).

n. gef. H:\Galileo\Born\46

Abbildung 4.6 Aufbau eines Schlüsseleintrags

Die Länge des Schlüsseleintrags steht im Kopfsatz. An diesen Schlüsseleintrag schließt sich ein 4-Byte-Zeiger im Motorolaformat (High Byte first) an. Die Interpretation dieses Zeigers hängt vom Attribut des Knotens ab. Handelt es sich um einen Endknoten (Attribut 2 oder 3), dann enthält der Zeiger die Datensatznummer der DBF-Datei. Bei Start- und Indexknoten (Attribut 0 oder 1) definiert der Zeiger den Offset auf den unterlagerten Folgeknoten mit dem Teilbaum (siehe Bild 4.5). Die Kombination aus Schlüssel und Zeiger tritt *n*-mal innerhalb des Knotens auf. Der Wert n findet sich ab Offset 2 am Anfang des Knotens und muß zwischen 0 und 100 (1 Byte Schlüssel und 4 Byte Zeiger) liegen. Bei n = 0 liegt ein leerer Knoten vor. Die Liste der leeren Knoten wird über den Zeiger im Kopfsatz der Indexdatei verwaltet.

Die Struktur einer kompakten IDX-Indexdatei

FoxPro bietet die Möglichkeit, den Inhalt einer IDX-Datei in komprimierter Form zu speichern. Dies reduziert die Dateigröße und beschleunigt die Suchzugriffe. Beim Pakken werden doppelte Buchstaben im Index zusammengefaßt. Die gepackten IDX-Dateien besitzen allerdings eine etwas modifizierte Struktur. Diese Struktur wird auch bei Multiindexdateien (CDX) verwendet.

Der Kopfsatz einer gepackten IDX-Datei

Der Kopfsatz einer kompakten Indexdatei umfaßt 1024 Byte und wird gemäß Tabelle 4.9 strukturiert.

Offset	Bytes	Bemerkungen
0	4	Zeiger auf den Startknoten
4	4	Zeiger auf die Liste der freien Knoten (-1 -> keine freien Knoten)
8	4	reserviert zur internen Verwendung
12	2	Schlüssellänge in Byte
14	1	Indexoptionen 1: eindeutiger Index 8: Index mit einer FOR-Klausel 32: kompakte Indexdatei 64: Mehrfachindexdatei

Tabelle 4.9 Kopfsatz einer kompakten IDX-Datei

Offset	Bytes	Bemerkungen
15	1	Indexkennung
16 ... 501	485	reserviert zur internen Verwendung
502	2	Kennung für Sortierfolge 0: aufsteigend 1: absteigend
504	2	reserviert zur internen Verwendung
506	2	Gesamtlänge aller FOR-Ausdrücke
508	2	reserviert zur internen Verwendung
510	2	Gesamtlänge aller Schlüsselausdrücke
512	512	Menge alle Schlüsselausdrücke

Tabelle 4.9 Kopfsatz einer kompakten IDX-Datei

Die ersten 4 Byte definieren den Zeiger auf den Startknoten. Dieser Zeiger wird im Intelformat gespeichert. Der nächste 4-Byte-Zeiger ab Offset 4 verweist auf den Anfang der Liste der freien Indexknoten. Der Wert wird im Intelformat gespeichert und enthält die Zahl -1 (FFFF FFFFH), falls keine Liste existiert.

Die 4 Byte ab Offset 8 sind für interne Zwecke reserviert. Daran schließt sich ab Offset 12 (0CH) ein Wort im Intelformat mit der Länge des Schlüsselbegriffs an.

Die Indexoptionen werden im Byte ab Offset 14 (0EH) gespeichert. Bei kompakten Indexdateien wird zusätzlich die Option 32 (20H) definiert. Sobald dieses Bit gesetzt wird, erkennt FoxPro eine komprimierte Indexdatei. Der Wert 64 (40H) wird zur Markierung von CDX-Dateien verwendet.

Das Byte ab Offset 15 (0FH) enthält eine Indexkennung, deren Bedeutung mir aber nicht klar ist. Die Bytes ab Offset 16 bis 501 sind für interne Zwecke reserviert. Das Wort ab Offset 502 definiert die Reihenfolge der Indexsortierung:

```
0:aufsteigend
1:absteigend
```

und wird gemäß den Intel-Konventionen gespeichert.

Das Wort ab Offset 504 ist wieder für interne Zwecke reserviert. Ab Offset 506 folgt ein Wort im Intelformat mit der Länge aller FOR-Ausdrücke in der Indexdatei. Bei IDX-Dateien bezieht sich diese Länge auf einen FOR-Ausdruck. Das Wort ab Offset 508 ist wieder reserviert.

Ab Offset 510 folgt ein Wort im Intelformat mit der Länge aller Schlüsselausdrücke. Bei IDX-Dateien liegt nur ein Schlüssel vor. Ab Offset 512 folgt dann ein 512 Byte langer Be-

reich mit dem eigentlichen Indexausdruck. Bei Multiindexdateien können hier auch mehrere Schlüsselausdrücke gespeichert sein.

Der Aufbau der Knotensätze

An den Kopfsatz schließen sich die Indexknoten an. Jeder Knoten umfaßt 512 Byte. Allerdings besteht die Möglichkeit, die Indexdaten innerhalb des Indexknotens zu komprimieren. Die Indexknoten werden deshalb in:

- interne Indexknoten und
- externe Indexknoten

aufgeteilt. Die internen Indexknoten enthalten die Daten in unkomprimierter Form (Tabelle 4.10).

Offset	Bytes	Bemerkungen
0	2	Attribut des Knotens 0: Indexknoten 1: Startknoten 2: Endknoten
2	2	Zahl der eingetragenen Schlüssel
4	4	Zeiger auf den linken Knoten der gleichen Ebene (-1 -> kein Knoten)
8	4	Zeiger auf den rechten Knoten der gleichen Ebene (-1 -> kein Knoten)
12	500	Bereich mit den Schlüsseleinträgen; jeder Eintrag besteht aus dem Schlüssel und einem 4-Byte-Zeiger

Tabelle 4.10 Aufbau eines internen Indexknotens

Diese Struktur entspricht dem Aufbau der Knoten in unkomprimierten Indexdateien. Das erste Wort enthält wieder das Attribut des Indexknotens (0: Indexknoten, 1: Startknoten, 2: Endknoten). Daran schließt sich ein Wort im Intelformat mit der Anzahl der im Knoten gespeicherten Schlüssel an. Ab Offset 4 folgen die zwei 4-Byte-Zeiger auf den linken und rechten Knoten der gleichen Ebene. Nicht existierende Knoten werden mit -1 markiert. Alle Werte bis Offset 12 (0CH) werden im Intelformat gespeichert.

Ab Offset 12 (0CH) bis Offset 511 (1FFH) erstreckt sich der Bereich mit den Schlüsseleinträgen. Der Knoten enthält dabei immer den Schlüsselindex und die Datensatznummer. Der Eintrag kommt n mal vor.

Für die Knoten mit komprimierten Indexdaten wurde eine modifizierte Struktur (Tabelle 4.11) eingeführt.

Offset	Bytes	Bemerkungen
0	2	Attribut des Knotens 0: Indexknoten 1: Startknoten 2: Endknoten
2	2	Zahl der eingetragenen Schlüssel
4	4	Zeiger auf den linken Knoten der gleichen Ebene (-1 -> kein Knoten)
8	4	Zeiger auf den rechten Knoten der gleichen Ebene (-1 -> kein Knoten)
12	2	Größe des freien Speichers im Knoten
14	4	Maske für die Datensatznummer
18	1	Maske für die Anzahl der doppelten Zeichen
19	1	Maske für die Anzahl der nachfolgenden Zeichen
20	1	Anzahl der Bits für die Datensatznummer
21	1	Anzahl der Bits für doppelte Zeichen
22	1	Anzahl der Bits für nachfolgende Zeichen
23	1	Anzahl der Bytes für die Datensatznummer, doppelte Zeichen und nachfolgende Zeichen
24	488	Bereich mit den Schlüsseleinträgen und den Informationen

Tabelle 4.11 Aufbau eines externen Indexknotens

Bei externen Indexknoten werden die ersten 12 Byte wie bei umkomprimierten Indexdateien gespeichert. Ab Offset 12 (0CH) findet sich ein Wort im Intelformat, welches den freien Speicher im Knoten angibt. FoxPro verwaltet den Bereich mit den Indexeinträgen ab Offset 24 (18H) bis 511 (1FFH) auf folgende Weise:

- Zu Beginn dieses Bereiches werden die komprimierten Indexwerte eingetragen.
- Am Ende des Bereiches wird der eigentliche Index eingetragen.

Damit ergibt sich die Möglichkeit, nachträglich neue Indizes am Ende des Bereiches einzufügen. Ab Offset 14 beginnen die Masken für die komprimierten Daten. Die 4 Byte ab Offset 14 (0EH) definieren eine Binärmaske für die Datensatznummer. Diese Datensatznummer muß mit der Maske über AND verknüpft werden, um den Satz der Datenbank zu berechnen. Dadurch können die Datensätze mit weniger als 4 Byte gespeichert werden, was Speicherplatz spart.

Die Bytes ab Offset 18 und 19 definieren Masken für die komprimierten Zeichen des Index. Die einzelnen Bits sind mit diesen Masken per AND zu verknüpfen, um den ursprünglichen Indexwert zu berechnen.

Das Byte ab Offset 20 (14H) definiert, wie viele Bit innerhalb eines Eintrages zur Darstellung der Datensatznummer benutzt werden (z.B. 16 oder 32). Das Byte ab Offset 21 (15H) definiert die Zahl der Bits zur Darstellung doppelter Zeichen im Index. Ab Offset

22 (16H) findet sich dann noch die Zahl der Bits für nachfolgende Zeichen. Die Zahl der Bytes für die Datensatznummer, die Zahl der doppelten Zeichen und die Zahl der nachfolgenden Zeichen findet sich im Byte ab Offset 23. Mit dem Wert 3 bedeutet dies, daß jeder Eintrag im Bereich ab Offset 24 (18H) bis 511 (1FFH) drei Byte umfaßt. Diese drei Byte werden in Bitfolgen mit der Datensatznummer, der Zahl der doppelten Zeichen und der Zahl der Folgezeichen unterteilt. Wie allerdings die Komprimierung der Zeichen erfolgt, ist nicht dokumentiert.

Das Format der Mehrfachindexdateien (CDX)

FoxPro erlaubt die Erzeugung von Indexdateien, die mehrere Indizes gleichzeitig aufnehmen. Diese Indexdateien werden immer im komprimierten Format gespeichert. Der Aufbau ist identisch mit der Struktur der gepackten IDX-Dateien. Die Indexoption im Kopfsatz (Offset 14) enthält den Wert 64 für die Mehrfachindexdatei. Weiterhin zeigen die Endknoten auf den Indexschlüssel des Mehrfachindex. Für alle Indexschlüssel innerhalb der Indexdatei gibt es einen eigenen Indexbaum. Dieser besitzt den Aufbau einer kompakten IDX-Datei.

Die Informationen wurden den FoxPro-Programmierhandbüchern entnommen. Allerdings sind nicht alle Aspekte der Indexdateien dokumentiert.

Die Struktur einer FoxPro 1.0 Etikettendatei (LBX)

In FoxPro 1.0 werden Etikettendaten in LBX-Dateien gespeichert. Diese besitzen den Aufbau gemäß Tabelle 4.12.

Offset	Bytes	Bemerkungen
0	1	Version 03H für FoxPro 1.0 LBX-Datei
1	60	Kommentar als ASCII-String
61	2	Zeilenzahl im Etikett
63	2	Spaltennummer des linken Randes
65	2	Breite des Etiketts
67	2	Anzahl der Etiketten, die nebeneinander in einer Zeile stehen
69	2	Abstand zwischen Etiketten in Leerzeichen
71	2	Anzahl der Leerzeilen zwischen den Etiketten
73	2	Länge n des Etikett-Textes in Zeichen
75	n	Text für den Etikettausdruck, Zeilen durch 0DH voneinander getrennt

Tabelle 4.12 Aufbau einer FoxPro 1.0 LBX-Datei

Die Daten werden komplett im Intelformat gespeichert. Der Text mit den Etikettendaten findet sich ab Offset 75 als ASCII-String, wobei einzelne Zeilen durch das Zeichen 0DH

getrennt werden. Die Länge des ASCII-Strings wird im Wort ab Offset 73 angegeben. Die restlichen Einträge der Tabelle definieren das Ausgabeformat (Breite, Höhe, Spalten, etc.) der Etiketten.

Ab FoxPro 2.0 werden Etikettendaten in DBF-Dateien gespeichert. Anschließend generiert der Reportgenerator eine ausführbare Datei aus diesen Daten. Ähnliches gilt für Report- und Maskendateien. Deren Struktur wird in den FoxPro-Handbücher beschrieben.

Dateien in Visual FoxPro 3.0

In Visual FoxPro 3.0 wurden Datenbanken (DBC-Dateien) eingeführt. Hierbei handelt es sich letztlich um eine Datei im DBF-Format, die im ersten Byte die Signatur 30H aufweist. Innerhalb dieser DBF-Datei werden dann Felder mit Informationen über die Objekte der Datenbank (Tabellen, Berichte, Formulare etc.) gespeichert. Ähnliche Strukturen finden sich auch zur Speicherung der Berichts- und Formulardaten.

5 Datenaustausch über das SDF-Format

In den vorhergehenden Abschnitten wurde die interne Struktur der dBASE-Dateien beschrieben. Für die Systemprogrammierung ist es sicherlich interessant, direkt auf diese Strukturen zuzugreifen. In vielen anderen Fällen ist es jedoch ausreichend, wenn die Daten aus dBASE im ASCII-Format vorliegen oder im ASCII-Format gelesen werden – man denke nur an den Datenaustausch mit LOTUS 1-2-3, Multiplan, Word etc.

Über die Option SDF (*System Data Format*) lassen sich mit dBASE II, dBASE III, dBASE III+ und dBASE IV Datenbanken in ASCII-Dateien konvertieren. Der Befehl dazu besitzt folgende Aufrufsyntax:

```
COPY TO <File> [Fields] [FOR/WHILE] [SDF] [DELIMITED With ...]
```

Die in eckigen Klammern stehenden Parameter sind optional und müssen nicht unbedingt mitangegeben werden. Im Prinzip wird der COPY- Befehl dazu genutzt, eine mit USE geöffnete Datenbank in eine mit <*File*> bezeichnete Datei zu kopieren. Der Parameter *Fields* erlaubt es, nur bestimmte Felder der Datenbank für die Kopieroperation zu selektieren. Mit der Option FOR/WHILE lassen sich noch Bedingungen zur Selektion der Datensätze formulieren. Bis hierher wird jedoch nur eine normale dBASE-Datenbank erzeugt. Sobald jedoch die Option SDF erscheint, erzeugt dBASE eine ASCII-Datei aus den Daten der DBF-Datei. Die folgende Anweisung erzeugt die ASCII-Datei ASCII.TXT, wobei die Sätze mit Returnzeichen abgeschlossen werden:

```
COPY TO ASCII.TXT SDF
```

Die in diesem Beispiel verwendeten Daten müssen dazu die folgende Struktur haben (Bild 5.1):

```
 Name   _ Typ  _ Länge  _ Dezimalstellen
________|______|________|________________
 Feld1  _  C   _ 020    _
 Feld2  _  N   _ 010    _
 Feld3  _  N   _ 005    _   002
 Feld4  _  L   _ 001    _
```

Abbildung 5.1 Feldformat der dBASE-Datei

Die Felder werden in der ursprünglich definierten Länge angelegt. Ist ein Wert kürzer als die Feldlänge, ergänzt dBASE die restlichen Stellen mit Blanks. Die ASCII-Sätze haben dann beispielsweise folgende Struktur:

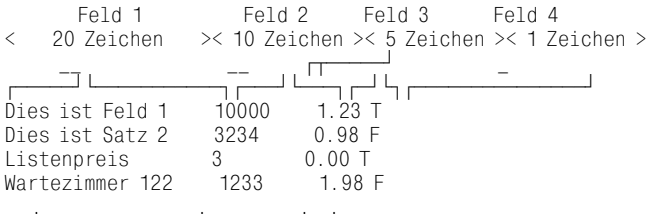


Abbildung 5.2 Ausgabeformat mit der SDF-Option

Damit lassen sich die einzelnen Felder mit einer fest definierten Länge satzweise lesen. Dieses Format eignet sich besonders zur Übernahme in Textverarbeitungsprogramme, da deren Ausgabe bereits eine Tabellenstruktur besitzt.

Die Option DELIMITED

Da die vorher beschriebene Formatierung für die Ein-/Ausgabe nicht immer gewünscht ist, besteht die Möglichkeit zur Definition von sogenannten *Delimitern* (Begrenzungszeichen) bei der Datenkonvertierung. Man wählt hierzu folgende Option:

```
DELIMITED WITH ...
```

Dieser Parameter weist dBASE an, zwischen den Feldern Trennzeichen auszugeben oder die definierten Zeichen als Begrenzungszeichen zu benutzen.

Die folgende Anweisung erzeugt eine ASCII-Datei HALLO.TXT, bei der die Felder der gerade benutzten DBF-Datei durch ein Komma getrennt sind:

```
COPY TO HALLO.TXT DELIMITED
```

Texte stehen zusätzlich in Anführungszeichen, wie Bild 5.3 zeigt:

```
7000,"Stuttgart","Waldstraße",13
8000,"M¤nchen","Station",5
5000,"Köln","Kalkstraße",10
```

Abbildung 5.3 Ausgabe der Daten über die Option Delimited

Neben normalen Trennzeichen lassen sich optional auch *Blanks*, *Semikolons* oder *Anführungszeichen* etc. als Begrenzungszeichen angeben. Die erste der folgenden zwei Anweisungen tauscht beispielsweise die Anführungszeichen in Bild 5.3 durch Hochkommata aus, bei der zweiten werden die Ausgabetexte mit Semikolons getrennt:

```
COPY TO HALLO.TXT  DELIMITED WITH '
COPY TO HALLO.TXT  DELIMITED WITH ;
```

Ein Blank als Delimiter ist allerdings problematisch, da nicht immer erkennbar ist, ob der Delimiter ein Feldende markiert oder Bestandteil eines Textfeldes sein soll. Dazu kommt, daß sich Zahlen aus Text- und numerischen Feldern nicht mehr unterscheiden lassen.

Umgekehrt gibt es auch einen Befehl, mit dem sich ASCII-Dateien in DBF-Dateien konvertieren lassen:

```
APPEND FROM <File> [FOR/WHILE][SDF][DELIMITED...]
```

Die ASCII-Datei muß dabei so formatiert vorliegen, daß die Daten in die dBASE-Felder passen.

Import/Export für Fremdformate

Ab dBASE III+ besteht die Möglichkeit, über folgende Dateiformate Daten direkt mit anderen Standardprodukten auszutauschen:

- DIF (Data Interchange Format), z. B. Visicalc
- SYLK (Symbolic Link Format), z. B. Multiplan
- WKS, z. B. LOTUS 1-2-3-Format

Der COPY-Befehl besitzt dabei folgendes allgemeine Format:

```
COPY TO <File>
   FIELDS <Feldliste>
   FOR/WHILE <Bedingung>
   TYPE <Typ>
```

Als Dateityp ist dann eine der Optionen DIF, SYLK oder WKS anzugeben. dBASE III+ erzeugt daraus die entsprechenden Dateien mit dem geforderten Format.

Über APPEND lassen sich in DBASE III auch Fremdformate einlesen. Der genaue Befehl lautet:

```
APPEND FROM <File> [FOR/WHILE] [SDF] [DELIMITED]
```

Um überhaupt in dBASE-Felder eingelesen werden zu können, muß eine Textdatei jedoch einige Anforderungen erfüllen:

- Die Struktur des Textes muß genau auf die Struktur der Datenbank passen.
- Jeder Satz ist mit einem CR/LF abzuschließen (*Carriage Return* bzw. *Line Feed*).

- Innerhalb einer Zeile müssen Anzahl, Reihenfolge und Länge der Felder mit der Datenbankdefinition übereinstimmen. Ist die Feldlänge des Textes kürzer als in der Datenbank definiert, funktioniert die Übernahme zwar noch, bei längeren Textfeldern schneidet dBASE den rechten Textrand jedoch einfach ab.
- Numerische Felder werden in der Datenbank mit 0 belegt, falls kein Wert verfügbar ist. Dadurch ergeben sich vielfältige Möglichkeiten zum Datenaustausch mit Fremdprogrammen.
  Weitere Hinweise hierzu finden sich in den Benutzerhandbüchern der jeweiligen dBASE-Versionen.

6 Der Aufbau einer CSV-Datei

Viele Programme erlauben den Austausch von Daten in Form von ASCII-Texten, wobei die Werte als »Comma Separated Values« (CSV) dargestellt werden.

Der Aufbau dieser Dateien folgt einem einfachen Schema:

- Die Datei wird als ASCII-String zeilenweise aufgebaut.
- Die einzelnen Werte werden durch Kommas separiert.
- Texte werden zusätzlich in Anführungszeichen (») eingerahmt.

Weiterhin stehen noch drei Kopfzeilen zur Verfügung, die die Feldnamen und die Datentypen definieren. Nachfolgend wird ein kurzes Beispiel aus der Software »Timeline« (Symantec) gezeigt, welches die Daten im CSV-Format enthält:

```
-110,5
-120,"Person","Preis","Vorgang","Einheiten","Ausgleich"
-130,1,2,6,6,5
-900,"Huber",150,"Resource","DM pro Std","Nein"
-900,"Meier",100,"Resource","DM pro Std","Ja"
-900,"Müller",250,"Resource","DM pro Std","Nein"
-900,"Schmitt",150,"Resource","DM pro Std","Nein"
```

Der Aufbau ist recht einfach: In der ersten Spalte wird eine negative Zahl definiert, die (spezifisch für Timeline) die nachfolgenden Daten des Satzes beschreibt. Der Wert -900 bezieht sich auf Nutzdaten, während kleinere Werte Definitionen markieren. Die erste Zeile mit der Zahl 5 definiert, daß alle folgenden Zeilen fünf Spalten aufweisen. In der zweiten Zeile werden die Namen der zu exportierenden Datenfelder (Spalten) angegeben. Die Namen werden dabei in Anführungszeichen eingeschlossen und durch Kommas separiert. Die dritte Zeile enthält 5 Zahlen, die den Datentyp der jeweiligen Spalte definieren.

Hierbei gilt:

Typ	Bemerkung
1	Text
2	Numerische Werte (positiv oder negativ) mit Dezimalpunkt
3	Ganze Zahlen ohne Dezimalstellen (Vorzeichen erlaubt)
4	Kardinalzahlen (kein Vorzeichen, keine Dezimalstellen)
5	Logischer Wert (Ja oder Nein)
6	Aufzählung
7,9	Datumsfeld (Start-Datum)

Tabelle 6.1 Datentypen der Spalten

Typ	Bemerkung
8,10	Datumsfeld (Ende-Datum)

Tabelle 6.1 Datentypen der Spalten

Auch hier handelt es sich um eine spezifische Darstellung von Timeline. Ab der vierten Zeile folgen die Datensätze (markiert durch die Zahl -900) in der ersten Spalte. Alle Werte werden dabei durch Kommas getrennt. Texte sind in Anführungszeichen (») einzuschließen. Falls im Text ein Anführungszeichen vorkommt, ist dieses Zeichen zu verdoppeln. Die Reihenfolge der Daten entspricht den Feldnamen im Kopf der Datei. Innerhalb eines Datumsfeldes können dabei als Ausnahme Kommas (z. B. 1988,12,31,9,0 = 31.12.1988, 9:00) auftreten.

Bei alphanumerischen Zeichen kann die festgelegte Feldbreite überschritten werden. Dann wird der Text rechts abgeschnitten. Ist der Text kürzer als die definierte Spaltenbreite, wird dies ignoriert. Der Text wird in Anführungszeichen eingeschlossen und durch Kommas getrennt. Es werden keine Stellen mit Leerzeichen aufgefüllt. Boolesche Felder enthalten die Einträge »Ja« oder »Nein« und werden durch die Spaltenbreite nicht beeinflußt. Daten in numerischen Feldern werden weder abgeschnitten noch aufgefüllt. Vielmehr wird der komplette Wert ausgegeben.

Beachten Sie aber, daß diese Definitionen abhängig von der verwendeten Software sind. Die Daten können für andere CSV-Dateien abweichend kodiert sein.

9 Das LOTUS 1-2-3-PIC-Format

LOTUS-Graphiken lassen sich in PIC-Dateien speichern und später mit PRINTGRAPH ausgeben oder durch Fremdprogramme importieren. LOTUS legt in den PIC-Dateien alle graphischen Informationen als Vektoren ab. Zusätzlich finden sich in der Datei Texte und Informationen über die zu benutzenden Fonts (incl. Fontgröße). Bild 9.1 gibt den Inhalt einer PIC-Datei als Hexdump wieder.

```
┌──────────── Header ──────────────────────────
01 00 00 00 01 00 08 00-44 00 00 00 00 0C 7F 09
   ┌─ Beginn des ersten Records
06 A7 01 B0 A0 01 90 08-20 A2 0B B8 08 20 A2 0B
└┘ └──┘ └┘ └─────────┘ └─────────┘ └──┘

   Font Color   Move        Draw          ...
B8 01 2C A2 01 90 01 2C-A2 01 90 08 20 A7 00 AC
└──────┘ └─────────┘ └─────────┘ └──────┘

00 8C 00 78 A0 06 3F 09-06 A8 02 54 69 6C 65 7A
└─────────┘ └─────────┘ └──┘ T i t e l

                      1. Textrecord
65 69 6C 65 20 31 00 A7-01 AC 00 46 00 3C A0 06
 e i l e    1 └┘ └──┘ └─────────┘ └──┘
3F 00 78 A8 02 54 65 78-74 20 58 2D 41 63 68 73
└──────┘ └──┘ T e x t   X ü A c h s
65 00 A0 00 00 04 A6 A8-12 54 65 78 74 20 59 2D
 e └┘ └─────────┘ └──┘ T e x t   Y -
41 63 68 73 65 00 A0 06-3F 08 3E A8 04 54 69 74
 A c h s e└┘ └─────────┘ └──┘ T i t
...
...
31 30 00 A0 04 F2 01 2C-A2 04 F2 01 0E A0 04 F2
 1 0 └┘ └─────────┘ └─────────┘ └──────┘
...
00 1E A0 05 D2 00 1E A8-01 4C 65 67 65 6E 64 65
└──┘ └─────────┘ └──┘ L e g e n d e
20 41 00 B0 60
└──────┘ └┘ └┘ EOF
```

Abbildung 9.1 Speicherdump einer PIC-Datei

Eine PIC-Datei besteht aus einem 17 Byte langen Header, wobei in den ersten 4 Bytes immer die Signatur 01 00 00 00 steht. Die Bedeutung der restlichen Bytes ist unbekannt, aber die in Bild 9.1 auftretende Bytefolge erscheint in allen PIC-Dateien.

Aufbau der PIC-Records

An den Header schließen sich mehrere Records unterschiedlicher Länge an. Der Recordtyp wird durch das erste Byte (Opcode) spezifiziert. Tabelle 9.1 enthält eine Aufstellung möglicher Recordtypen.

Typ	Bedeutung
30H	FILL (x1,y1) .. (xn,yn) zeichnet ein gefülltes Polygon.
60H-6FH	EOF-Marke, steht am Ende der Datei (1 Byte Länge)
A0H	MOVE x,y (5 Byte Länge)
A2H	DRAW x,y (5 Byte Länge)
A7H	FONT n (2 Byte Länge)
A8H	TEXT als ASCIIZ-String mit variabler Länge. In Byte 2 steht ein Subtyp, der die Lage des Textes spezifiziert. <A8><Subtyp><ASCIIZ-String>
ACH	SIZE n,m (5 Byte Länge)
B0H-BFH	COLOR Im Byte ist die Farbe in den unteren 4 Bits codiert.
D0H	FILLO (x1,y1) .. (xn,yn)

Tabelle 9.1 Records in LOTUS-PIC-Files

Die Records innerhalb der PIC-Datei besitzen dabei folgenden Aufbau:

FILL (x1,y1)...(xn,yn) (Opcode 30H)

Der Befehl FILL zeichnet ein gefülltes Vieleck (Polygon) in der aktuell gesetzten Farbe. Das Vieleck hat dabei keinen Rand. Es dürfen beliebig viele Koordinatenpaare (x,y) angegeben werden. Die Koordinaten werden dabei jeweils als 16-Bit-Wertepaare (x,y) dargestellt. Hinter dem Opcode steht ein Byte mit der Anzahl der Koordinatenpaare (Zählung beginnt bei 0).

END OF FILE (Codes 60H bis 6FH)

Das Ende der PIC-Datei wird in einem Byte durch den EOF-Code signalisiert. Als EOF-Code sind die Zahlen 60H bis 6FH erlaubt, wobei meist 60H verwendet wird.

MOVE (X,Y) (Opcode A0H)

Dieser Record besteht aus 5 Byte und enthält den Opcode A0H. Daran schließen sich zwei 16-Bit-Zahlen an, die die X- und Y-Koordinate für den neuen Punkt angeben. Der »Cursor« wird zum entsprechenden Punkt verschoben, ohne daß eine Linie gezeichnet wird.

DRAW (X,Y) (Opcode A2H)

Dieser Record besteht ebenfalls aus 5 Byte und enthält den Opcode A2H. Daran schließen sich zwei 16-Bit-Zahlen an, die die X- und Y-Koordinate für den neuen Punkt angeben. Der »Cursor« wird von der aktuellen Position zum entsprechenden Punkt verschoben, wobei eine Linie (Breite 1 Pixel) in der eingestellten Farbe gezogen wird. Der neue Punkt wird dann zur aktuellen Koordinate.

FONT n (Opcode A7H)

Dieser Record umfaßt 2 Byte und definiert den zu verwendenden Font. Das erste Byte enthält dabei den Opcode A7H, gefolgt von einem Byte für die Fontdefinition. Dabei gilt die Kodierung:

0 = Schriftart 1
1 = Schriftart 2

Weitere Varianten sind zur Zeit nicht realisiert.

TEXT xxxx (Opcode A8H)

Dieser Record beginnt mit dem Opcode A8H und besitzt eine variable Länge. Auf den Opcode folgt ein Byte, welches die Ausrichtung des Textes definiert. Die oberen 4 Bits spezifizieren dabei die Ausgaberichtung (D), während in den unteren 4 Bits die Position (P) auftritt. Für diese Bits gelten die in Tabelle 9.2 angegebenen Optionen.

Wert	Bedeutung
--	Bits 4-7 (Direction)
0	waagerecht von links nach rechts
1	senkrecht von oben nach unten
2	waagerecht von rechts nach links
3	senkrecht von unten nach oben
--	Bits 0-3 (Position)
0	Mittelpunkt des Zeichenrechtecks
1	Mittelpunkt der linken Seite
2	Mittelpunkt der Oberseite
3	Mittelpunkt der rechten Seite
4	Mittelpunkt der Unterseite
5	linke obere Ecke
6	rechte obere Ecke
7	linke untere Ecke
8	rechte untere Ecke

Tabelle 9.2 Kodierung Textausrichtung

Zur Positionierung des Textes ist folgendes anzumerken: Der Text wird an der aktuellen Cursorposition ausgegeben. Dabei läßt sich annehmen, daß der Text von einem Rechteck eingerahmt wird. Das Rechteck kann nun relativ zum aktuellen Punkt positioniert werden. Damit verschiebt sich auch der Text relativ zum aktuellen Koordinatenpunkt. Mit dem Code 0 wird der Text so positioniert, daß die Mitte auf dem aktuellen Koordinatenpunkt liegt.

Auf den Opcode A8H und das Byte mit den Richtungsbits folgt der eigentliche Text. Dieser wird dabei als ASCIIZ-String gespeichert, d.h. das letzte Byte wird mit dem Code 00H abgeschlossen. Der Text darf dabei nicht länger als 64 Kbyte sein, was in der Praxis wohl auch nicht vorkommt. Der Text wird dann an der aktuellen Position in der angegebenen Ausrichtung im aktuell eingestellten Zeichenfont ausgegeben. Auch die eingestellte Farbe beeinflußt die Textausgabe.

SIZE n,m (Opcode ACH)

Dieser Record bestimmt die Größe eines Zeichens. Auf den Opcode ACH folgen zwei 16-Bit-Zahlen, die die Abmessungen eines Rechtecks mit den Koordinaten (0,0) und (n,m) angeben. In den Zeichenfonts ist die Größe der einzelnen Zeichen fest definiert. Der SIZE-Befehl beeinflußt nun über die Parameter (n,m) die Skalierung der Zeichen, d.h. diese werden entsprechend vergrößert.

COLOR (Opcode BxH)

Mit diesem 1-Byte-Record wird die aktuelle Farbe eingestellt. Dabei wird die Farbe in den unteren 4 Bits kodiert. Damit sind 16 Farben möglich, die sich in den Opcodes B0H bis BFH niederschlagen. Die tatsächliche Farbe wird dann bei der Ausgabe durch PRINTGRAPH umgerechnet.

FILLO (x1,y1)..(xn,yn) (Opcode D=H)

Dies ist wieder ein Record, der ein gefülltes Polygon erzeugt. Dabei wird das Polygon jedoch mit einem Rand in der aktuellen Farbe ausgezogen. Es lassen sich mehrere Koordinatenpaare (X,Y) als 16-Bit-Zahlen angeben. Die Zahl der Koordinaten wird im zweiten Byte hinter dem Opcode angegeben.

Dies sind die Records, die in der PIC-Datei auftreten können. Damit lassen sich recht einfache Zeichnungen erstellen. Alle Koordinatenangaben innerhalb der PIC-Datei erfolgen als 16-Bit-Zahlen. Die Angaben beziehen sich dabei auf ein virtuelles rechtwinkliges karthesisches Koordinatensystem mit 3200 Punkten in X-Richtung und 2311 Punkten in Y-Richtung. Die Anpassung an die Koordinaten des Ausgabegerätes erfolgt dann durch PRINTGRAPH. Der Nullpunkt des Koordinatensystems liegt in der linken unteren Ecke. Weiterhin sind die Werte der Koordinaten entgegen der üblichen Intel-Konvention gespeichert. Das High-Byte eines Koordinatenwertes wird dabei zuerst in der Datei gespeichert. Aus der Zahl 3F00H wird dann in der Datei die Bytefolge 3F 00H. (Normalerweise wird die Zahl als 00H 3FH gespeichert.)

10 LOTUS-Symphony-Format

Das Programm Symphony stellt eine Erweiterung des Tabellenkalkulationsprogramms LOTUS 1-2-3 dar und wird ebenfalls von der Firma LOTUS-Development vertrieben. Der Inhalt der Rechenblätter läßt sich, ähnlich wie bei LOTUS 1-2-3, einschließlich der Daten und Kalkulationsformeln in Dateien speichern. In Abhängigkeit von der jeweiligen Programmversion (1.1, 2.0) werden diese Daten in einem Dateiformat gespeichert, das sich sehr stark an die LOTUS-Vorgaben hält. Symphony benutzt zur Ablage der Daten und Kalkulationsformeln Binärdateien, wobei Texte aus den Rechenblättern im ASCII-Format dargestellt werden.

Dabei gilt der gleiche Satzaufbau wie bei LOTUS 1-2-3:

```
<Record Typ> <Record Länge> <Daten>
```

Der Aufbau wird zwischen den verschiedenen Versionen beibehalten. Die einzelnen Felder haben folgende Bedeutung:

- Das Feld *Record Typ* ist zwei Byte lang und enthält – wie der Name schon sagt – Informationen über den Recordtyp. Dieser bestimmt dann den Aufbau des nachfolgenden Datenfeldes. Bei Sätzen mit Daten wird dieser Recordtyp auch als Opcode für Rechenanweisungen oder zum Aufbau des Kalkulationsblatts interpretiert. Die Begriffe *Recordtyp* oder *Opcode* werden deshalb synonym behandelt. Beim Recordtyp-Feld wird im ersten Byte der Datei das niederwertige Byte (low byte) gespeichert. Die Opcodes variieren mit den Symphony-Versionen.
- Das *Längenfeld* umfaßt zwei Byte und spezifiziert die Länge des nachfolgenden Datenfeldes in Byte. Dabei wird das LSB ebenfalls zuerst abgespeichert.
- Das *Datenfeld* besitzt eine variable Länge, die durch den jeweiligen Feldtyp spezifiziert wird. In diesem Feld finden sich dann Werte, Berechnungsformeln, Definitionen für den Aufbau des Kalkulationsblatts etc.

Die Erweiterung der Datei ist abhängig von der Programmversion: entweder WKS (Symphony-Version 1.0) oder WK1 (Version 2.0). Ein Hexdump der WK1-Datei entfällt an dieser Stelle, da der Aufbau zum größten Teil mit der LOTUS-Datei übereinstimmt. Das gleiche gilt für die Abbildung der Datei in das Kalkulationsblatt. Zeilen und Spalten werden nur aus Benutzersicht mit Buchstaben und Nummern versehen; intern arbeitet LOTUS/Symphony mit Zeilen- und Spaltennummern mit 16-Bit-Breite. Jede Zellposition läßt sich dann eindeutig durch zwei Zahlen identifizieren. In der Datei wird die Numerierung ebenfalls verwendet.

Recordtypen in Symphony

Nachfolgend werden die in den WKS-Dateien vorkommenden Satzarten (*Opcodes*) mit ihren Datenstrukturen vorgestellt. Viele der Opcodes werden auch in LOTUS 1-2-3 benutzt. In späteren Versionen wurden allerdings neue Opcodes hinzugefügt. Jeder Opcode umfaßt zwei Bytes, wobei das niederwertige Byte (LSB) zuerst gespeichert wird.

BOF (Opcode 0000H)

Dieser Feldtyp markiert den Beginn einer gültigen WKS-/WK1-Datei. Der Record hat folgenden Aufbau:

Offset	Bytes	Bedeutung
00H	2	Opcode BOF = 0000H
02H	2	Länge = 0002
04H	2	Versionsnummer Dateiformat 0404H 1-2-3-WKS-Format in Version 1 0405H Symphony-Datei 0406H 1-2-3-Datei im WK1-Format ab Version 2.0 und Symphony 1.1

Tabelle 10.1 Symphony-Recordstruktur (Opcode 0000H)

Das Datenfeld umfaßt zwei Bytes, in denen der Versionscode des Dateiformats abgelegt ist. Neuere Symphony-Versionen setzen diese Numerierung fort.

EOF (Opcode 0001H)

Dieser Record bildet den Abschluß einer WKS- oder WK1-Datei. Der Satz besitzt folgenden Aufbau:

Offset	Bytes	Bedeutung
00H	2	Opcode EOF = 0001H
02H	2	Länge = 0000

Tabelle 10.2 Symphony-Recordstruktur (Opcode 0001H)

Der Satz ist nur 4 Byte lang, das Datenfeld bleibt unbesetzt.

CALC_MODE (Opcode 0002H)

In Symphony kann der Anwender festlegen, ob eine Neuberechnung der Ergebnisse automatisch nach jeder Eingabe (Standardeinstellung) oder erst auf manuelle Anforderung hin stattfindet. Der Berechnungsmodus wird in einem eigenen Record mit in der Datendatei gespeichert. Der Satz besitzt folgenden Aufbau:

Offset	Bytes	Bedeutung
00H	2	Opcode CALC_MODE = 0002H
02H	2	Länge = 0001
04H	2	Berechnungsmodus 00 = manuelle Neuberechnung FF = automatische Neuberechnung

Tabelle 10.3 Symphony-Recordstruktur (Opcode 0002H)

Standardmäßig enthält das Datenbyte den Wert FFH für eine automatische Neuberechnung nach jeder Eingabe.

CALC_ORDER (Opcode 0003H)

Bei jeder Berechnung der Ergebnisse läßt sich festlegen, in welcher Reihenfolge die Formeln in der Tabelle bearbeitet werden. Diese Einstellung wird ebenfalls in einem Satz mit dem Aufbau aus Tabelle 10.4 gesichert.

Offset	Bytes	Bedeutung
00H	2	Opcode CALC_ORDER Opcode = 0003H
02H	2	Länge = 0001
04H	2	00 = natürliche Reihenfolge 01 = spaltenweise Berechnung FF = zeilenweise Berechnung

Tabelle 10.4 Symphony-Recordstruktur (Opcode 0003H)

Bei der spaltenweisen Berechnung (Code 01) werden erst die Formeln der betreffenden Tabelle abgearbeitet. Anschließend beginnt Symphony mit der Berechung der Formeln der folgenden Spalte.

RANGE (Opcode 0006H)

Mit dem Satz wird in Symphony der Bereich (RANGE) der in der Datei zu speichernden Zellen spezifiziert. In der Regel wird das komplette Rechenblatt gesichert. Es gilt folgender Aufbau:

Offset	Bytes	Bedeutung
00H	2	Opcode RANGE = 0006H
02H	2	Länge = 0008H
04H	2	Start Spalte (Column)
06H	2	Start Zeile (Row)
08H	2	Ende Spalte (Column)

Tabelle 10.5 Symphony-Recordstruktur (Opcode 0006H)

Offset	Bytes	Bedeutung
0AH	2	Ende Zeile (Row)

Tabelle 10.5 Symphony-Recordstruktur (Opcode 0006H)

Im Datenfeld werden die obere linke Ecke (Zeile/Spalte) und die untere rechte Ecke abgelegt. Falls die Datei durch das *File Save*-Kommando angelegt wurde, finden sich alle Felder des Rechenblatts in der Datei. Mit *File Xtract* werden nur die Zellen des spezifizierten Ausschnitts abgespeichert, so daß sich lediglich die Ausschnittskoordinaten im Datenfeld befinden. Zu beachten ist, daß Symphony die Spalten- und Zeilenadressen als 16-Bit-Nummern ablegt. Das LSB wird dabei zuerst gespeichert. Bei der Abspeicherung der Zellen werden leere Felder am Ende einer Spalte oder Zeile nicht berücksichtigt. Falls in dem mit Range bezeichneten Bereich keine Daten vorhanden sind, setzt Symphony im Datenfeld den Wert der Startspalte auf -1. Der Satztyp 06H findet sich in der Regel direkt nach dem BOF-Record. Dies sollte auch beachtet werden, wenn WKS- oder WK1-Dateien durch Fremdprogramme angelegt werden.

COLUMN_WIDTH_1 (Opcode 0008H)

Mit diesem Satz wird in Symphony die Breite der Spalten des im nächsten Satz beschriebenen Fensters definiert. Es gilt folgender Satzaufbau:

Offset	Bytes	Bedeutung
00H	2	Opcode COLUMN_WIDTH_1 = 0008H
02H	2	Länge = 0003H
04H	2	Spaltennummer (16 Bit)
06H	1	Breite der Spalte

Tabelle 10.6 Symphony-Recordstruktur (Opcode 0008H)

Mit dem Record 08H lassen sich die Breiten verschiedener Spalten angeben. Im ersten Wort steht dabei die betreffende Spaltennummer (LSB zuerst). Das folgende Byte gibt die Breite der Spalte in Zeichen an.

BLANK (Opcode 000CH)

Normalerweise speichert Symphony leere Zellen nicht mit ab, so daß auch die Information geschützter oder formatierter Leerzellen beim Speichern verloren ginge. Um dies zu vermeiden, existiert die Satzart 0CH, die diese Zellen mitsichert. Es gilt folgende Struktur:

Offset	Bytes	Bedeutung
00H	2	Opcode BLANK = 000CH
02H	2	Länge = 0005H

Tabelle 10.7 Symphony-Recordstruktur (Opcode 000CH)Symphony-Recordstruktur (Opcode 000CH)

Offset	Bytes	Bedeutung
04H	1	Formatbyte
05H	2	Spaltennummer
07H	2	Zeilennummer

Tabelle 10.7 Symphony-Recordstruktur (Opcode 000CH)Symphony-Recordstruktur (Opcode 000CH)

Im ersten Datenbyte speichert Symphony die Kodierung für das Zellformat:

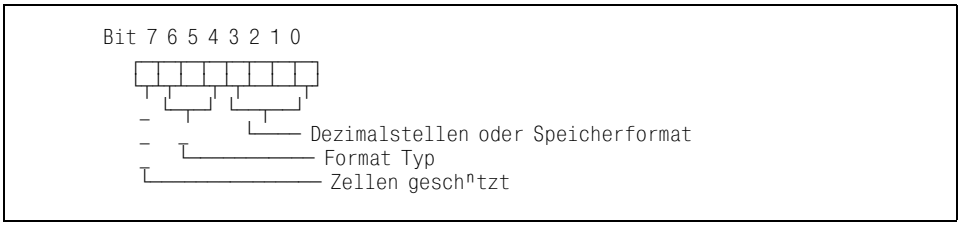


Abbildung 10.1 Zellformat in Symphony

Das oberste Bit gibt an, ob die Zellen innerhalb des Fensters für Schreibzugriffe gesperrt (*protected*) sind. Dabei gilt folgende Kodierung:

Bit 7	Funktion
1	1 protected
0	0 unprotected

Tabelle 10.8 Kodierung der Formate

In den Bits 4 bis 6 wird eine dreistellige Binärzahl mit dem Formattyp der Wertedarstellung geführt. Dabei gilt folgende Nomenklatur:

Bit 654	Format
000	Festkommaformat (fixed)
001	Exponentdarstellung (scientific notation)
010	Währungsdarstellung (currency)
011	Prozent
100	Komma
101	frei
110	frei
111	Spezialformat

Tabelle 10.9 Kodierung der Formate

Für die Formattypen 0 bis 6 spezifizieren die restlichen Bits 0 bis 3 die Zahl der Dezimalstellen (zwischen 0 und 15). Der Formattyp 7 repräsentiert ein spezielles Format, das über die Bits 0 bis 3 weiter spezifiziert wird.

Bit 3210	Format
0000	+/-
0001	generelles Format
0010	Datumsformat: Tag, Monat, Jahr
0011	Datumsformat: Tag, Monat
0100	Datumsformat: Monat, Jahr
0101	Textformate
0110	geschützt (*hidden*) nur in Symphony
0111	Datum + Stunden, Minuten, Sekunden nur in Symphony
1000	Datum + Stunden, Minuten nur in Symphony
1001	Datum, international 1 nur in Symphony
1010	Datum, international 2 nur in Symphony
1011	Zeit, international 1 nur in Symphony
1100	Zeit, international 2 nur in Symphony
1101	unbelegt
1110	unbelegt
1111	Standarddarstellung

Tabelle 10.10 Kodierung des Spezialformates in Symphony

Anschließend folgen zwei Worte mit den Zellkoordinaten. Der Satz wird nur angelegt, wenn geschützte Zellen vorhanden sind.

INTEGER (Opcode 000DH)

Direkt eingegebene Ganzzahlen (*Integer*) werden aus dem Kalkulationsblatt in die Datei übernommen. Der Record besitzt in Symphony folgende Struktur:

Offset	Bytes	Bedeutung
00H	2	Opcode INTEGER = 000DH
02H	2	Länge = 0007H
04H	1	Formatbyte
05H	2	Spaltennummer
07H	2	Zeilennummer
09H	2	Integerwert

Tabelle 10.11 Symphony-Recordstruktur (Opcode 000DH)

Der Satz enthält insgesamt 7 Datenbytes, wobei im ersten Byte das Format der Zahl abgelegt wird (die Kodierung ist in Tabelle 10.10 aufgeführt). In den folgenden Wörtern steht die Position der Zelle mit dem Integerwert. Daran schließt sich ein 16-Bit-Wort an, das dann den Integerwert enthält. Das oberste Bit gibt dabei an, ob der Wert positiv oder negativ (Bit = 1) ist. In einer Zelle läßt sich somit ein Wertebereich zwischen -32768 und +32767 abbilden.

NUMBER (Opcode 000EH)

Mit diesem Recordtyp sichert Symphony Fließkommazahlen. Der Record besitzt folgende Struktur:

Offset	Bytes	Bedeutung
00H	2	Opcode NUMBER = 000EH
02H	2	Länge = 000DH (13 Datenbyte)
04H	1	Formatbyte
05H	2	Spaltennummer
07H	2	Zeilennummer
09H	8	64 Bit IEEE-Fließkommazahl

Tabelle 10.12 Symphony-Recordstruktur (Opcode 000EH)

Der Satz umfaßt insgesamt 13 Datenbyte, wobei im ersten Byte das Zahlenformat abgelegt wird (die Kodierung ist in Tabelle 10.8/10.9 aufgeführt). In den folgenden Wörtern steht die Zellposition mit dem Fließkommawert. Daran schließen sich 8 Bytes an, in denen der Wert als 64 Bit-IEEE-Fließkommazahl gespeichert wird. Diese Darstellung entspricht der Kodierung des 8087-Formats, intern benutzt Symphony eine eigene Darstellung.

Offset	Bytes	Bedeutung
00H	1	Vorzeichen der Zahl 0 = positive Zahl -1 = negative Zahl (-1 = FFH) 2 = Range Byte 3 = String Byte
01H	2	Exponentbyte als vorzeichenbehafteter Integer
03H	8	64 Bit vorzeichenloser Nachkommateil der Zahl

Tabelle 10.13 LOTUS-Symphony-Fließkommadarstellung

Intern werden 11 Byte zum Speichern der Fließkommazahl benutzt. Im ersten Byte steht gemäß Tabelle 10.13 ein Wert, der die Interpretation der nachfolgenden Zahl spezifiziert. Mit den Codes 2 und 3 werden Range- und Stringwerte markiert.

Liegt in der betreffenden Zelle der Wert ERR vor, belegt LOTUS-Symphony die 11 Byte mit folgender Signatur:

Offset	Wert
00H	ERR = 0; NA = -1
01-02H	2047 = 0FFFH
03-0AH	8 * 0

Tabelle 10.14 Interne Darstellung der Werte ERR und NA in Symphony

Der Wert ERR enthält im ersten Byte die Signatur 0, während das Exponentwort mit 0FFFH belegt wird. Die 8 Bytes der Mantisse werden auf 0 gesetzt. Ähnliches gilt für den Code NA (*not available*). Hier wird lediglich das erste Byte auf den Wert -1 gesetzt. Die restlichen Bytes entsprechen der Kodierung laut Tabelle 10.14.

LABEL (Opcode 000FH)

Symphony speichert feste Textpassagen eines Rechenblatts in Labels. In der WKS-/WK1-Datei existiert zur Textablage ein eigener Satztyp mit dem Aufbau aus Tabelle 10.15.

Offset	Bytes	Bedeutung
00H	2	Opcode LABEL = 000FH
02H	2	Länge = 00xxH (variabel bis zu 245 Byte)
04H	1	Formatbyte
05H	2	Spaltennummer
07H	2	Zeilennummer
09H	5-245	ASCIIZ-String mit LABEL-Text

Tabelle 10.15 Symphony-Recordstruktur (Opcode 000FH)

Ein Beispiel für Labels findet sich in Bild 7.2 in den Textkonstanten (z. B. *Test Spread Sheet*). Der Datensatz ist von variabler Länge, je nach der Länge des Labeltextes.

Im ersten Datenbyte steht das Formatbyte mit der Kodierung gemäß Tabelle 10.8/10.9. Daran schließen sich zwei Wörter mit Spalten- und Zeilennummer an. Ab Offset 09H beginnt der eigentliche Text, der durch ein Nullbyte (00H) abgeschlossen werden muß und maximal 240 Byte lang sein darf. Das Feld selbst besitzt eine variable Länge von 5 bis 245 Byte. Das Byte ab Offset 09H enthält immer eines der folgenden Steuerzeichen:

Zeichen	Bedeutung
\	Wiederholungszeichen
'	linksbündiger Text

Tabelle 10.16 Steuerzeichen im Symphony-LABEL-Record

Zeichen	Bedeutung
"	rechtsbündiger Text
^	zentrierter Text

Tabelle 10.16 Steuerzeichen im Symphony-LABEL-Record

Mit dem Zeichen »\« werden in Symphony Wiederholungen eingeleitet. Es ist mir allerdings nicht klar, wann dieses Zeichen benutzt wird, da es bei den Textlabels im Testbeispiel nicht verwendet wird.

FORMULA (Opcode 0010H)

Neben reinen Zahlenwerten kann in einer Symphony-Zelle auch eine Berechnungsformel stehen. Diese Formel wird in einem Satz mit dem Opcode 10H im folgendem Format gespeichert:

Offset	Bytes	Bedeutung
00H	2	Opcode FORMULA = 0010H
02H	2	Länge = xxxxH (variabel bis zu 2064 Byte)
04H	1	Formatbyte
05H	2	Spaltennummer
07H	2	Zeilennummer
09H	8	Formelergebnis als 64 Bit IEEE-Long Real
11H	2	Länge der Formel in Byte
13H	15-2063	Formelcode (max. 2048 Byte)

Tabelle 10.17 Symphony-Recordstruktur (Opcode 0010H)

Im Datenfeld wird zuerst das Formatbyte für die Zelle gemäß der Kodierung in Tabelle 10.8/10.9 gespeichert. Daran schließen sich die Koordinaten für die Zelle als zwei 16-Bit-Werte an. Ab Offset 09H ist das Ergebnis der Berechnungsformel als 8 Byte-IEEE-Fließkommazahl mit doppelter Genauigkeit verzeichnet. Die Länge der Formel in Byte findet sich im folgenden Wort. Den Abschluß des Datenfeldes bildet der Formelcode. Die Länge des Datensatzes variiert zwischen 23 und 2064 Byte, wobei die Formel eine Länge zwischen 15 und 2048 Byte haben darf. Sowohl LOTUS als auch Symphony setzen eine eingegebene Formel direkt in die umgekehrte polnische Notation um. Jeder Eintrag in dieser Formel wird durch einen eigenen Funktionscode mit zugehörigem Datenfeld dargestellt:

```
"Code,Datenfeld",.....,"Code,Datenfeld"
```

Der Code umfaßt ein Byte und spezifiziert den Typ des dargestellten Operators (Variable, Konstante, Klammer, Addition etc.). Daran schließen sich die Daten für diesen Operator an.

Hierbei gilt folgende Kodierung:

Code	Bytes	Bedeutung
00H	1 8	Konstante 64 Bit Long Real Zahl
01H	1 2 2	Variable Spaltennummer (LSB zuerst) Zeilennummer (LSB zuerst)
02H	1 2 2 2 2	Range Start Spaltennummer (LSB zuerst) Start Zeilennummer (LSB zuerst) Ende Spaltennummer (LSB zuerst) Ende Zeilennummer (LSB zuerst)
03H	1	Ende der Formel
04H	1	Klammer
05H	1 2	Integerkonstante 16-Bit-Integerwert
06H	1 x	Stringkonstante ASCIIZ-String variabler Länge
07H	1	unbekannt
08H	1	Negation (unary minus)
09H	1	Addition +
0AH	1	Subtraktion -
0BH	1	Multiplikation *
0CH	1	Division /
0DH	1	Exponentialfunktion ^
0EH	1	Gleich =
0FH	1	Ungleich <>
10H	1	Kleiner gleich <=
11H	1	Größer gleich >=
12H	1	Kleiner <
13H	1	Größer >
14H	1	AND
15H	1	OR
16H	1	NOT
17H	1	unary + (nicht bei Neuberechnung)
18H-1EH	1	--
1FH	1	@NA (not applicable)
20H	1	@ERR (Error)

Tabelle 10.18 Opcodes innerhalb einer Symphony-Formel

Code	Bytes	Bedeutung
21H	1	@ABS (Absolutwert)
22H	1	@INT (Integerwert)
23H	1	@SQRT (Wurzel)
24H	1	@LOG (Logarithmus Basis 10)
25H	1	@LN (natürlicher Logarithmus)
26H	1	@PI (Konstante Pi)
27H	1	@SIN (Sinusfunktion)
28H	1	@COS (Cosinusfunktion)
29H	1	@TAN (Tangensfunktion)
2AH	1	@ATAN2 (Arcustangens 4. Quadrant)
2BH	1	@ATAN (Arcustangens 2. Quadrant)
2CH	1	@ASIN (Arcussinusfunktion)
2DH	1	@ACOS (Arcuscosinusfunktion)
2EH	1	@EXP (Exponentialfunktion)
2FH	1	@MOD(X,Y) (Modulofunktion)
30H	1	@CHOOSE (Auswahlfunktion)
31H	1	@ISNA(x) (x=NA Then 1)
32H	1	@ISERR(x) (x=ERR Then 1)
33H	1	@FALSE (Return 0)
34H	1	@TRUE (Return 1)
35H	1	@RAND (Random Number 0...1)
36H	1	@DATE (Zahl der Tage seit 1.1.1990)
37H	1	@TODAY (Datumsnummer ausgeben)
38H	1	@PMT (Payment)
39H	1	@PV (Present Value)
3AH	1	@FV (Future Value)
3BH	1	@IF (IF-Abfrage)
3CH	1	@DAY (Tag des Monats)
3DH	1	@MONTH (Monat)
3EH	1	@YEAR (Jahr)
3FH	1	@ROUND (Rundungsfunktion)
40H	1	@TIME (Zeit)
41H	1	@HOUR (Stunden)
42H	1	@MINUTE (Minuten)

Tabelle 10.18 Opcodes innerhalb einer Symphony-Formel

Code	Bytes	Bedeutung
43H	1	@SECOND (Sekunden)
44H	1	@ISNUMBER (Wert = Nummer)
45H	1	@ISSTRING (Wert = String)
46H	1	@LENGTH (Längenfunktion)
47H	1	@VALUE (Wert ermitteln)
48H	1	@FIXED (Festzahl)
49H	1	@MID (Mittelwert)
4AH	1	@CHR (Zeichen ermitteln)
4BH	1	@ASCII
4CH	1	@FIND (suche)
4DH	1	@DATEVALUE
4EH	1	@TIMEVALUE
4FH	1	@CELLPOINTER
50H	1	@SUM (Range Cell Konstante)
51H	1	@AVG (Range Cell Konstante)
52H	1	@CNT (Range Cell Konstante)
53H	1	@MIN (Range Cell Konstante)
54H	1	@MAX (Range Cell Konstante)
55H	1	@VLOOKUP (X,Range,OFFSET)
56H	1	@NPV (Int, Range)
57H	1	@VAR (Range)
58H	1	@STD (Range)
59H	1	@IRR (Guess,Range)
5AH	1	@HLOOCKUP (X,Range,Offset)
5BH	1	DSUM (Datenbankfunktion)
5CH	1	DAVG (Datenbankfunktion)
5DH	1	DCNT (Datenbankfunktion)
5EH	1	DMIN (Datenbankfunktion)
5FH	1	DMAX (Datenbankfunktion)
60H	1	DVAR (Datenbankfunktion)
61H	1	DSTD (Datenbankfunktion)
62H	1	@INDEX
63H	1	@COLS
64H	1	@ROWS

Tabelle 10.18 Opcodes innerhalb einer Symphony-Formel

Code	Bytes	Bedeutung
65H	1	@REPEAT
66H	1	@UPPER
67H	1	@LOWER
68H	1	@LEFT
69H	1	@RIGHT
6AH	1	@REPLACE
6BH	1	@PROPER
6CH	1	@CELL
6DH	1	@TRIM
6EH	1	@CLEAN
6FH	1	@S
70H	1	@V
71H	1	@STREQ
72H	1	@CALL
73H	1	@APP (Symphony 1.0) @INDIRECT (Symphony 1.1)
74H	1	@RATE
75H	1	@TERM
76H	1	@CTERM
77H	1	@SLN
78H	1	@SOY
79H	1	@DDB
7AH-9BH	1	---
9CH	1	@AAFSTART
CEH	1	---
FFH	1	@AAFMAX (Symphony 1.1)

Tabelle 10.18 Opcodes innerhalb einer Symphony-Formel

TABLE (Opcode 0018H)

Mit diesem Datensatz werden Datentabellen aus Symphony gespeichert. Folgende Struktur liegt hier zugrunde:

Offset	Bytes	Bedeutung
00H	2	Opcode LABEL = 0018H
02H	2	Länge = 0019H (25 Bytes)

Tabelle 10.19 Symphony-Recordstruktur (Opcode 0018H)

Offset	Bytes	Bedeutung
04H	1	0 = keine Tabelle 1 = Tabelle 1 2 = Tabelle 2
05H	2	Tabelle Range Start Spaltennummer
07H	2	Tabelle Range Start Zeilennummer
09H	2	Tabelle Range Ende Spaltennummer
0BH	2	Tabelle Range Ende Zeilennummer
0DH	2	Eingabezelle 1 Start Spalte
0FH	2	Eingabezelle 1 Start Zeile
11H	2	Eingabezelle 1 Ende Spalte
13H	2	Eingabezelle 1 Ende Zeile
15H	2	Eingabezelle 2 Start Spalte
17H	2	Eingabezelle 2 Start Zeile
19H	2	Eingabezelle 2 Ende Spalte
1BH	2	Eingabezelle 2 Ende Zeile

Tabelle 10.19 Symphony-Recordstruktur (Opcode 0018H)

In Symphony spezifiziert der Record die Lage der *Wenn...dann*-Tabellen. Die genaue Bedeutung der Datenstruktur ist nicht bekannt.

PRINT_RANGE (Opcode 001AH)

Mit diesem Datensatz werden ab Symphony 1.1 Daten eines Druckbereichs (PRINT Range) gespeichert. Der Datensatz besitzt folgendes Format:

Offset	Bytes	Bedeutung
00H	2	Opcode PRINT_RANGE = 001AH
02H	2	Länge = 0008H
04H	2	Start Spalte
06H	2	Start Zeile
08H	2	Ende Spalte
0AH	2	Ende Zeile

Tabelle 10.20 Symphony-Recordstruktur (Opcode 001AH)

In diesem Satz stehen die Koordinaten eines Rechenblattausschnitts, der auf dem Drukker ausgegeben wird. Bei einem Druckkommando werden alle Zellen innerhalb des Fensters ausgegeben.

FILL_RANGE (Opcode 001CH)

Mit diesem Datensatz werden Daten eines Fill-Bereichs (*Fill Range*), d.h. die Koordinaten eines mit Daten zu füllenden Rechenblattausschnitts, gespeichert. Das Format sieht so aus:

Offset	Bytes	Bedeutung
00H	2	Opcode FILL_RANGE = 001CH
02H	2	Länge = 0008H
04H	2	Start Spalte
06H	2	Start Zeile
08H	2	Ende Spalte
0AH	2	Ende Zeile

Tabelle 10.21 Symphony-Recordstruktur (Opcode 001CH)

HRANGE (Opcode 0020H)

Mit diesem Datensatz werden interne Daten eines Bereichs in Symphony gespeichert. Der Datensatz besitzt folgendes Format:

Offset	Bytes	Bedeutung
00H	2	Opcode HRANGE = 0020H
02H	2	Länge = 0010H (16 Byte)
04H	2	Value Range Start Spalte
06H	2	Value Range Start Zeile
08H	2	Value Range Ende Spalte
0AH	2	Value Range Ende Zeile
0CH	2	Binary Range Start Spalte
0EH	2	Binary Range Start Zeile
10H	2	Binary Range Ende Spalte
12H	2	Binary Range Ende Zeile

Tabelle 10.22 Symphony-Recordstruktur (Opcode 0020H)

Die genaue Bedeutung dieses Datensatzes ist nicht bekannt.

PROTECT (Opcode 0024H)

In diesem Datensatz speichert Symphony die Information darüber, ob das Arbeitsblatt geschützt ist oder nicht. Es wird nur ein Byte mit der Information über einen eventuellen Schreibschutz der Arbeitsblattzellen gespeichert.

Offset	Bytes	Bedeutung
00H	2	Opcode PROTECT = 0024H
02H	2	Länge = 0001H
04H	1	Code 00: Schutz ausgeschaltet 01: Schutz eingeschaltet

Tabelle 10.23 Symphony-Recordstruktur (Opcode 0024H)

LABEL_FORMAT (Opcode 0029H)

In diesem Datensatz merkt sich Symphony, auf welche Art und Weise Labels justiert werden. Es gilt folgende Kodierung:

Offset	Bytes	Bedeutung
00H	2	Opcode LABEL_FORMAT = 0029H
02H	2	Länge = 0001H
04H	1	Formatcode 27H linksbündig 22H rechtsbündig 5EH zentriert

Tabelle 10.24 Symphony-Recordstruktur (Opcode 0029H)

In Abhängigkeit des eingetragenen Codebytes werden Labels linksbündig, rechtsbündig oder zentriert geschrieben.

CALC_COUNT (Opcode 002FH)

In diesem Datensatz legt Symphony die Information darüber ab, wie oft eine Berechnung (*Iteration*) auszuführen ist.

Offset	Bytes	Bedeutung
00H	2	Opcode CALC_COUNT = 002FH
02H	2	Länge = 0001H
04H	1	Wiederholungszähler

Tabelle 10.25 Symphony-Recordstruktur (Opcode 002FH)

WINDOW (Opcode 0032H)

Dieser Datensatz enthält in Symphony die Information über den Fensteraufbau.

Offset	Bytes	Bedeutung
00	2	Opcode WINDOW = 0032H

Tabelle 10.26 Symphony-Recordstruktur (Opcode 0032H)

Offset	Bytes	Bedeutung
02	2	Länge = 0090H (144 Byte)
04/ 04H	16	Name des Fensters als ASCII-Zeichenstring
20/ 14H	2	Cursorposition Spalte
22/ 16H	2	Cursorposition Zeile
24/ 18H	1	Formatbyte
25/ 19H	1	unbenutzt
26/ 1AH	2	Spaltenbreite
28/ 1CH	2	Spaltenzahl
30/ 1EH	2	Zeilenzahl
32/ 20H	2	Anfangsspalte des »no title«-Bereichs
34/ 22H	2	Anfangszeile des »no title«-Bereichs
36/ 24H	2	Zahl der Spalten Titelbereich
38/ 26H	2	Zahl der Zeilen Titelbereich
40/ 28H	2	linke Spalte Beginn Titelbereich
42/ 2AH	2	oberste Zeile Titelbereich
44/ 2CH	2	HOME-Position Cursor Spalte
46/ 2EH	2	HOME-Position Cursor Zeile
48/ 30H	2	Zahl der Bildschirmspalten
50/ 32H	2	Zahl der Bildschirmzeilen
52/ 34H	1	Hidden-Status 00H = hidden FFH = unhidden
53/ 35H	1	vorhergehendes Fenster 00 = SHEET 01 = DOC 02 = GRAPH 03 = COMM 04 = FORM 05 = APPLICATION
54/ 36H	1	Anzeige der Umrandung 00H = als Zelle FFH = keine LOTUS 1-2-3 Zelle
55/ 37H	1	Anzeige der Randlinien 00H = Linien anzeigen FFH = keine Linien anzeigen
56/ 38H	2	Window Range Start Spalte
58/ 3AH	2	Window Range Start Zeile
60/ 3CH	2	Window Range Ende Spalte

Tabelle 10.26 Symphony-Recordstruktur (Opcode 0032H)

Offset	Bytes	Bedeutung
62/ 3EH	2	Window Range Ende Zeile
64/ 40H	2	Offset
66/ 42H	1	Insert Mode Flag 00 = Einfügemodus AUS alle anderen Werte = Einfügemodus EIN
67/ 43H	16	Graph Name
83/ 53H	1	Fenstertyp 00 = SHEET 01 = DOC 02 = GRAPH 03 = COMM 04 = FORM 05 = APPLICATION
84/ 54H	1	Auto Display Mode Flag 61H = ('a') Auto Display EIN alle anderen Werte = manuelle Anzeige
85/ 55H	1	Forms Filter 00H = Filter aktiv alle anderen Werte = kein Filter
86/ 56H	16	zugewiesener FORM-Name
102/66H	2	FORMS aktuelle Recordnummer
104/68H	1	Space Display Flag 00H = Leerzeichen nicht anzeigen alle anderen Werte = Leerzeichen anzeigen
105/69H	1	Zahl der Leerzeilen 01H = 1 Leerzeile 02H = 2 Leerzeile 03H = 3 Leerzeile
106/6AH	1	Textausrichtung 'l' = linksbündig (Code 6CH) 'r' = rechtsbündig (Code 72H) 'c' = zentriert (Code 63H) 'e' = gerade Position (Code 65H)
107/6BH	2	rechter Rand FFH = Standardeinstellung gültig 00H – FEH rechte Randeinstellung
109/6DH	2	linker Rand FFH = Standardeinstellung gültig 00H – FEH linke Randeinstellung
111/6FH	2	TAB Intervall

Tabelle 10.26 Symphony-Recordstruktur (Opcode 0032H)

Offset	Bytes	Bedeutung
113/71H	1	Return Display Mode 00H = Carriage Return softwaremäßig erzeugen sonst = »harte Returns« 0DH,0AH im Text
114/72H	1	automatische Justierung 00H = ausgeschaltet sonst = eingeschaltet
115/73H	16	Name der Anwendung
131/83H	17	reserviert

Tabelle 10.26 Symphony-Recordstruktur (Opcode 0032H)

Die Belegung der einzelnen Felder in diesem Record orientiert sich an den Funktionen von Symphony.

STRING (Opcode 0033H)

Offset	Bytes	Bedeutung
00H	2	Opcode STRING = 0033H
02H	2	Länge = 00xxH (variabel)
04H	1	Formatcode
05H	2	Spaltennummer
07H	2	Zeilennummer
09H	xx	ASCIIZ-String variabler Länge

Tabelle 10.27 Symphony-Recordstruktur (Opcode 0033H)

Dieser Datensatz dient zur Aufnahme des Ergebnisses einer Stringfunktion. Im ersten Datenbyte steht der Formatcode der Zelle gemäß Tabelle 10.8/10.9. Die folgenden Felder enthalten die Koordinaten der betreffenden Zelle. Daran schließt sich der String mit einer variablen Länge an. Der String ist mit einem Nullbyte (00H) abgeschlossen.

LOCK_PASSWORD (Opcode 0037H)

Dieser Datensatz enthält ein Paßwort, mit dem der Schreibzugriff auf bestimmte Zellen gesperrt wird.

Offset	Bytes	Bedeutung
00H	2	Opcode LOCK_PASSWORD = 0037H
02H	2	Länge = 0004H
04H	4	Paßwort

Tabelle 10.28 Symphony-Recordstruktur (Opcode 0037H)

LOCKED (Opcode 0038H)

Dieser Datensatz enthält das Lock-Flag, das den Zustand des Schreibschutzes anzeigt.

Offset	Bytes	Bedeutung
00H	2	Opcode LOCKED = 0038H
02H	2	Länge = 0001H
04H	1	Lock-Flag 00 = Lock ausgeschaltet 01 = Lock eingeschaltet

Tabelle 10.29 Symphony-Recordstruktur (Opcode 0038H)

Wenn das Flag gesetzt ist, unterdrückt Symphony den Schreibzugriff auf bestimmte Zellen. Dieser Zugriff ist dann nur noch über das gesetzte Paßwort möglich.

QUERY (Opcode 003CH)

Dieser Datensatz beschreibt die Einstellungen für den QUERY-Befehl.

Offset	Bytes	Bedeutung
00 / 00H	2	Opcode QUERY = 003CH
02 / 02H	2	Länge = 007FH (127 Byte)
04 / 04H	16	Name als ASCIIZ-String
20 / 14H	2	Input Range Start Spalte
22 / 16H	2	Input Range Start Zeile
24 / 18H	2	Input Range Ende Spalte
26 / 1AH	2	Input Range Ende Zeile
28 / 1CH	2	Output Range Start Spalte
30 / 1EH	2	Output Range Start Zeile
32 / 20H	2	Output Range Ende Spalte
34 / 22H	2	Output Range Ende Zeile
36 / 24H	2	Criteria Range Start Spalte
38 / 26H	2	Criteria Range Start Zeile
40 / 28H	2	Criteria Range Ende Spalte
42 / 2AH	2	Criteria Range Ende Zeile
44 / 2CH	2	Form Entry Start Spalte
46 / 2EH	2	Form Entry Start Zeile
48 / 30H	2	Form Entry Ende Spalte
50 / 32H	2	Form Entry Ende Zeile

Tabelle 10.30 Symphony-Recordstruktur (Opcode 003CH)

Offset	Bytes	Bedeutung
52 / 34H	2	Form Definition Range Start Spalte
54 / 36H	2	Form Definition Range Start Zeile
56 / 38H	2	Form Definition Range Ende Spalte
58 / 3AH	2	Form Definition Range Ende Zeile
60 / 3CH	2	Report Output Start Spalte
62 / 3EH	2	Report Output Start Zeile
64 / 40H	2	Report Output Ende Spalte
66 / 42H	2	Report Output Ende Zeile
68 / 44H	2	Report Header Start Spalte
70 / 46H	2	Report Header Start Zeile
72 / 48H	2	Report Header Ende Spalte
74 / 4AH	2	Report Header Ende Zeile
76 / 4CH	2	Report Footer Start Spalte
78 / 4EH	2	Report Footer Start Zeile
80 / 50H	2	Report Footer Ende Spalte
82 / 52H	2	Report Footer Ende Zeile
84 / 54H	2	Table Range Start Spalte
86 / 56H	2	Table Range Start Zeile
88 / 58H	2	Table Range Ende Spalte
90 / 5AH	2	Table Range Ende Zeile
92 / 5CH	2	Input Cell Start Spalte
94 / 5EH	2	Input Cell Start Zeile
96 / 60H	2	Input Cell Ende Spalte
98 / 62H	2	Input Cell Ende Zeile
100/ 64H	2	1. Key Range Start Spalte
102/ 66H	2	1. Key Range Start Zeile
104/ 68H	2	1. Key Range Ende Spalte
106/ 6AH	2	1. Key Range Ende Zeile
108/ 6CH	2	2. Key Range Start Spalte
110/ 6EH	2	2. Key Range Start Zeile
112/ 70H	2	2. Key Range Ende Spalte
114/ 72H	2	2. Key Range Ende Zeile
116/ 74H	2	3. Key Range Start Spalte
118/ 76H	2	3. Key Range Start Zeile

Tabelle 10.30 Symphony-Recordstruktur (Opcode 003CH)

Offset	Bytes	Bedeutung
120/ 78H	2	3. Key Range Ende Spalte
122/ 7AH	2	3. Key Range Ende Zeile
124/ 7CH	1	letztes Kommando 00 = kein Kommando 01 = Find Kommando 02 = Extract Kommando 03 = Delete Kommando 04 = Unique Kommando
125/ 7DH	1	1. Key Order 00H = absteigend FFH = aufsteigend
126/ 7EH	1	2. Key Order 00H = absteigend FFH = aufsteigend
127/ 7FH	1	3. Key Order 00H = absteigend FFH = aufsteigend
128/ 80H	1	Report Recordzahl Flag 00H = mehrere Records FFH = ein Record
129/ 81H	1	Recordzahl Flag 00H = mehrere Records FFH = ein Record
130/ 82H	1	Marken Flag 00H = Marken vorhanden FFH = keine Marken

Tabelle 10.30 Symphony-Recordstruktur (Opcode 003CH)

QUERY_NAME (Opcode 003DH)

Dieser Datensatz dient zur Aufnahme des aktuellen QUERY-Namens.

Offset	Bytes	Bedeutung
00H	2	Opcode QUERY_NAME = 003DH
02H	2	Länge = 0010H
04H	16	ASCIIZ-String mit aktuellem QUERY-Namen

Tabelle 10.31 Symphony-Recordstruktur (Opcode 003DH)

PRINT (Opcode 003EH)

Dieser Datensatz enthält die Definitionen für den PRINT-Record in Symphony.

Offset	Bytes	Bedeutung
00 / 00H	2	Opcode PRINT = 003EH
02 / 02H	2	Länge = 02A7H (679 Byte)
04 / 04H	16	Name als ASCIIZ-String
20 / 14H	2	Source Range Start Spalte
22 / 16H	2	Source Range Start Zeile
24 / 18H	2	Source Range Ende Spalte
26 / 1AH	2	Source Range Ende Zeile
28 / 1CH	2	Zeilenrand Start Spalte
30 / 1EH	2	Zeilenrand Start Start Zeile
32 / 20H	2	Zeilenrand Start Ende Spalte
34 / 22H	2	Zeilenrand Start Ende Zeile
36 / 24H	2	Spaltenrand Start Spalte
38 / 26H	2	Spaltenrand Start Zeile
40 / 28H	2	Spaltenrand Ende Spalte
42 / 2AH	2	Spaltenrand Ende Zeile
44 / 2CH	2	Ziel Range Start Spalte
46 / 2EH	2	Ziel Range Start Zeile
48 / 30H	2	Ziel Range Ende Spalte
50 / 32H	2	Ziel Range Ende Zeile
52 / 34H	1	PRINT Format 00 = Ausgabe des Displays sonst Ausgabe der Formeln
53 / 35H	1	Einzelblatteinzug 00 = Unterbrechung am Seitenende, sonst keine Unterbrechung
54 / 36H	1	Leerzeilen
55 / 37H	2	linker Rand
57 / 39H	2	rechter Rand
59 / 3BH	2	Seitenlänge
61 / 3DH	2	oberer Seitenrand
63 / 3FH	2	unterer Seitenrand
65 / 41H	41	Set Up String als ASCIIZ-String (40 Byte)
106/ 6AH	241	Kopfzeile als ASCIIZ-String (240 Byte)
347/15BH	241	Fußzeile als ASCIIZ-String (240 Byte)
589/24DH	16	Name der Quelldatenbank als ASCIIZ-String
605/25DH	1	Attribute 00 = keine Attribute sonst Attribute vorhanden

Tabelle 10.32 Symphony-Recordstruktur (Opcode 003EH)

Offset	Bytes	Bedeutung
606/25EH	1	Unterdrückung von Leerzeichen, 00 = nein sonst ja
607/25FH	1	Ziel der Druckausgabe 00 = Drucker 01 = Datei 02 = Spread Sheet Bereich
608/260H	2	Start Seite
610/262H	2	Ende Seite
612/264H	70	Name der Zieldatei als ASCIIZ-String
682/2AAH	1	Wait Flag,00 = nein sonst ja

Tabelle 10.32 Symphony-Recordstruktur (Opcode 003EH)

PRINT_NAME (Opcode 003FH)

Dieser Datensatz dient zur Aufnahme des aktuellen Namens des PRINT-Records.

Offset	Bytes	Bedeutung
00H	2	Opcode PRINT_NAME = 003FH
02H	2	Länge = 0010H
04H	16	ASCIIZ-String mit aktuellem PRINT-Namen

Tabelle 10.33 Symphony-Recordstruktur (Opcode 003FH)

GRAPH_2 (Opcode 0040H)

Dieser Datensatz beschreibt die Einstellungen zur Erzeugung von Grafiken in Symphony.

Offset	Bytes	Bedeutung	
00 / 00H	2	Opcode GRAPH_2 = 0040H	
02 / 02H	2	Länge = 01F3H (499 Byte)	
04 / 04H	16	Name als ASCIIZ-String	
20 / 14H	2	X Range	Start Spalte
22 / 16H	2	X Range	Start Zeile
24 / 18H	2	X Range	Ende Spalte
26 / 1AH	2	X Range	Ende Zeile
28 / 1CH	2	A Range	Start Spalte
30 / 1EH	2	A Range	Start Zeile
32 / 20H	2	A Range	Ende Spalte
34 / 22H	2	A Range	Ende Zeile
36 / 24H	2	B Range	Start Spalte

Tabelle 10.34 Symphony-Recordstruktur (Opcode 0040H)

Offset	Bytes	Bedeutung	
38 / 26H	2	B Range	Start Zeile
40 / 28H	2	B Range	Ende Spalte
42 / 2AH	2	B Range	Ende Zeile
44 / 2CH	2	C Range	Start Spalte
46 / 2EH	2	C Range	Start Zeile
48 / 30H	2	C Range	Ende Spalte
50 / 32H	2	C Range	Ende Zeile
52 / 34H	2	D Range	Start Spalte
54 / 36H	2	D Range	Start Zeile
56 / 38H	2	D Range	Ende Spalte
58 / 3AH	2	D Range	Ende Zeile
60 / 3CH	2	E Range	Start Spalte
62 / 3EH	2	E Range	Start Zeile
64 / 40H	2	E Range	Ende Spalte
66 / 42H	2	E Range	Ende Zeile
68 / 44H	2	F Range	Start Spalte
70 / 46H	2	F Range	Start Zeile
72 / 48H	2	F Range	Ende Spalte
74 / 4AH	2	F Range	Ende Zeile
76 / 4CH	2	A Labels	Start Spalte
78 / 4EH	2	A Labels	Start Zeile
80 / 50H	2	A Labels	Ende Spalte
82 / 52H	2	A Labels	Ende Zeile
84 / 54H	2	B Labels	Start Spalte
86 / 56H	2	B Labels	Start Zeile
88 / 58H	2	B Labels	Ende Spalte
90 / 5AH	2	B Labels	Ende Zeile
92 / 5CH	2	C Labels	Start Spalte
94 / 5EH	2	C Labels	Start Zeile
96 / 60H	2	C Labels	Ende Spalte
98 / 62H	2	C Labels	Ende Zeile
100/ 64H	2	D Labels	Start Spalte
102/ 66H	2	D Labels	Start Zeile
104/ 68H	2	D Labels	Ende Spalte

Tabelle 10.34 Symphony-Recordstruktur (Opcode 0040H)

Offset	Bytes	Bedeutung	
106/ 6AH	2	D Labels	Ende Zeile
108/ 6CH	2	E Labels	Start Spalte
110/ 6EH	2	E Labels	Start Zeile
112/ 70H	2	E Labels	Ende Spalte
114/ 72H	2	E Labels	Ende Zeile
116/ 74H	2	F Labels	Start Spalte
118/ 76H	2	F Labels	Start Zeile
120/ 78H	2	F Labels	Ende Spalte
122/ 7AH	2	F Labels	Ende Zeile
124/ 7CH	1	Graphtyp 00 = XY-Grafik 01 = Balkengrafik (Bar Graph) 02 = Tortengrafik (Pie Chart) 04 = Liniengrafik 05 = Balken übereinander (stacked bar)	
125/ 7DH	1	Gittertyp 00 = kein Gitter 01 = horizontal 02 = vertikal 03 = beide Richtungen	
126/ 7EH	1	Farbe 00 = Schwarzweiß FF = Farbdarstellung	
127/ 7FH	1	A Range Linienformat 00 = kein Linienformat 01 = Linie 02 = Symbol 03 = Linien + Symbol	
128/ 80H	1	B Range Linienformat 00 = kein Linienformat 01 = Linie 02 = Symbol 03 = Linien + Symbol	
129/ 81H	1	C Range Linienformat 00 = kein Linienformat 01 = Linie 02 = Symbol 03 = Linien + Symbol	
130/ 82H	1	D Range Linienformat 00 = kein Linienformat 01 = Linie 02 = Symbol 03 = Linien + Symbol	

Tabelle 10.34 Symphony-Recordstruktur (Opcode 0040H)

Offset	Bytes	Bedeutung
131/ 83H	1	E Range Linienformat 00 = kein Linienformat 01 = Linie 02 = Symbol 03 = Linien + Symbol
132/ 84H	1	F Range Linienformat 00 = kein Linienformat 01 = Linie 02 = Symbol 03 = Linien + Symbol
133/ 85H	1	A Range Daten-Label Justierung 00 = zentriert 01 = rechtsbündig 02 = unterhalb 03 = linksbündig 04 = oberhalb
134/ 86H	1	B Range Daten-Label Justierung 00 = zentriert 01 = rechtsbündig 02 = unterhalb 03 = linksbündig 04 = oberhalb
135/ 87H	1	C Range Daten-Label Justierung 00 = zentriert 01 = rechtsbündig 02 = unterhalb 03 = linksbündig 04 = oberhalb
136/ 88H	1	D Range Daten-Label-Justierung 00 = zentriert 01 = rechtsbündig 02 = unterhalb 03 = linksbündig 04 = oberhalb
137/ 89H	1	E Range Daten-Label-Justierung 00 = zentriert 01 = rechtsbündig 02 = unterhalb 03 = linksbündig 04 = oberhalb
138/ 8AH	1	F Range Daten-Label-Justierung 00 = zentriert 01 = rechtsbündig 02 = unterhalb 03 = linksbündig 04 = oberhalb

Tabelle 10.34 Symphony-Recordstruktur (Opcode 0040H)

Offset	Bytes	Bedeutung
139/ 8BH	1	Skalierung X-Achse 00H = automatisch FFH = manuell
140/ 8CH	8	X-Achse Untergrenze (X lower limit) 64 Bit IEEE-Fließkommawert
148/ 94H	8	X-Achse Obergrenze (X upper limit) 64 Bit IEEE-Fließkommawert
156/ 9CH	1	Skalierung Y-Achse 00H = automatisch FFH = manuell
157/ 9DH	8	Y-Achse Untergrenze (X lower limit) 64 Bit IEEE-Fließkommawert
165/ A5H	8	Y-Achse Obergrenze (X upper limit) 64 Bit IEEE-Fließkommawert
173/ ADH	40	Text erster Titel (40 Zeichen)
213/ D5H	40	Text zweiter Titel (40 Zeichen)
253/ FDH	40	Text X-Achse (40 Zeichen)
293/125H	40	Text Y-Achse (40 Zeichen)
333/14DH	20	Legende A-Achse (20 Zeichen)
353/161H	20	Legende B-Achse (20 Zeichen)
373/175H	20	Legende C-Achse (20 Zeichen)
393/189H	20	Legende D-Achse (20 Zeichen)
413/19DH	20	Legende E-Achse (20 Zeichen)
433/1B1H	20	Legende F-Achse (20 Zeichen)
53/1C5H	1	X-Format
454/1C6H	1	Y-Format
455/1C7H	2	Skip Faktor
457/1C9H	1	Skalenfaktor X-Achse: 00H = eingeschaltet FFH = ausgeschaltet
458/1CAH	1	Skalenfaktor Y-Achse: 00H = eingeschaltet FFH = ausgeschaltet
459/1CBH	1	Unterdrückung (suppress): 00 = eingeschaltet sonst ausgeschaltet
460/1CCH	8	Nullpunkt Balkengrafik (IEEE-Fließkommazahl)
468/1D4H	8	lineare Skalierung X-Achse (Fließkomma)

Tabelle 10.34 Symphony-Recordstruktur (Opcode 0040H)

Offset	Bytes	Bedeutung
476/1DCH	8	lineare Skalierung Y-Achse (Fließkomma)
484/1E4H	1	logarithmische Skalierung X-Achse
485/1E5H	1	logarithmische Skalierung Y-Achse
486/1E6H	1	Farbe Grafik X-Skala
487/1E7H	1	Farbe Grafik A-Skala
488/1E8H	1	Farbe Grafik B-Skala
489/1E9H	1	Farbe Grafik C-Skala
490/1EAH	1	Farbe Grafik D-Skala
491/1EBH	1	Farbe Grafik E-Skala
492/1ECH	1	Farbe Grafik F-Skala
493/1EDH	2	Höhe Y-Achse
495/1EFH	8	Breite X-Achse (Aspekt)

Tabelle 10.34 Symphony-Recordstruktur (Opcode 0040H)

GRAPH_NAME (Opcode 0041H)

Dieser Datensatz dient zur Aufnahme des aktuellen Namens des GRAPH-Records.

Offset	Bytes	Bedeutung
00H	2	Opcode GRAPH_NAME = 0041H
02H	2	Länge = 0010H
04H	16	ASCIIZ-String mit aktuellem GRAPH-Namen

Tabelle 10.35 Symphony-Recordstruktur (Opcode 0041H)

ZOOM (Opcode 0042H)

ZOOM beschreibt die Originalkoordinaten eines vergrößerten Fensters.

Offset	Bytes	Bedeutung
00H	2	Opcode ZOOM = 0042H
02H	2	Länge = 0009H
04H	1	ZOOM Flag 00 = ZOOM aus 01 = ZOOM ein
05H	2	X-Koordinate
07H	2	Y-Koordinate
09H	2	Spaltenzahl
0BH	2	Zeilenzahl

Tabelle 10.36 Symphony-Recordstruktur (Opcode 0042H)

SYMPHONY_SPLIT (Opcode 0043H)

In diesem Datensatz wird die Zahl der geteilten (*split*) Fenster gespeichert. Der Satz besitzt folgenden Aufbau:

Offset	Bytes	Bedeutung
00H	2	Opcode SYMPHONY_SPLIT = 0043H
02H	2	Länge = 0002H
04H	2	Zahl der geteilten Fenster

Tabelle 10.37 Symphony-Recordstruktur (Opcode 0043H)

NUMBER_SCREEN_ROWS (Opcode 0044H)

In diesem Datensatz wird die Zeilenanzahl der Bildschirmausgabe festgehalten.

Offset	Bytes	Bedeutung
00H	2	Opcode NUMBER_SCREEN_ROWS = 0044H
02H	2	Länge = 0002H
04H	2	Zeilenzahl auf dem Bildschirm

Tabelle 10.38 Symphony-Recordstruktur (Opcode 0044H)

NUMBER_SCREEN_COLUMNS (Opcode 0045H)

In diesem Datensatz wird die Spaltenanzahl der Bildschirmausgabe festgehalten.

Offset	Bytes	Bedeutung
00H	2	Opcode NUMBER_SCREEN_COLUMNS = 0045H
02H	2	Länge = 0002H
04H	2	Spaltenzahl auf dem Bildschirm

Tabelle 10.39 Symphony-Recordstruktur (Opcode 0045H)

RULER (Opcode 0046H)

In diesem Datensatz wird der *Ruler-Range* gespeichert. Es gilt dabei folgender Satzaufbau:

Offset	Bytes	Bedeutung
00H	2	Opcode RULER = 0046H
02H	2	Länge = 0019H (25 Byte)
04H	16	Name als ASCIIZ-String
14H	2	Range Start Spalte

Tabelle 10.40 Symphony-Recordstruktur (Opcode 0046H)

Offset	Bytes	Bedeutung
16H	2	Range Start Zeile
18H	2	Range Ende Spalte
1AH	2	Range Ende Zeile
1CH	1	Range Typ 00 = single cell 01 = Range

Tabelle 10.40 Symphony-Recordstruktur (Opcode 0046H)

Das letzte Byte spezifiziert, ob sich die Angaben der Range-Felder wirklich auf einen ganzen Bereich (*Range*) oder nur auf eine Zelle (*single cell*) beziehen.

NAMED_SHEET (Opcode 0047H)

In diesem Datensatz werden die Daten eines Bereichs mit seinem Namen gespeichert. Es gilt dabei folgender Satzaufbau:

Offset	Bytes	Bedeutung
00H	2	Opcode NAMED_SHEET = 0047H
02H	2	Länge = 0019H (25 Byte)
04H	16	Name als ASCIIZ-String
14H	2	Range Start Spalte
16H	2	Range Start Zeile
18H	2	Range Ende Spalte
1AH	2	Range Ende Zeile
1CH	1	Range Typ 00 = single cell 01 = Range

Tabelle 10.41 Symphony-Recordstruktur (Opcode 0047H)

Das letzte Byte spezifiziert, ob sich die Angaben der Range-Felder wirklich auf einen ganzen Bereich (*Range*) oder nur auf eine Zelle (*single cell*) beziehen. Mit dem Record wird der durch einen Namen zu identifizierende Bereich markiert.

AUTO_COMM (Opcode 0048H)

Dieser Datensatz enthält den Namen der automatisch zu ladenden Kommunikationsdatei.

Offset	Bytes	Bedeutung
00H	2	Opcode AUTO_COMM = 0048H

Tabelle 10.42 Symphony-Recordstruktur (Opcode 0048H)

Offset	Bytes	Bedeutung
02H	2	Länge = 0041H (65 Byte)
04H	65	Pfadname als ASCIIZ-String

Tabelle 10.42 Symphony-Recordstruktur (Opcode 0048H)

In dem ASCIIZ-String muß der Dateiname einschließlich Laufwerk und Pfadname spezifiziert werden. Symphony lädt die betreffende Datei mit den Informationen für die Kommunikation automatisch aus dieser Datei.

AUTO_MACRO (Opcode 0049H)

Dieser Datensatz spezifiziert einen Bereich im Rechenblatt, in dem ein Makro steht. Der Makro wird automatisch ausgeführt:

Offset	Bytes	Bedeutung
00H	2	Opcode AUTO_MACRO = 0049H
02H	2	Länge = 0008H
04H	2	Start Spalte
06H	2	Start Zeile
08H	2	Ende Spalte
0AH	2	Ende Zeile

Tabelle 10.43 Symphony-Recordstruktur (Opcode 0049H)

PARSE (Opcode 004AH)

Offset	Bytes	Bedeutung
00H	2	Opcode PARSE = 004AH
02H	2	Länge = 0010H (16 Byte)
04H	2	Parse Range Start Spalte
06H	2	Parse Range Start Zeile
08H	2	Parse Range Ende Spalte
0AH	2	Parse Range Ende Zeile
0CH	2	Review Range Start Spalte
0EH	2	Review Range Start Zeile
10H	2	Review Range Ende Spalte
12H	2	Review Range Ende Zeile

Tabelle 10.44 Symphony-Recordstruktur (Opcode 004AH)

In diesem Datensatz stehen die Adressen eines Zellbereichs, der die Informationen über den QUERY-Befehl enthält.

WKS_PASSWORD (Opcode 004BH)

Dieser Datensatz dient zur Entschlüsselung eines kodierten Arbeitsblatts. Der Satztyp wird ab LOTUS 1-2-3, Version 2.0 und in Symphony ab Version 1.1 unterstützt.

Offset	Bytes	Bedeutung
00H	2	Opcode WKS_PASSWORD = 004BH
02H	2	Länge = 0004H
04H	4	Paßwort

Tabelle 10.45 Symphony-Recordstruktur (Opcode 004BH)

Die genaue Bedeutung dieses Satzes ist nicht bekannt.

HIDDEN_VECTOR (Opcode 0064H)

Offset	Bytes	Bedeutung
00H	2	Opcode HIDDEN_VECTOR = 0064H
02H	2	Länge = 0020H (32 Byte)
04H	32	Bitfeld

Tabelle 10.46 Symphony-Recordstruktur (Opcode 0064H)

Dieser Datensatz enthält 32 Bytes, die genau 256 Einzelbits repräsentieren. Jedem Bit ist eine Spalte des Arbeitsblatts zugeordnet. Ist das Bit auf 1 gesetzt, handelt es sich um eine verborgene, d.h. nicht sichtbare Spalte (*hidden*).

Das Bitfeld ist so angeordnet, daß das niedrigstwertige Byte zuerst gespeichert wird. Bit 0 im Byte 0 gehört dann zur ersten Spalte. Die Funktion ist erst ab Symphony 1.1 verfügbar.

CELL_POINTER_INDEX (Opcode 0096H)

Dieser Datensatz enthält ab Symphony 1.1 den *Cell Pointer Index* und besitzt folgenden Aufbau:

Offset	Bytes	Bedeutung
00H	2	Opcode CELL_POINTER_INDEX = 0096H
02H	2	Länge = 0006H
04H	2	Spaltennummer (Integerwert)
06H	2	niedrigste aktive Zellennummer
08H	2	höchste aktive Zellennummer

Tabelle 10.47 Symphony-Recordstruktur (Opcode 0096H)

Der Satz enthält eine Liste mit Spalten der aktiven Zellen. Die genaue Bedeutung dieses Records ist nicht bekannt.

Im Format späterer Symphony-Versionen sind weitere Opcodes definiert, deren Belegung zur Zeit allerdings nicht bekannt ist – sie lassen sich jedoch anhand der oben angeführten Informationen leicht identifizieren. Da die Opcodes aufwärtskompatibel sind, können die Daten in der Regel auch aus zukünftigen Dateiversionen gelesen werden.

13 Standard Interface Format (SIF)

In früheren Version von Open Access wurde ein weiteres Format zur Übertragung von Daten definiert, das den Inhalt der Kalkulationsblätter als ASCII-Datei überträgt.

Ähnlich wie bei DIF und SDI wird jedes Element des Kalkulationsblatts in einer eigenen Zeile übertragen. Die erste Zeile der Datei enthält eine Nummer, die die Zeilenzahl des Rechenblatts angibt. Jedes Datum wird durch ein Komma separiert, während ein Semikolon das Ende einer Rechenblattzeile markiert. Texte werden mit Apostroph gerahmt. Bild 13.1 zeigt das Kalkulationsblatt aus Bild 7.1 im SIF-Format:

```
10
' ',
' ',
'Test Spread Sheet';
' ',
' ',
'=================';
'Produkt'
' ',
'Preis',
'Rabatt',
'Netto';
'-----------------------------------------';
'Disketten 5 1/4',
15.00,
10.00,
13.50;
'Papier',
' ',
25.0,
7.80,
23.50;
'Ordner',
' ',
3.50,
```

Abbildung 13.1 Ausgabe des Testkalkulationsblatts als SIF-Datei.

```
5.00,
3.325;
'----------------------------------------';
'Summe',
' ',
43.50,
' ',
39.875;
```

Abbildung 13.1 Ausgabe des Testkalkulationsblatts als SIF-Datei.

Das übertragene Kalkulationsblatt enthält zehn durch Semikolons abgeschlossene Zeilen, was in der ersten Zeile der SIF-Datei spezifiziert wird. Daran schließen sich die einzelnen Sätze an.

Textverarbeitungsformate

MS-Word-Format

WinWord-Format

Windows 3.x Write-Binary-Format (WRI)

WordStar-Format

WordPerfect-Format

Rich Text-Format (RTF)

Standard Generalised Markup Language (SGML)

Hypertext Markup Language (HTML)

Das AMI Pro Format (SAM)

17 MS-WORD-Format

Das Textverarbeitungsprogramm WORD für DOS der Firma Microsoft benutzt in den Versionen 3.0, 4.0 und 5.0 ein gemischtes ASCII-/Binärformat zur Abspeicherung erstellter Textdateien und gliedert diese Dateien in drei Teile:

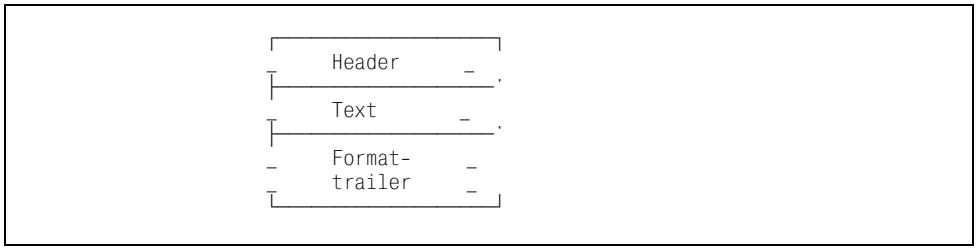


Abbildung 17.1 Aufbau einer MS-WORD-Datei

Die Daten innerhalb der Datei sind in Blöcken zu 128 Byte organisiert. Diese Information ist deswegen wichtig, da einige interne Zeiger eine Blocknummer in der Datei bezeichnen, womit sich der Offset auf das erste Byte des betreffenden Blocks mit folgender Formel berechnen läßt:

```
Offset = Blocknummer * 80H
```

Bild 17.2 enthält einen Beispieltext, der mit WORD erfaßt und abgespeichert wurde:

Falls ein Block nicht komplett mit Zeichen gefüllt werden kann, enthält das Blockende zufällig im Speicher stehende Zeichen. Diese Bytes sind dann undefiniert. Der erste Block (Nr. 0) einer MS-WORD-Datei enthält den 128 Byte langen Header mit einigen Steuerinformationen. Daran schließen sich mehrere Blöcke mit dem eigentlichen Text an, die bis auf wenige Ausnahmen keine Steuercodes enthalten. Falls kein Text abgespeichert wurde, entfallen diese Blöcke. Den Abschluß bilden mehrere Blöcke mit den Formatinformationen für den Text.

Die entsprechende WORD-Datei ist in Bild 17.3 als Speicherdump abgebildet.

In der Datei werden verschiedene Arten von Zeigern benutzt:

- 4-Byte-Pointer (*Dateizeiger*), die eine absolute Position eines Bytes als Offset vom Dateianfang angeben.
- 4-Byte-Pointer (*Textzeiger*), die eine relative Position (vom Textanfang) auf ein Zeichen im Textbereich spezifizieren. Durch Addition von 80H lassen sich diese Zeiger in Dateizeiger umwandeln.
- 2-Byte-Pointer (*Blocknummer*), die auf einen der (128 Byte) Blöcke zeigen. Dieser Wert läßt sich dann durch Multiplikation mit 80H in einen Dateipointer umrechnen.

```
Trennung weich (Alt -)
Trennung 2 (ctrl -)
Absatz Blocksatz
Fettschrift eingeschaltet mit Alt F
Normalschrift mit ALT Leertaste
Kursivschrift mit ALT I
unterstreichen mit ALT U
doppelt unterstreichen mit ALT D
Kapitälchen mit ALT K
durchgestrichen mit ALT D
hochgestellt mit ALT H
tiefgestellt mit ALT T
verborgen mit ALT G
zentrieren mit ALT Z
linksbündig mit ALT L
rechtsbündig mit ALT R
Einzug links 1,5 cm mit ALT M
Einzug rechts mit ALT V
wählbarer Abstand mit ALT O
Standardabsatz
Einzug erste Zeile negativ: Die zweite und alle folgenden Zeilen eines Absatzes
werden eingerückt.

doppelter Zeilenabstand ALT 2
Dies ist eine Überschrift ALT +
```

Abbildung 17.2 Textbeispiel für MS-Word 4.0

```
                < Header >
 Signatur                          Textende
┌──────────┐                              ┌────┐
31 BE 00 00 00 AB 00 00-00 00 00 00 00 00 17 03
          <    Block_pointer    >
└────┘ └────┘ └────┘ └────┘ └────┘ └────┘ └────┘ └────┘
00 00 09 00 0C 00 0C 00-0C 00 0C 00 0C 00 43 3A
                 Druckformatedatei -->  C  :
5C 54 45 58 54 45 5C 53-54 41 4E 44 41 52 44 2E
 \  T  E  X  T  E  \  S  T  A  N  D  A  R  D  .
```

Abbildung 17.3 Speicherdump einer MS-WORD 4.0-Datei

```
44 46 56 00 00 00 00 00-00 00 00 00 00 00 00 00
 D  F              V  .  .  .
...        <    Druckertreiber   >
00 00 45 50 53 45 58 00-00 00 0D 00 00 00 00 00
                E  P  S  E  X  .  .
00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
          <    Textbereich      >
54 65 73 74 20 54 65 78-74 20 6D 69 74 20 57 6F
 T  e  s  t     T  e  x  t     m  i  t     W  o
72 64 20 53 74 65 75 65-72 7A 65 69 63 68 65 6E
 r  d     S  t  e  u  e  r  z  e  i  c  h  e  n
2E 0D 0A 0D 0A 54 72 65-6E 6E 75 6E 67 20 77 65
              T  r  e  n  n  u  n  g     w  e
69 63 68 20 28 41 6C 74-20 2D 29 0D 0A 54 72 65
 i  c  h     (  A  l  t     û  )        T  r  e
6E 1F 6E 75 6E 67 20 32-20 28 63 74 72 6C 20 2D
 n  .  n  u  n  g     2     (  c  t  r  l     -
29 0D 0A 41 62 73 61 74-7A 20 42 6C 6F 63 6B 73
 )  .  .  A  b  s  a  t  z     B  l  o  c  k  s
61 74 7A 0D 0A 46 65 74-74 73 63 68 72 69 66 74
                .....
                .....
                .....
65 73 20 69 73 74 20 65-69 6E 65 20 9A 62 65 72
 e  s     i  s  t     e  i  n  e     ≡  b  e  r
73 63 68 72 69 66 74 20-41 4C 54 20 2B 0D 0A 0D
 s  c  h  r  i  f  t     A  L  T     +
0A 0D 0A 0D 0A 0D 0A 0D-0A 0D 0A 0D 0A 0D 0A 0D
0A 0D 0A 0D 0A 0D 0A 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
                          ┌ Textende
20 20 20 20 20 20 20 20-00 00 00 00 00 00 00 00
00 00 00 00 4E 00 22 08-2A 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00-FF 00 00 00 00 00 00 00
00 00 25 00 00 00 00 00-3D 00 00 00 00 00 17 00

   < Blöcke mit den Formatinformationen >
      < Block mit den Zeichenformaten>
80 00 00 00 E5 00 00 00-FF FF 09 01 00 00 78 00
2A 01 00 00 FF FF 43 01-00 00 75 00 5D 01 00 00
70 00 7F 01 00 00 6B 00-96 01 00 00 66 00 B1 01
00 00 61 00 C9 01 00 00-5A 00 E1 01 00 00 53 00
59 02 00 00 FF FF 72 02-00 00 4E 00 11 03 00 00
FF FF 04 00 00 18 80 06-00 00 18 00 00 F4 06 00
00 18 00 00 0C 04 00 00-18 02 04 00 00 18 30 04
00 00 18 04 04 00 00 18-01 02 00 02 02 00 01 0D
            < Absatzformate >
11 03 00 00 13 03 00 00-78 00 17 03 00 00 FF FF
2A 01 00 00 FF FF 43 01-00 00 75 00 5D 01 00 00
70 00 7F 01 00 00 6B 00-96 01 00 00 66 00 B1 01
00 00 61 00 C9 01 00 00-5A 00 E1 01 00 00 53 00
59 02 00 00 FF FF 72 02-00 00 4E 00 11 03 00 00
FF FF 04 00 00 18 80 06-00 00 18 00 00 F4 06 00
00 18 00 00 0C 04 00 00-18 02 04 00 00 18 30 04
00 00 18 04 04 00 00 18-01 02 00 02 02 00 01 02
        < Bereichsformate Block 1 >
80 00 00 00 A3 00 00 00-FF FF A5 00 00 00 FF FF
BD 00 00 00 FF FF D3 00-00 00 FF FF E5 00 00 00
```

Abbildung 17.3 Speicherdump einer MS-WORD 4.0-Datei

```
78 00 09 01 00 00 78 00-2A 01 00 00 78 00 43 01
00 00 78 00 5D 01 00 00-78 00 7F 01 00 00 78 00
96 01 00 00 78 00 B1 01-00 00 78 00 C9 01 00 00
78 00 E1 01 00 00 78 00-F6 01 00 00 6F 00 0C 02
              Pointer Folgeblock ──────┘
00 00 66 00 0C 04 00 00-18 02 08 3C 01 1E 00 00
00 C5 02 08 3C 03 1E 00-00 00 C5 02 02 3C 03 10
             < Folgeblock >
0C 02 00 00 22 02 00 00-72 00 3A 02 00 00 69 00
59 02 00 00 67 00 72 02-00 00 67 00 8F 02 00 00
59 00 9F 02 00 00 FF FF-A1 02 00 00 FF FF BD 02
00 00 4E 00 BF 02 00 00-FF FF DE 02 00 00 41 00
96 01 00 00 78 0C 3C 00-1E 00 00 00 00 00 00 00
E0 01 0A 3C 00 1E 00 00-00 C5 02 3B FD 0D 3C 00
1E 00 00 00 00 00 00 00-F0 00 F0 01 3C 08 3C 02
1E 00 00 00 C5 02 08 3C-00 1E 00 00 00 C5 02 0A
DE 02 00 00 FF 02 00 00-6E 00 01 03 00 00 6E 00
03 03 00 00 6E 00 05 03-00 00 6E 00 07 03 00 00
6E 00 09 03 00 00 6E 00-0B 03 00 00 6E 00 0D 03
00 00 6E 00 0F 03 00 00-6E 00 11 03 00 00 65 00
13 03 00 00 5C 00 15 03-00 00 59 00 17 03 00 00
FF FF 18 03 00 00 FF FF-00 C5 02 3B FD 02 3C 03
08 3C 03 1E 00 00 00 C5-02 08 3C 01 1E 00 00 00
C5 02 0C 3C 00 1E 00 00-00 00 00 00 00 E0 01 0E
             < Info-Block>
12 00 13 00 14 00 15 00-16 00 17 00 18 00 20 00
28 00 00 00 00 00 00 00-31 2E 31 2E 39 30 20 20
 (                      1  .  1  .  9    0
31 2E 31 2E 39 30 20 20-97 02 00 00 00 00 00 00
               1  .  1  .  9  0
00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

Abbildung 17.3 Speicherdump einer MS-WORD 4.0-Datei

Der genaue Aufbau der drei Abschnitte (Header, Text, Formate) innerhalb einer WORD-Datei wird nachfolgend beschrieben.

Der WORD-Header (Versionen 3.0, 4.0, 5.0)

Wie in Bild 17.3 zu sehen ist, führt WORD in den ersten 128 Byte (Block 0) die Headerinformationen. Die ersten 4 Bytes der Datei enthalten immer die Hexcodes *31 BE 00 00*. Vermutlich benutzt WORD diese Bytes als Signatur für formatierte Dateien. Tabelle 17.1 enthält die detaillierte Aufteilung des Headers:

Offset	Bytes	Bedeutung
00H	4	WORD-Signatur 31 BE 00 00H
04H	8	reserviert (00 AB 00 00 00 00 00 00)

Tabelle 17.1 Format des WORD-Headers

Offset	Bytes	Bedeutung
0EH	4	Dateizeiger auf 1. Zeichen hinter dem Textbereich
12H	2	Blocknummer (Ptr) mit den Absatzformaten
14H	2	Blocknummer (Ptr) mit den Fußnotentabellen
16H	2	Blocknummer (Ptr) mit den Bereichsformaten
18H	2	Blocknummer (Ptr) mit der Bereichstabelle
1AH	2	Blocknummer (Ptr) mit der Tabelle Seitenumbruch
1CH	2	Blocknummer (Ptr) mit Kurzinfotext (Autor, Datum etc.)
1EH	66	Dateiname des Druckformats als ASCII-Zeichenstring
60H	2	Flag (reserviert für Windows Write)
62H	8	Name des Druckertreibers als ASCII-String
6AH	2	Zahl der benutzten Blöcke in der Datei
6CH	2	Bitfeld für überarbeitete Textteile
6EH	18	reserviert (in Version 4.0 immer 00); ab 5.0 mit unbekannten Codes belegt

Tabelle 17.1 Format des WORD-Headers

Alle Angaben in der Spalte »Bytes« erfolgen im Dezimalsystem. Im Header findet sich ab Offset 0EH ein 4-Byte-Pointer (*Dateizeiger*), der auf das erste unbelegte Zeichen hinter dem Text zeigt. Der Wert ist als Offset vom Dateianfang bis zu dem betreffenden Byte zu interpretieren. Durch Subtraktion der Zahl 80H (Länge Block 0) erhält man die Zahl der Zeichen im Text. Die folgenden 6 Wörter enthalten 2-Byte-Zeiger, die als *Blocknummern* interpretiert werden. In den spezifizierten Blöcken finden sich weitere Informationen zur Textformatierung. Durch Multiplikation mit 80H läßt sich aus der Blocknummer ein Dateizeiger auf das erste Byte des Blocks berechnen. (Der Aufbau der Formatblöcke wird in einem eigenen Abschnitt behandelt.)

Ab Offset 1EH steht der *Pfad mit Laufwerk und Dateinamen* der Druckformatvorlage. Dieser Text wird als ASCIIZ-String gespeichert, d.h. das letzte Zeichen ist mit 00H abgeschlossen. In MS-DOS darf der Pfad maximal 65 Zeichen umfassen, weshalb im Header 66 Zeichen reserviert werden. Nicht benutzte Bytes müssen mit 00H überschrieben werden.

Ab Offset 62H findet sich ein Feld mit 8 Byte, das den Namen des Druckertreibers aufnimmt. Falls der Name kürzer als 8 Zeichen ist, müssen die restlichen Stellen mit 00H aufgefüllt werden. Laufwerk, Pfad und Erweiterung des Treibers können nicht mitspezifiziert werden.

Das Feld ab Offset 6AH gibt die Anzahl der Blöcke an, die relevante Informationen enthalten. Es kann durchaus sein, daß eine WORD-Datei weitere Blöcke enthält. Diese sind aber meist mit Nullbytes gefüllt und dürfen nicht interpretiert werden.

Das Wort ab Offset 6CH wird als 16-Bit-Feld interpretiert. Es dient zur Aufnahme der Formatkodierung überarbeiteter Textteile und ist in der Regel mit 00 00H besetzt. Erst wenn ein Text mit dem Kommando *FORMAT/Überarbeitung* modifiziert wird, speichert WORD die gesetzten Einstellungen in den einzelnen Bits (siehe Tabelle 17.2).

Bit	Bedeutung
0	Formatleiste: 1 = Ja, 0 = Nein
3...1	Darstellung eingefügter Text 000 = unterstrichen 001 = Großbuchstaben 010 = Normaldarstellung 011 = Fettschrift 100 = -- 101 = -- 110 = doppelt unterstrichen 111 = --
5...4	Position der Korrekturleiste 00 = keine Korrekturleiste 01 = Korrekturleiste links 10 = Korrekturleiste rechts 11 = Korrekturleiste abwechselnd
15...6	reserviert bis zur Version 4.0, ab 5.0 mit unbekannten Codes belegt

Tabelle 17.2 Kodierung FORMAT/Überarbeitung in WORD 4.0

Bis zur Version 4.0 sind die Bits 6 bis 15 nicht belegt. Ab WORD 5.0 werden die Bits 6 und 7 benutzt; ihre Bedeutung ist jedoch nicht bekannt. Die restlichen 18 Byte im Header sind reserviert und enthalten die Werte 00H. Ab Version 5.0 des Programms findet man hier einige Zeiger, deren Bedeutung jedoch nicht bekannt ist.

Der WORD-Textteil

Ab Offset 80H beginnt in den WORD-Versionen 3.0 bis 5.0 der Block 1 mit dem abgespeicherten Text im ASCII-Format. Dieser Text kann durchaus über mehrere Blöcke reichen. Im letzten »Textblock« ist der Bereich vom letzten gültigen Textzeichen bis zum Blockende mit zufälligen Werten gefüllt. Das Textende ist im Header vermerkt (Offset 0EH). Sobald WORD ein leeres Textfenster speichert, entfällt der Textblock, und WORD fügt die Formatinformationen direkt an den Header an.

Der Text enthält nur wenige Zeichen, die als Steuercodes interpretiert werden können. Tabelle 17.3 enthält eine Aufstellung einiger Codes.

Die Codes 1 bis 5 markieren die von WORD angelegten Textbausteine. Fußnoten werden nur dann im Text markiert, wenn der Benutzer kein Fußnotenkennzeichen angibt. Sobald der Fußnote eine Nummer zur automatischen Verwaltung zugeordnet wird, speichert

WORD diese Fußnote ganz normal im Text. Die Informationen über die Fußnotenformatierung werden dann in einem eigenen Block im Anhang geführt.

Code	Bedeutung
01H	Textbaustein Seite
02H	Textbaustein Druckdatum
03H	Textbaustein Druckzeit
04H	reserviert
05H	Fußnote ohne Fußnotenkennzeichen
09H	Tabulatorzeichen
0BH	Zeilenumbruch
0CH	Seitenwechsel (Form Feed)
0D,0AH	CR/LF als Absatzende
1FH	bedingter Trennstrich
C4H	geschützter Trennstrich
FFH	geschütztes Leerzeichen

Tabelle 17.3 Interpretation verschiedener Steuercodes in WORDInterpretation verschiedener Steuercodes in WORD

Die Zeichen CR/LF (*Carriage Return/Line Feed*) markieren in WORD ein Absatzende. Der Import normaler ASCII-Dateien ist in WORD also durchaus möglich, da viele Editoren hinter jeder Zeile ein CR/LF ablegen. Sämtliche Absatzkommandos aus WORD beziehen sich bei diesen eingelesenen Texten dann aber ebenfalls nur auf einzelne Zeilen, da WORD diese als Absatz interpretiert; die CR/LF-Zeichen am Zeilenende müssen deshalb gegebenenfalls entfernt werden.

Der ASCII-Code 31 (1FH) wird in WORD zur Markierung möglicher Trennstellen benutzt. Mit dem Wert 255 (FFH) läßt sich ein Wortzwischenraum für Trennungen sperren; weitere Steuercodes sind mir nicht bekannt.

Der Formatbereich in WORD

Hinter dem letzten Textblock beginnt der Bereich, in dem WORD die Informationen über Textformatierungen ablegt. Hierbei werden mehrere Bereiche unterschieden:

- Blöcke mit Zeichenformaten
- Blöcke mit Absatzformaten
- Blöcke mit der Fußnotentabelle
- Blöcke mit den Bereichsformaten
- Blöcke mit der Bereichstabelle
- Blöcke mit der Seitenumbruchtabelle

▶ Blöcke mit Informationen des Dateimanagers

Jeder dieser Bereiche kann sich über mehrere Blöcke zu je 128 Byte erstrecken. Im Header der Word-Datei finden sich ab Offset 12H die Nummern der entsprechenden Blöcke mit den Formatinformationen (Bild 17.4).

Interpretation verschiedener Steuercodes in WORD

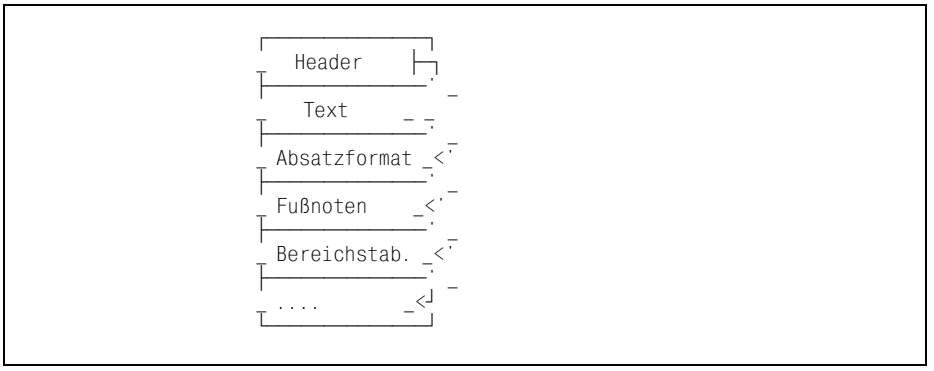


Abbildung 17.4 Zeiger auf die Formatbereiche

Die Zeichenformate

Der erste Block hinter dem Text enthält die Zeichenformate. WORD verzichtet darauf, im Header einen eigenen Zeiger auf diesen Block zu definieren. Die Position innerhalb der Datei läßt sich jedoch durch den Textzeiger ab Offset 0EH bestimmen. Falls der Textbereich fehlt, beginnt die Beschreibung der Zeichenformate ab Block 1. WORD benutzt eine recht ausgefeilte Technik zum Abspeichern der jeweiligen Zeichenformate: Die Zahl der möglichen Kombinationen zur Zeilenformatierung (fett, kursiv etc.) steht von Anfang an fest. WORD legt in einer Tabelle die möglichen Formatspezifikationen ab. Nun muß nur noch vermerkt werden, wie die einzelnen Textteile zu formatieren sind (Bild 17.5).

Der Text aus Bild 17.5 soll die in einer Hilfstabelle abgelegten Zeichenformate »normal«, »fett« und »kursiv« erhalten. Immer wenn im Text eine Fettschrift auftritt, genügt es, auf den entsprechenden Eintrag in dieser Tabelle zu verweisen. Ein Zeiger markiert dann noch den Beginn des Textes mit der Fettschrift. Eine erneute Formatspezifikation hebt anschließend die Fettschrift wieder auf. Dieses Verfahren wird in den Versionen 3.0, 4.0 und 5.0 benutzt. Die nachfolgenden Angaben beziehen sich zwar auf die Version 4.0 von WORD, dürften aber weitgehend auch für 5.0 gelten.

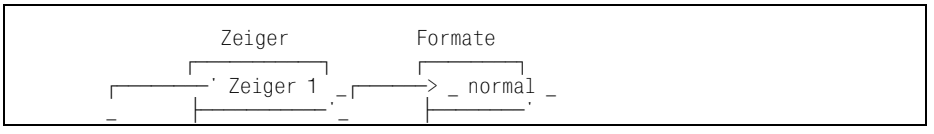


Abbildung 17.5 Markierung der Zeichenformate

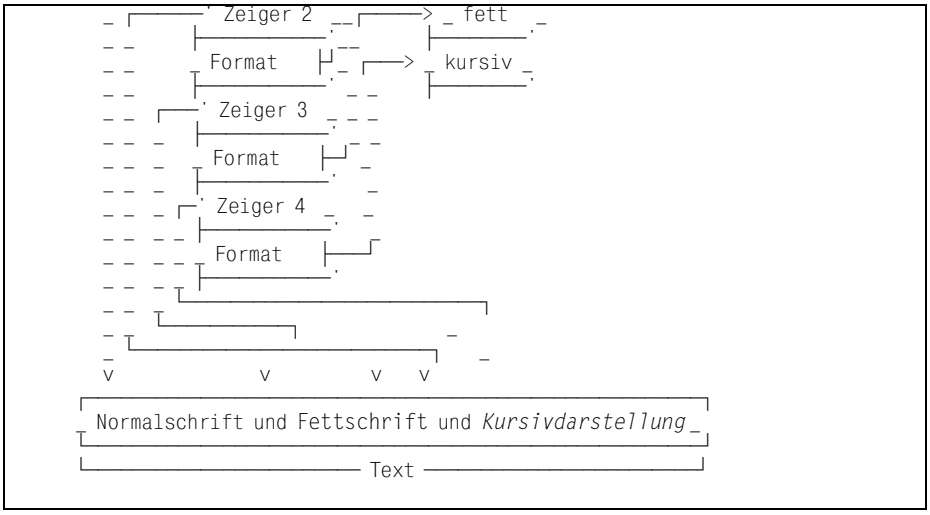


Abbildung 17.5 Markierung der Zeichenformate

WORD benutzt dieses Verfahren für die Textformatierung. Der Block mit den Zeichenformaten besitzt daher den Aufbau gemäß Tabelle 17.4.

Offset	Bytes	Bedeutung
00H	4	Zeiger auf das erste Zeichen, für das das Format gilt
Beginn der Tabelle mit den Text- und Formatzeigern:		
04H	4	Zeiger auf das erste Zeichen, für das das 1. Format nicht mehr gilt
08H	2	Zeiger in Formattabelle 1. Format
0AH	4	Zeiger auf das erste Zeichen, für das das 2. Format nicht mehr gilt
08H	2	Zeiger in Formattabelle 2. Format
...	..	....
Beginn der Hilfstabelle mit den Zeichenformaten (Formattabelle)		
...	...	....
7EH	...	letzter Formateintrag
7FH	1	Anzahl der Formatbereiche (Entries)

Tabelle 17.4 Aufbau des ZeichenformatblocksAufbau des Zeichenformatblocks

Tabelle 17.4 enthält in den ersten vier Bytes den Offset (Anfangszeiger) auf das erste Zeichen im Text, für das das Format gilt. Da dieses Zeichen immer in Block 1 steht, besitzt der Zeiger den Wert 00 00 00 80H. Man muß dabei jedoch beachten, daß in MS-DOS das unterste Byte zuerst gespeichert wird (80 00 00 00H). Ab Offset 04H enthält die Tabelle 17.4 eine Datenstruktur mit je zwei Zeigern für jeden Formatbereich:

- Textzeiger auf das erste Zeichen des folgenden Formatabsatzes
- Zeiger auf die Formatdefinition in der Hilfstabelle (Formattabelle)

Ein 4-Byte-Zeiger (*Textzeiger*) spezifiziert die Offsetadresse des ersten Zeichens, für das das angegebene Format nicht mehr gilt. Dieser Zeiger dient gleichzeitig als Anfangszeiger für die neue Formatspezifikation. Das folgende Wort der Datenstruktur ist der Zeiger auf die Formatdefinition in der Hilfstabelle (Formattabelle am Ende des Blocks). Der Wert wird als Offset vom ersten Textzeiger (Offset 04H) auf den Formateintrag in der Hilfstabelle (Bild 17.6) interpretiert.

Da während der Textbearbeitung die Zahl der zu formatierenden Abschnitte schwankt, beginnt WORD mit dem Aufbau der Hilfstabelle am Blockende (letzter Eintrag bei Offset 07EH). Jede neue Formatdefinition wird durch WORD vor dem letzten Eintrag gespeichert. Die Zahl der zu formatierenden Textbereiche und damit die Zahl der gültigen Textzeiger (ohne Anfangszeiger) steht am Blockende (Offset 7FH). Der Aufbau der Hilfstabelle mit den Formatinformationen wird nachfolgend noch genauer beschrieben.

Bei längeren Texten übersteigt die Zahl der Textzeiger den Platz in der Tabelle, was zur Folge hat, daß die Tabelle überläuft. WORD legt dann einen weiteren Block mit Zeichenformaten an und speichert die Information über das Vorhandensein dieses Folgeblocks indirekt im letzten Textzeiger. Falls dessen Wert mit der Anfangsadresse des folgenden Blocks übereinstimmt, handelt es sich um einen Folgeblock. Sobald ein solcher Folgeblock anzulegen ist, kopiert WORD den Inhalt des letzten Blocks in den Speicher und setzt die Zahl der Einträge (letztes Byte) auf Null. Dann wird der Anfangszeiger mit dem Wert des letzten gültigen Textzeigers im Vorgängerblock überschrieben. Dadurch enthält die Kopie alle Informationen des Vorgängerblocks, und WORD füllt die neue Tabelle sukzessive mit Text- und Formatzeigern auf.

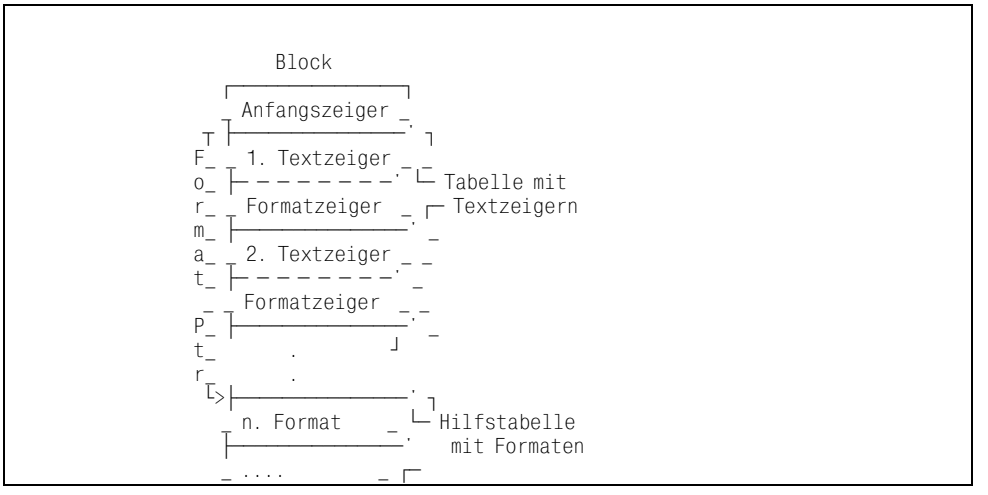


Abbildung 17.6 Lage der Hilfstabelle mit den Formatbeschreibungen

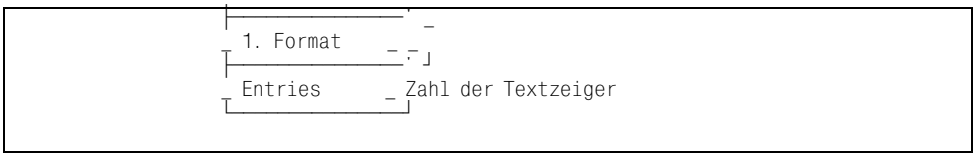


Abbildung 17.6 Lage der Hilfstabelle mit den Formatbeschreibungen

Jedem 4-Byte-Textzeiger ist in der Tabelle ein 2-Byte-Formatzeiger zugeordnet. Dieser gibt den Offset vom ersten Textzeiger auf die jeweilige Formatdefinition am Blockende an. Wird zu diesem Wert die Zahl 4 addiert, entspricht das Ergebnis dem Offset vom Blockanfang. Ein Eintrag FFFFH im Zeiger signalisiert, daß der Text im Standardformat auszugeben ist: So steht er beispielsweise hinter dem letzten gültigen Textzeiger, um auf das Standardformat zurückzuschalten. Bei Vorhandensein eines Folgeblocks steht dort die Information über die Zeichenformatierung des markierten Textbereichs. Alle Werte ungleich FFFFH sind dagegen als Zeiger zu interpretieren. Für den Aufbau der Hilfstabelle mit der Zeichenformatierung gilt folgende Datenstruktur (Tabelle 17.5):

Offset	Bytes	Bedeutung
00H	1	Zahl der Folgebytes für diesen Eintrag
01H	1	Kodierung Druckformatvorlage: Bit 0 = 1: Das Zeichen wurde mit einer Druckformatvorlage formatiert, Bits 1-7 enthalten die Varianten gemäß Tabelle 17.6
02H	1	Formatcode gemäß Bild 17.7
03H	1	Größe des Zeichensatzes in 1/2 Punkt
04H	1	Zeichenattribut gemäß Tabelle 17.7
05H	1	reserviert
06H	1	Zeichenpositionierung (hoch, tief etc.)
07H-0AH	4	reserviert

Tabelle 17.5 Aufbau des ZeichenformatsAufbau des Zeichenformats

Eine Zeichenformatierung besteht in der Regel aus mehreren Bytes. Das erste Byte gibt dabei die Zahl der Folgebytes der Definition an. Die minimale Länge einer Formatdefinition beträgt 2 Byte (1 Byte Länge, 1 Byte Format). Falls jedoch eine Information aus einem der folgenden Bytes (z. B. Zeichenpositionierung) benötigt wird, müssen alle dazwischen befindlichen Felder mitabgespeichert werden.

Das zweite Byte des Zeichenformats spezifiziert die verwendete Variante der Druckformatvorlage, nach der das Zeichen dann zu formatieren ist. Bild 17.7 gibt die Kodierung des zweiten Bytes wieder.

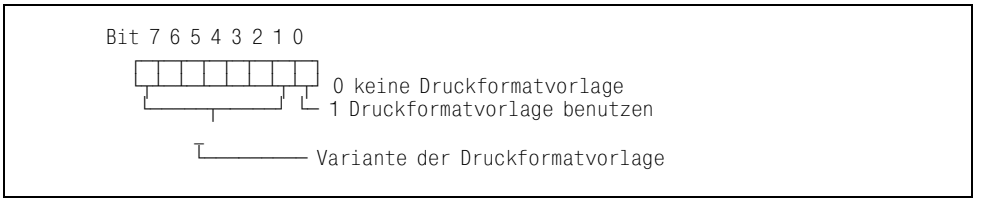


Abbildung 17.7 Kodierung der Definition für die Druckformatvorlage

Wenn das unterste Bit (Bit 0) gesetzt ist, enthalten die restlichen Bits die Nummer mit der Variante der Druckformatvorlage. Tabelle 17.6 gibt einige der im WORD-Handbuch angegebenen Vorlagen wieder:

Code	Bedeutung
0	Standardzeichen
1-12	Druckformatvorlagen Varianten 1-12
13	Verweis auf eine Fußnote
14-18	Druckformatvorlagen Varianten 13-17
19	Seitenzahl
20-27	Druckformatvorlagen Varianten 18-25
28	Kurzinformationen
29	Zeilennummern
30-64	unbenutzt bei der Zeichenformatierung

Tabelle 17.6 Varianten der DruckformatvorlagenVarianten der Druckformatvorlagen

Weitere Informationen zu diesem Thema finden Sie in der WORD-Dokumentation.

Im dritten Byte (falls vorhanden) legt WORD die Informationen über den Formataufbau (fett, kursiv, Fontnummer) ab. Die Kodierung erfolgt dabei gemäß Bild 17.8:

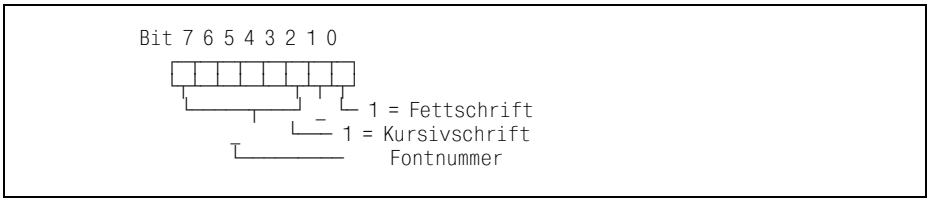


Abbildung 17.8 Kodierung des Fontformats

Die Bits 0 und 1 legen den Schriftstil (fett, kursiv) fest, während die restlichen Bits zur Auswahl der Fontnummer dienen. Die Zuordnung zwischen Zeichensatz und Nummer ist vom Druckertreiber abhängig.

Das vierte Byte spezifiziert die Zeichengröße in 1/2 Punkt. Byte 5 gibt die restlichen Attribute für die Zeichen an; die Kodierung erfolgt gemäß Tabelle 17.7:

Bits	Bedeutung
0	1 = unterstrichen
1	1 = durchstreichen
2	1 = doppelt durchstreichen
3	1 = Zeichen im Korrekturmodus einfügen
4-5	Zeichengröße 00: normal 01: Großbuchstaben 10: -- 11: Kapitälchen
6	Spezialzeichen (Seite, Datum etc.)
7	Zeichen verbergen

Tabelle 17.7 Zeichenformatattribute

Byte 6 ist bisher *reserviert*. Das gleiche gilt für die Bytes 8 bis 11. In Byte 7 wird vermerkt, ob ein Zeichen hoch- oder tiefgestellt werden soll. Dabei gilt:

Byte 7	Darstellung
00	Zeichen normal
01- 7FH	Zeichen hochgestellt
80-FFH	Zeichen tiefgestellt

Tabelle 17.8 Kodierung von Byte 7

Letztendlich entscheidet also nur Bit 7 über die Darstellung dieses Bytes.

Der Absatzformatblock

Im Header steht ab Offset 12H ein Zeiger auf den Block mit der Absatzformatierung. Die Struktur dieses Blocks stimmt mit der Struktur des Blocks zur Zeichenformatierung überein: Ein Anfangszeiger (4 Byte) spezifiziert das erste Zeichen des ersten Absatzes, das in der Regel den Textanfang darstellt. Dann beginnt die Tabelle mit jeweils einem Textzeiger (4 Byte) und dem Offset (2 Byte), der auf die Formatinformation verweist. Der Textzeiger markiert den nächsten Absatz, der Offset zur Formatinformation bezieht sich auf den ersten Textzeiger (die zugrundeliegende Struktur finden Sie in Tabelle 17.4). Das letzte Byte im Block gibt die Zahl der gültigen Einträge (Textzeiger) an. Falls der letzte gültige Textzeiger mit dem Anfangszeiger des nächsten Blocks übereinstimmt, handelt es sich um einen Folgeblock mit weiteren Absatzformaten.

Der Aufbau der Definition für die Absatzformate weicht etwas von der Definition der Zeichenformate ab – im ersten Byte steht die Zahl der Folgebytes. Tabelle 17.9 gibt den Aufbau einer Formatdefinition für einen Absatz wieder:

Offset	Bytes	Bedeutung
00H	1	Zahl der Folgebytes für diesen Eintrag
01H	1	Kodierung Druckformatvorlage: Bit 0 = 1: Das Zeichen wurde mit einer Druckformatvorlage formatiert. Die Bits 1-7 enthalten die Varianten gemäß Tabelle 17.10
02H	1	Attribute des Absatzes gemäß Tabelle 17.11
03H	1	Variante der Standard-Absatzformatierung (meist Code 30 laut Tabelle 17.10)
04H	1	Gliederungsebene und Darstellung (Bild 17.8)
05H	2	Rechter Einzug in 1/20 Punkt
07H	2	Linker Einzug in 1/20 Punkt
09H	2	Linker Einzug erste Zeile in 1/20 Punkt
0BH	2	Zeilenabstand in 1/20 Punkt
0DH	2	Anfangsabstand in 1/20 Punkt
0FH	2	Endabstand in 1/20 Punkt
11H	1	Kopf- und Fußzeile, Rahmen
12H	1	Lage der Linien für Kopf- u. Fußzeile
13H	5	reserviert (00)
17H	81	Tabelle mit Tabulatorbeschreibungen

Tabelle 17.9 Aufbau des Absatzformats in WORD

Das Byte ab Offset 01H spezifiziert die Variante der Druckformatvorlage. Ähnlich wie in Bild 17.6 markiert ein Wert 1 für Bit 0, daß der Absatz mit einer Druckformatvorlage formatiert werden soll. Bei einer nachträglichen Direktformatierung wird das Bit gelöscht, während die restlichen Bits mit dem Variantencode erhalten bleiben. Der Code in den Bits 1 bis 7 gibt die Varianten der Druckformatvorlage für die Absatzformatierung gemäß Tabelle 17.10 an:

Code	Bedeutung der Codes in den Bits 1...7
30	Standardformat Absatz
31-38	Druckformatvorlagen Absatzvarianten 1-8
39	Absatz Fußnotentext
40-87	Druckformatvorlagen Absatzvarianten 9-56
88-94	Absatz Überschriftenebene 1-7

Tabelle 17.10 Varianten der Druckformatvorlagen für Absätze

Code	Bedeutung der Codes in den Bits 1...7
95-98	Absatz Indexebene 1-7
99-102	Absatz Tabellenebene 1-7
103	Absatz Kopf-/Fußzeile

Tabelle 17.10 Varianten der Druckformatvorlagen für Absätze

Das nächste Byte ab Offset 02 definiert das Attribut zur Ausrichtung des Absatzes (links, rechts etc.). Tabelle 17.11 gibt die Kodierung dieser Attribute wieder.

Bit	Bedeutung
0...1	Absatzausrichtung 00 = linksbündig 01 = zentriert 10 = rechtsbündig 11 = Blocksatz
2	Absatz selbe Seite
3	nächster Absatz selbe Seite
4	Absatz zweispaltig
5...7	reserviert

Tabelle 17.11 Kodierung der AbsatzattributeKodierung der Absatzattribute

Jedem Absatz wird zunächst eine Standardformatierung zugewiesen. Bei der nachträglichen direkten Formatierung dieses Absatzes legt WORD im Byte ab Offset 03H die Information über das Absatzdruckformat ab (siehe Tabelle 17.10).

Das Byte ab Offset 04H spezifiziert die Gliederungsebene eines Absatzes und enthält Informationen darüber, ob der Absatz ausgeblendet werden soll. Bild 17.9 gibt die Kodierung dieses Bytes wieder:

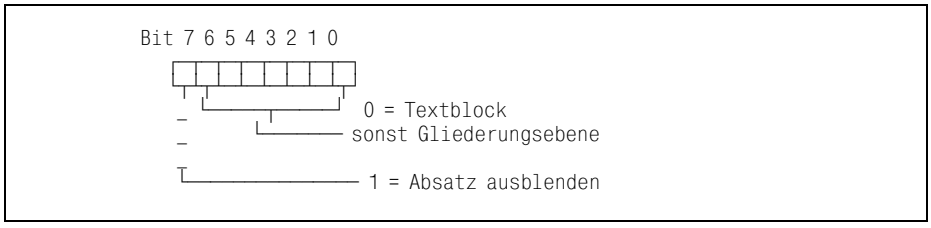


Abbildung 17.9 Kodierung der Gliederungsebenen

Die nächsten 6 Bytes geben die Einstellungen für Einzug, Zeilenabstand etc. in 1/20 Punkt Abstand wieder (siehe Tabelle 17.9). Ab Offset 11H stehen die Information über Kopf- bzw. Fußzeilen und über die Rahmen. Tabelle 17.12 gibt die Kodierung dieses Bytes wieder:

Bit	Bedeutung
0	0 = Kopfzeile 1 = Fußzeile
1	1 = Kopf-/Fußzeile auf ungeraden Seiten
2	1 = Kopf-/Fußzeile auf geraden Seiten
3	1 = Kopf-/Fußzeile auf der ersten Seite
4...5	Rahmentyp 00 = kein Rahmen 01 = Rahmen 10 = Rahmenseiten durch Linien definiert 11 = --
6...7	Art des Rahmens 00 = einfacher Rahmen 01 = doppelter Rahmen 10 = einfacher Rahmen fett 11 = --

Tabelle 17.12 Kodierung der RahmenattributeKodierung der Rahmenattribute

Falls im Byte ab Offset 11H die Bits 4 und 5 den Wert 10 besitzen, werden die Rahmenseiten durch einzelne Linien gebildet. Das Byte ab Offset 12H spezifiziert die Lage dieser Linien (Bild 17.10).

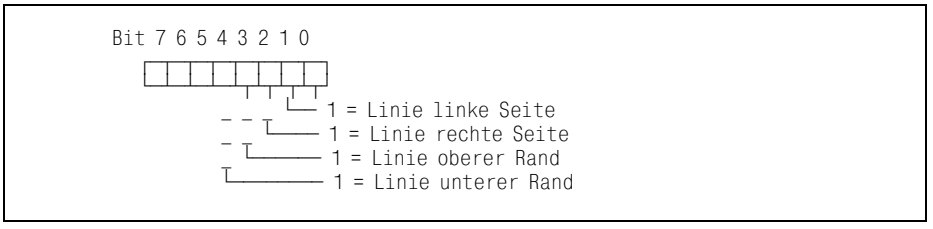


Abbildung 17.10 Kodierung eines Rahmens mit Linien

Das Ende eines Eintrages (ab Offset 17H) zur Formatbeschreibung enthält eventuell Hinweise auf Tabulatoren im Text. Für jeden Eintrag, dessen Format in Tabelle 17.13 festgehalten ist, sind 4 Byte vorgesehen.

Offset	Bedeutung
00	Abstand in 1/20 Punkt vom linken Rand

Tabelle 17.13 Kodierung der Tabulatorformate

Offset	Bedeutung
02	Attribut des Tabulators Bit 0-2: Ausrichtung 000 = linksbündig 001 = zentriert 010 = rechtsbündig 011 = ? 100 = ? 101 = ? 110 = ? 111 = ? Bit 3-5: Füllzeichen 000 = Leerzeichen 001 = Punkte 010 = Bindestriche 011 = Unterstriche Bit 6-7: reserviert
03	reserviert (00 00)

Tabelle 17.13 Kodierung der Tabulatorformate

Der letzte Eintrag der Tabulatortabelle muß nicht unbedingt 4 Byte umfassen, sondern kann auch zwischen 2 und 4 Byte enthalten, da sich die Zahl der direkt formatierten Tabulatoren aus der Länge des Formatfeldes berechnen läßt.

Das Format des Fußnotenblocks

WORD speichert Fußnoten und die zugehörigen Verweise als normale ASCII-Strings im Textbereich. Zur Verwaltung der Fußnotennumerierung bei der Druckausgabe legt das Programm dann einen eigenen Block mit den Formatinformationen an. Der Zeiger ab Offset 14H im Header gibt die betreffende Blocknummer an. Diese Fußnotentabelle existiert jedoch nicht immer: Stimmt der Wert des Vektors mit dem Eintrag für den Block mit den Bereichsinformationen (Offset 16H) überein, dann existieren keine Fußnoteninformationen. Andernfalls enthält der Block eine Tabelle, in der alle Fußnoten beschrieben werden. Diese Tabelle besitzt folgende Struktur:

Offset	Bytes	Bedeutung
00H	2	Anzahl der Fußnoten + 1 im Text
02H	2	Anzahl vorhandener Fußnoten im Text + 1 (einschließlich gelöschter Fußnoten)
Beginn der Tabelle mit der Fußnotenbeschreibung		
04H	4	Offset des Fußnotenzeichens ab Textanfang
08H	4	Offset auf den Fußnotentext ab Textanfang
...	...	.....

Tabelle 17.14 Aufbau des Blocks mit den Fußnoten

Im ersten Wort ist die Zahl der im Text vorhandenen Fußnoten + 1 vermerkt. Im darauffolgenden Wort findet sich die Zahl der maximal vorhanden Fußnoten im Text. WORD benutzt diese Information, um festzustellen, wie weit die dahinterliegende Tabelle mit den Informationen über die Fußnoten (ab Offset 04H) bereits aufgebaut war. Dies ist zum Beispiel wichtig, falls mehr als ein Block belegt ist. Für jede Fußnote wird ein 4-Byte-Textzeiger auf die Position der Fußnote und einer auf den eigentlichen Text der Fußnote abgespeichert. Das erste Zeigerpaar markiert die Anfangs- und Endadresse des letzten Fußnotentextes – deshalb wird die Zahl der Fußnoten + 1 in der Tabelle angegeben. WORD nutzt die beiden ersten Einträge zur Bestimmung der Länge des letzten Fußnotentextes.

Das Format des Bereichstabellenblocks

Ein Text läßt sich in WORD in mehrere Bereiche aufteilen. Sobald der Benutzer solche Bereiche definiert, legt WORD einen Block mit der Bereichstabelle und einen Block mit den Bereichsformaten an. Der Zeiger im Header ab Offset 16H gibt die Blocknummer mit den Bereichsformaten an, während ab Offset 18H die Blocknummer mit der Bereichstabelle steht. Falls die Blocknummern mit dem Zeiger auf die Seitenumbruchtabelle (ab Offset 1AH) übereinstimmen, existieren diese Tabellen und Formatblöcke nicht. Andernfalls speichert WORD für jeden Bereich die entsprechenden Informationen in den beiden Blöcken ab. Für die Bereichstabelle gilt folgender Aufbau:

Offset	Bytes	Bedeutung
00H	2	Anzahl der Bereiche
02H	2	Anzahl der maximal eingetragenen Bereiche
Beginn der Tabelle mit den Bereichsbeschreibungen		
04H	4	Offset des ersten Zeichens ab Textanfang, das nicht mehr zum Bereich gehört
08H	2	reserviert
0AH	2	Offset der zugehörigen Formatbeschreibung im Bereichsformatblock
...	...	.....

Tabelle 17.15 Aufbau des Blocks mit der Bereichstabelle

Im ersten Wort steht die Zahl aller vorhandenen Bereiche, während das folgende Wort die maximale Zahl der vormals angelegten Bereiche enthält. Damit kann WORD feststellen, wie weit diese Tabelle bereits aufgebaut war. Ab Offset 04H beginnt dann die eigentliche Bereichstabelle, die für jeden Bereich drei Einträge enthält: Der erste Zeiger markiert das Ende eines Bereichs, der letzte Eintrag ist als Zeiger auf die zugehörige Formatbeschreibung zu interpretieren. Hierbei wird nur der Offset vom Blockanfang (Bereichsformatblock) bis zu der Beschreibung angegeben. Der mittlere Eintrag ist in WORD 4.0 vermutlich unbenutzt.

Das Format des Bereichsformatblocks

Die Blocknummer mit den Bereichsformaten findet sich im Header ab Offset 16H. Für jeden Eintrag im Block wird folgende Struktur angelegt:

Offset	Bytes	Bedeutung
00H	1	Zahl der Folgebytes für den Eintrag
01H	1	Kodierung Druckformatvorlage Bit 0 = 1: Das Zeichen wurde mit einer Druckformatvorlage formatiert; Bits 1-7 enthalten die Varianten laut Tabelle 17.17
02H	1	Attribute des Bereichs laut Tabelle 17.17
03H	2	Seitenlänge in 1/20 Punkt
05H	2	Seitenbreite in 1/20 Punkt
07H	2	Erste Seitennummer oder FFFFH für fortlaufende Seitennumerierung
09H	2	oberer Randabstand in 1/20 Punkt
0BH	2	Länge Textfeld in 1/20 Punkt
0DH	2	linker Rand in 1/20 Punkt
0FH	2	Breite Textfeld in 1/20 Punkt
11H	1	Formatierung des Bereichs (Zeilennummern)
12H	1	Spaltenanzahl im Bereich
13H	2	Distanz Kopfzeile in 1/20 Punkt von oben
15H	2	Distanz Fußzeile in 1/20 Punkt von oben
17H	2	Distanz zwischen den Spalten in 1/20 Punkt (Durchschuß)
19H	2	Breite Bundsteg in 1/20 Punkt
1BH	2	Abstand Seitennummern in 1/20 Punkt vom oberen Blattrand
1DH	2	Abstand Seitennummern in 1/20 Punkt vom linken Blattrand
1FH	2	Abstand Zeilennummer in 1/20 Punkt vom linken Blattrand
21H	2	Intervall Zeilennummern

Tabelle 17.16 Aufbau des Bereichsformats

Für die Kodierung der Druckformatvorlagen des Bereichs gilt: Falls Bit 0 = 1, wird eine Druckformatvorlage benutzt. In diesem Fall enthalten die Bits 1 bis 7 die Variante des Druckformats gemäß Tabelle 17.17.

Code	Bedeutung
105	Standardformat Bereich
106-126	Druckformatvorlagen Bereichsvarianten 1-21

Tabelle 17.17 Varianten der Druckformatvorlagen für Bereiche

Für die Formatierung des Bereichs sind bestimmte Parameter zulässig (Format der Zeilennummern etc.). Diese Informationen speichert WORD ab Offset 02 in einem Attributbyte mit folgender Kodierung:

Bit	Bedeutung
0...2	Bereichswechsel 000 = fortlaufend 001 = Spalte 010 = Seite 011 = gerade 100 = ungerade
3...5	Darstellung Seitennummer 000 = arabische Ziffern 001 = lateinische Schreibweise groß 010 = lateinische Schreibweise klein 011 = Großbuchstaben 100 = Kleinbuchstaben
6...7	Zählung Zeilennummern ab 00 = Seitenanfang 01 = Bereichsanfang 10 = fortlaufend

Tabelle 17.18 Kodierung der BereichsattributeKodierung der Bereichsattribute

Weiterhin gibt es noch ein Byte ab Offset 11H, das die Einstellungen für Fußnotendarstellung und Zeilennumerierung enthält. Dabei gilt die Kodierung gemäß Bild 17.11:

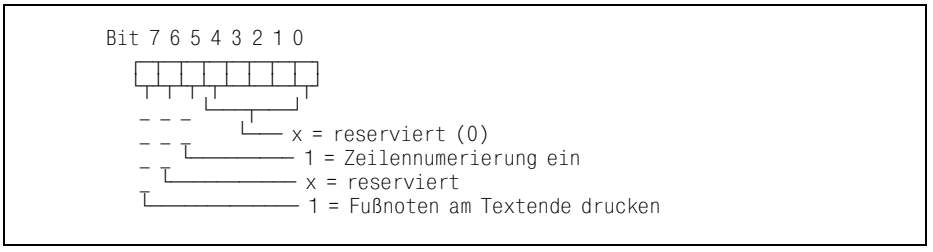


Abbildung 17.11 Kodierung der Einstellung für die Zeilennumerierung

Das Format des Seitenumbruchblocks

Ab Offset 1AH befindet sich im WORD-Header die Nummer des Blocks mit den Angaben zum Seitenumbruch. Stimmt der Eintrag mit den Blocknummern der anderen Bereiche (Offset 16H, 18H, 1CH) überein, ist die Tabelle nicht vorhanden. Tabelle 17.19 zeigt das Format des Seitenumbruchs:

Offset	Bytes	Bedeutung
00H	2	Anzahl der Umbruchabschnitte
02H	2	Anzahl der maximal vorhandenen Abschnitte
Beginn der Tabelle mit der Seitenumbruchbeschreibung		
04H	4	Offset des ersten Seitumbruchs
08H	4	Offset des zweiten Seitenumbruchs
...	...	.....

Tabelle 17.19 Aufbau des Blocks mit dem Seitenumbruch

Im ersten Wort steht die Zahl der umzubrechenden Bereiche. Ab Offset 04H beginnt die Tabelle mit der Seitenbeschreibung.

Der Info-Block des Dateimanagers

Im WORD-Header findet sich bis zur Version 5.0 ab Offset 1CH die Nummer eines Blocks, in dem der Dateimanager seine Informationen ablegt. WORD benutzt diese Funktionen zum Beispiel bei der Textrecherche und bei der Suche nach einem bestimmten Text. Der Block hat den in Tabelle 17.20 dargestellten Aufbau:

Offset	Bytes	Bedeutung
00H	2	mit 12 00H belegt
...	...	
Beginn der Informationen des Dateimanagers		
12H	40	Name des Dokuments als ASCIIZ-String mit maximal 40 Byte Länge
3AH	12	Name des Autors als ASCIIZ-String mit maximal 12 Byte Länge
46H	11	Name des Bearbeiters als ASCIIZ-String mit maximal 12 Byte Länge
51H	14	Schlüsselwort als ASCIIZ-String mit maximal 14 Byte Länge
5FH	10	Kommentar als ASCIIZ-String mit maximal 10 Byte Länge
69H	9	Versionsnummer als ASCIIZ-String mit maximal 9 Byte Länge
72H	8	Änderungsdatum im Format MM/TT/JJ als ASCIIZ-String
79H	1	00H
7AH	8	Erstellungsdatum im Format MM/TT/JJ als ASCIIZ-String
81H	1	00H
82H	4	Textgröße

Tabelle 17.20 Aufbau des Info-Blocks

Die Datumsinformationen sind im Format Monat/Tag/Jahr (z. B. 01.23.96) im ASCII-Format abgelegt. Den Abschluß bildet ein Nullbyte.

Die Informationen müssen nicht unbedingt vorhanden sein; die Felder können also auch unausgefüllt bleiben. Ab WORD 5.0 werden unbenutzte Einträge in den Blöcken mit der Signatur DCH überschrieben.

Die Informationen über das interne Speicherformat wurden bisher durch Microsoft nicht veröffentlicht. Es ist deshalb möglich, daß nicht alle Angaben der vorhergehenden Abschnitte in allen WORD-Versionen unterstützt werden.

19 Windows 3.x WRITE-Binary-Format (WRI)

WINDOWS WRITE benutzt ein Binärformat, das viele Gemeinsamkeiten mit WORD besitzt, um seine Daten zu speichern. WRI-Dateien lassen sich daher mit WORD lesen, wenn auch bestimmte Formatinformationen nicht korrekt behandelt werden. Bild 19.1 zeigt einen Auszug aus einem Speicherdump einer WRI-Datei.

```
<--- Header
31 BE 00 00 00 AB 00 00-00 00 00 00 00 00 A2 00
 1  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
00 00 03 00 04 00 04 00-04 00 04 00 04 00 00 00
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
......
<--- Textbereich
44 69 65 73 20 69 73 74-20 65 69 6E 20 54 65 73
 D  i  e  s     i  s  t     e  i  n     T  e  s
74 0D 0A 52 6F 6D 61 6E-20 38 20 50 75 6E 6B 74
 t  .  .  R  o  m  a  n     8     P  u  n  k  t
0D 0A 00 00 00 00 00 00-00 00 00 00 00 00 00 00
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .

<--- Formatangaben
80 00 00 00 93 00 00 00-77 00 A2 00 00 00 73 00
 .  .  .  .  .  .  .  .  w  .  .  .  .  .  s  .
......
02 00 0E 00 20 55 6E 69-76 65 72 73 20 28 45 31
 .  .  .  .     U  n  i  v  e  r  s     (  E  1
29 00 07 00 10 52 6F 6D-61 6E 00 00 00 00 00 00
 )  .  .  .  .  R  o  m  a  n  .  .  .  .  .  .
...
```

Abbildung 19.1 Das Windows WRITE-Format

Die Datei wird in Blöcke zu 128 Byte aufgeteilt. Diese Blöcke werden zu drei Gruppen zusammengefaßt:

- Header
- Textbereich
- Formatierung

Der Aufbau der einzelnen Gruppen wird nachfolgend kurz dargestellt.

Der WRITE-Header

Der Header umfaßt 128 Byte und lehnt sich stark an die Struktur des MS-WORD-Headers an. Hier finden sich die Signaturen für gültige WRI-Dateien und Zeiger in den Text. Tabelle 19.1 enthält eine Aufstellung der Headerstruktur.

Offset	Bytes	Name	Bedeutung
00H	2	wIdent	Signatur (muß 31 BEH sein)
02H	2	dty	Signatur (muß 00 00 sein)
04H	2	wTool	Signatur (muß 00 ABH sein)
06H	8		reserviert (00 00 00 00 00 00 00 00)
0EH	4	fcMac	Zeiger auf erstes Zeichen hinter dem Text
12H	2	pnPara	Block Nummer Absatzformate
14H	2	pnFntb	Block Nummer Fußnotentabelle
16H	2	pnSep	Block Nummer Bereichsformate
18H	2	pnsetb	Block Nummer Bereichstabelle
1AH	2	pnPgtb	Block Nummer Seitenumbruch
1CH	2	pnFfnt	Infotext
20H	66	-----	reserviert (für WORD)
60H	2	pnMac	Blockzähler im Dokument
62H	2	-----	reserviert (00 00)
64H	28	-----	unbelegt (00 00)

Tabelle 19.1 WRITE-Header

Die ersten 14 Bytes entsprechen der Signatur von WORD und sind immer mit festen Werten belegt. Die folgenden Einträge im Header beziehen sich auf das Textdokument. In »fcMac« (Offset 0EH) findet sich ein 4-Byte-Zeiger mit dem Offset hinter das letzte gültige Zeichen im Text. Die Zahl der Textbytes läßt sich damit zu fcMac-128 berechnen.

Die folgenden Einträge geben Blocknummern an. Der absolute Offset in Byte vom Dateianfang zum Beginn eines Blocks läßt sich dann zu:

```
Blocknummer * 128
```

berechnen. Bei einer Blocknummer von 10 befindet sich der Text ab Offset 1280. Ab Offset 12H findet sich ein Zeiger auf den Block mit den Absatzformaten (Paragraph Info). Die Blocknummer mit den Informationen bezüglich der Zeichenformate (Character Info) schließt sich hinter dem letzten Textblock an (siehe auch Beschreibung des WORD-Formats) und läßt sich aus der Textlänge berechnen ((Textlänge + 127)/128).

Im Wort ab Offset 14Hwl steht der Zeiger auf den Block mit den Fußnotenformaten. Offset 16H enthält einen Zeiger auf den Block mit den Bereichsformaten. Die Bereichstabelle wird ab Offset 18H als Blocknummer geführt. Die Seitenumbrüche finden sich ebenfalls in einem eigenen Block, der ab Offset 1AH im Header verwaltet wird.

Ab Offset 1CH verwaltet WORD den Zeiger auf den Infoblock bzw. die Zahl der Blöcke in der Datei. Das Wort ab Offset 60H wird dann auf 0 gesetzt. Deshalb wird das Wort ab Offset 60H zur Unterscheidung von WORD- und WRITE-Dateien verwendet. Ist der Wert von Offset 60H = 0 und der Inhalt von Offset 62H ungleich 0, liegt eine WORD-Datei vor. Andernfalls ist der Text im WRI-Format gespeichert.

Die restlichen Bytes des Headers bis 7FH sind unbelegt und werden zu 0 gesetzt.

Der Text- und Bildbereich

Ab Offset 128 (80H) beginnt der Textbereich. Dieser wird ebenfalls in Blocks zu 128 Byte aufgeteilt. Das Ende des Textes (Offset ab Dateianfang) wird im Header durch den Zeiger »fcMac« angegeben. Der letzte Block wird dann bis zur 128-Byte-Grenze mit Füllbytes ergänzt.

Wichtig ist, daß die Zeichen im Textbereich (im Gegensatz zu WORD) nicht als ASCII-, sondern als ANSI-Zeichen abgelegt werden. WINDOWS nutzt den ANSI-Zeichensatz. Dieser ANSI-Zeichensatz stimmt in den unteren 128 Zeichen weitgehend mit dem ASCII-Zeichensatz überein. Abweichungen gibt es lediglich bei den Codes oberhalb 128, z. B. bei den Sonderzeichen (Ä, Ö, Ü, etc.).

Weiterhin sind folgende Formatinformationen im Text zu berücksichtigen:

- Absätze werden am Ende mit einem »harten« RETURN (Code 0DH, 0AH) abgeschlossen. Dies ist die Sequenz, die durch Betätigung der RETURN-Taste erzeugt wird.
- Alle festen Seitenumbrüche werden durch das Zeichen Form Feed (Code 0CH) im Text markiert.
- Zeilenumbrüche oder Worttrennungen werden nicht im Text gespeichert, sondern direkt von WRITE bei der Ausgabe vorgenommen.
- Tabulatoren im Text werden mit dem Zeichen 09H markiert.

Die Informationen über die Formatierung des Textes finden sich im Anhang der Datei.

Bilder im Textbereich

WRITE kann innerhalb des Textes Bilder direkt abspeichern. Ein Bild wird innerhalb des Textes wie ein Absatz als Bytesequenz gespeichert. Dabei erhält man durch die Formatanweisung die Information, daß es sich um ein Bild handelt. Hier wird für jedes Bild ein

spezielles Bit in der Paragraph-Formatierung (PAP) gesetzt (siehe unten). Die Bytesequenz für ein Bild besitzt dabei folgende Struktur:

Offset	Bytes	Bedeutung
00H	8	METAFILEPICT Struktur (hMF Feld undefiniert)
08H	2	Offset Bild linker Rand (in twips = 1/1440 inch)
0AH	2	Bildbreite in twips
0CH	2	Bildhöhe in twips
0EH	2	Zahl der folgenden Bytes (auf 0 gesetzt)
10H	14	Zusatzinformationen für Bitmap-Bilder
1EH	2	Zahl der Bytes im Header (cbHeaderNumber)
20H	4	Zahl der folgenden Bytes (WMF oder BMP)
24H	2	Skalierungsfaktor (x)
26H	2	Skalierungsfaktor (y)
28H	x	Füllbytes (siehe cbHeaderNumber)
..H	x	Bilddaten als Bitmap oder Metafile

Tabelle 19.2 Aufbau eines Bildes in WRITE

Das Bild ist dabei entweder als WINDOWS-Bitmap (BMP) oder als WINDOWS-Metafile (WMF) gespeichert. Die Länge des Headers wird durch das Wort ab Offset 1EH angegeben. An den Header schließen sich die Bilddaten an. Ist das Wort ab Offset 00H auf 99 (dezimal) gesetzt, liegen die folgenden Bilddaten im Bitmap-Format vor. Diese Markierung in METAFILEPICT wird nur durch WRITE genutzt. Ist der Wert ungleich 99, liegen die Daten im Metafile-Format vor.

Die Kodierung dieser Formate wird in den folgenden Abschnitten beschrieben.

Das Wort ab Offset 0EH ist ab Windows 3.x unbenutzt und wurde durch die Längenangabe ab Offset 20H ersetzt. Dieses Feld definiert, wie viele Byte das Bild enthält. Sofern das Bild noch nicht umskaliert wurde, enthalten die Einträge ab Offset 24H eine Skalierung von 1000 (dezimal) was einer 100 %igen Skalierung entspricht.

OLE-Objekte im Textbereich

Ab Windows 3.1 besteht die Möglichkeit, OLE-Objekte (Bilder, Klänge, Videos etc.) als Objekte in Write-Dateien einzubinden. In diesem Fall wird anstelle des Bild-Headers ein OLE-Header verwendet. Dieser besitzt folgende Struktur:

Offset	Bytes	Bemerkungen
00H	2	OLE-Signatur (0E4H)

Tabelle 19.3 Struktur eines OLE-Objekts in WRITE

Offset	Bytes	Bemerkungen
02H	4	unbenutzt
06H	2	Objekt Typ (1=Static, 2=Embedded, 3=Link)
08H	2	Offset Bild linker Rand (in twips = 1/1440 Zoll)
0AH	2	Horizontale Größe in twips
0CH	2	Vertikale Größe in twips
0EH	2	unbenutzt
10H	4	Bytes Objektdaten, die dem Header folgen
14H	4	unbenutzt
18H	4	Hexadezimal-Nummer, die bei Konvertierung zum 8-Digit-String den Objektnamen darstellt
1CH	2	unbenutzt
1EH	2	Bytes im Header
20H	4	unbenutzt
24H	2	Skalierungsfaktor (x)
26H	2	Skalierungsfaktor (y)
28H	x	Objekt Inhalt

Tabelle 19.3 Struktur eines OLE-Objekts in WRITE

Der Formatbereich

Hinter dem Textbereich schließen sich verschiedene Blocks mit den Formatinformationen zum Text an. Der erste Formatblock beginnt immer an einer 128-Byte-Grenze. Die Blöcke mit den Zeichen- und Absatzformaten besitzen einen identischen Aufbau. Jeder Block ist nach folgendem Format gespeichert:

Offset	Bytes	Bedeutung
00H	4	Bytenummer (Offset) in der Datei mit dem ersten zu formatierenden Zeichen
04H	n*4	Feld mit Formatdescriptoren (rgfod)
..H	x	Gruppe von Formatbereichen (grpfprop)
7FH	1	Zahl der Formatdescriptoren (FOD) der Seite

Tabelle 19.4 Aufbau des Blocks mit Zeichen- und Absatzformaten

Die einzelnen Blöcke beginnen mit einem Zeiger auf das erste Zeichen im Text, für welches die Formate gelten. Die Zeiger beziehen sich dabei immer auf den Offset vom Fileanfang. Daran schließt sich (ab Offset 04H) ein Feld mit Formatbeschreibungen an.

Dieses Feld enthält für jeden Eintrag folgende Datenstruktur:

Offset	Bytes	Bedeutung (FOD)
00H	4	Bytenummer, ab dem Format nicht mehr gilt
04H	2	Offset in Formatbeschreibung (FPROP)

Tabelle 19.5 Aufbau der Formatdescriptoren (FOD)

Eine Formatinformation bezieht sich dann auf alle Zeichen (vom Anfangszeiger), bis eine neue Formatdefinition gefunden wird. Der Anfang der neuen Definition findet sich jeweils in der FOD in den ersten vier Bytes. Das folgende Wort enthält einen Zeiger mit dem Offset (ab Beginn des Felds mit den Formatdescriptoren) zur Formatbeschreibung (Format Property, FPRO) im Formatblock. Der Eintrag FFFFH bedeutet, daß keine gültige Formatbeschreibung existiert. Dies ist ein variabler Bereich, der dynamisch bei der Formatierung aufgebaut wird. Der erste Eintrag im Bereich mit der Formatbeschreibung beginnt dabei am Blockende (Offset 7EH). Das letzte Byte im Block (Offset 7FH) enthält einen Zähler, in dem die Zahl der Einträge in der Formatdescriptoren-(FOD)-Tabelle geführt wird. Ist ein Block voll, wird ein neuer Block mit der obigen Struktur angelegt. Der Bereich mit der Formatbeschreibung besitzt dabei folgende Struktur:

Offset	Bytes	Bedeutung (FPROP)
00H	1	Zahl der Bytes der Formatbeschreibung
01H	x	Formatbeschreibung

Tabelle 19.6 Aufbau der Formatbeschreibung (FROP)

Die Einträge im Feld Formatbeschreibung werden dann in verschiedene Gruppen unterteilt:

Zeichenformat (Chararacter Property, CHP)

Dieser Block mit der Formatbeschreibung beginnt an einer 128-Byte-Grenze und besitzt folgenden Aufbau:

Offset	Bytes	Bedeutung
00H	1	reserviert (WRITE ignoriert den Wert)
01H	1	Zeichenformat Bit 0: Bold (fett) 1: Italic (kursiv) 2..7: Fontcode (low Bits Index in FFNTB)
02H	1	Fontgröße in half points (Standard ist 24)

Tabelle 19.7 Formatbeschreibung Zeichen

Offset	Bytes	Bedeutung
03H	1	Bit 0: Zeichen unterstreichen 1-5: reserviert 6: gesetzt nur für (page) 7: reserviert
04H	1	Bit 0-2: Fontcode (high Bits) 3-7: reserviert
05H	1	Position des Zeichens 0 normal 1..127 superscript (hochgestellt) 128..255 subscript (tiefgestellt)

Tabelle 19.7 Formatbeschreibung Zeichen

Der Standard CHP benutzt Byte 0 = 1, Byte 2 = 24 und belegt alle anderen Byte mit 0.

Absatzformat (Paragraph Property, PAP)

Die Formatbeschreibung für Absätze besitzt folgenden Aufbau:

Offset	Bytes	Bedeutung
00H	1	reserviert (muß 0 sein)
01H	1	Bit 0-1: Justierung 0 = links 1 = zentriert 2 = rechts 3 = beide Bit 2-7: reserviert (muß 0 sein)
02H	1	reserviert (muß 0 sein)
03H	1	reserviert (muß 0 sein)
04H	2	rechter Einzug (1/20 Punkt)
06H	2	linker Einzug (1/20 Punkt)
08H	2	Einzug erste Zeile links (relativ zum linken Einzug)
0AH	2	Zwischenraum Zeilen (Standard 240)
0CH	2	reserviert (0)
0EH	2	reserviert (0)

Offset	Bytes	Bedeutung
10H	1	Page Flag Bit 0: 0 = Header, 1 = Footer 1-2: reserviert (0 = Absatz, sonst Fuß- oder Kopftext) Bit 3: 1 = Ausgabe beginnt auf erster Seite Bit 4: Absatztyp (0 = Bild, 1 = Text) 5-7: reserviert (0)
11H	5	reserviert (müssen 0 sein)
16H	14*4	bis zu 14 TAB-Descriptoren

Tabelle 19.8 Absatzformat (PAP)

Die Beschreibung für die Tabulatorenposition (TAB-Descriptor) im Absatzformat besitzt dabei folgenden Aufbau:

Offset	Bytes	Bedeutung
00H	2	Tab-Position vom linken Rand (1/20 pt)
02H	1	Flags Bit 0-2: 0 = normale Tabs, 3 = dezimale Tabs Bit 3-5: reserviert Bit 6-7: reserviert (müssen 0 sein)
03H	1	reserviert (wird ignoriert)

Tabelle 19.9 Format der Tabulatorpositionen

Eine Formatbeschreibung für einen Standardabsatz besitzt in Byte 0 den Wert 61, in Byte 2 den Wert 30 und in Byte 10 bis 11 den Wert 240. Alle anderen Bytes sind zu 0 gesetzt.

Als Unterschied zwischen Absatz- und Zeichenformat ist allerdings die Tatsache zu sehen, daß für jeden Absatz eine Formatbeschreibung vorliegen muß. Bei der Zeichenformatierung kann durchaus ein längerer Text (mit mehreren Absätzen) im gleichen Format gespeichert werden. Dann genügt eine Formatbeschreibung für die Zeichen. In der Formatbeschreibung dürfen »keine Löcher« auftreten, d.h. sie beginnt mit dem ersten Zeichen ab Offset 128 und endet beim letzten gültigen Zeichen.

In WRITE-Files finden sich alle Kopf- und Fußtexte am Beginn des Dokuments. Die Texte sind als Paragraph definiert. Falls WRITE Dateien liest, die durch WORD erzeugt wurden, kann WRITE deshalb nur Kopf- und Fußzeilen am Textanfang erkennen. Folgen weitere Definitionen im Text, werden diese als normaler Text beschrieben.

Fußnoten und Bereichsabschnitte (Sections) werden durch WRITE nicht in einer Fußnotentabelle verwaltet. WRITE legt nur eine Sektion für ein Dokument an.

Section Property

Das Format für eine Section (Section Property, SEP) entspricht folgender Tabelle:

Offset	Bytes	Bedeutung
00H	1	Zahl der Bytes ohne dieses Byte
01H	2	reserviert (muß 0 sein)
03H	2	Seitenlänge in 1/20 Punkt (Standard ist 11 x 1440 = 15840)
05H	2	Seitenbreite in 1/20 Punkt (Standard ist 8.5 x 1440 = 12240)
07H	2	reserviert (muß 0 sein)
09H	2	oberer Rand in 1/20 Punkt (Standard ist 1440)
0BH	2	Texthöhe in 1/20 Punkt (Standard ist 9 x 1440 = 12960)
0DH	2	linker Rand in 1/20 Punkt (Standard ist 1.25 x 1440 = 1800)
0FH	2	Breite eines Textbereichs in 1/20 Punkt (Standard ist 6 x 1440 = 8640)

Tabelle 19.10 Formatierung eines Textbereichs

Unterer und rechter Rand lassen sich aus den obigen Angaben berechnen.

Sofern die obigen Einstellungen übernommen werden, ist keine Section Table (SETB) oder kein Section Property (SEP) erforderlich. Wird eine Section Tabelle benutzt, dann ist das erste Byte der SEP mit einem Wert zwischen 1 und 16 belegt. Die zugehörige Section-Table besitzt folgenden Aufbau:

Offset	Bytes	Bedeutung
00H	2	Zahl der Sections (immer 2 bei WRITE)
02H	2	undefiniert
04H		Feld mit folgenden Variablen pro Eintrag:
	4	Offset 1 Zeichnen hinter Section (cp)
	2	undefiniert
	4	Offset zugehörige Section Property (fcSep)

Tabelle 19.11 Aufbau der Section-Table

Ein WRITE-Dokument besitzt genau 2 Einträge (Section Descriptors, SED) in der Tabelle (siehe Offset 04H in Tabelle 19.11). Der Rest der Tabelle wird mit Füllbytes bis zur Grenze von 128 ergänzt. Der erste SED-Eintrag in der Variablen »cp« zeigt an, daß er sich auf alle Zeichen im Dokument bezieht. Das Feld »fcSep« enthält dann einen Zeiger auf den SEP-Block in der Datei. Der zweite Eintrag in der Tabelle ist ein Dummy und enthält im Feld »fcSep« den Wert FFFFFFFFH.

Im Block direkt nach der SEP-Section kann optional eine PGTB-Section folgen. Diese Page-Table (PGTB) besitzt folgenden Aufbau:

Offset	Bytes	Bedeutung
00H	1	Zahl der PGD (cpgd) 1 bis n
01H	1	undefiniert
02H	x	Feld mit PGD
	2	Seitennummer im gedruckten Dokument (pgn)
	4	Offset erste Zeichen der Seite (cpMin)

Tabelle 19.12 Aufbau einer Seitenbeschreibungstabelle (PGT)

Das Feld mit den Seitenbeschreibungen enthält *n* Einträge. Der restliche Bereich bis zum Ende des Blocks wird mit Füllbytes bis zur Blockgrenze ergänzt.

Die Font Tabelle (FFNTB)

In einem Block kann die »Font Face Name TaBle« (FFNTB) gespeichert werden. Diese besitzt folgende Struktur:

Offset	Bytes	Bedeutung
00H	2	Zahl der Einträge im folgenden Feld
02H	x	Feld mit Font Face Name (FFN)
	2	Zahl der folgenden Bytes in der FFN (cbFfn)
	2	ID der Fontfamilie (ffid)
	n	Fontname als ASCIIZ-Text (szFfn)

Tabelle 19.13 Aufbau einer Fonttabelle (FFNTB)

Falls im Feld »cbFfn« der Wert FFFFH steht, wird der nächste Eintrag der Tabelle im folgenden Block gespeichert. Ein Wert von 0 in diesem Feld signalisiert, daß keine weiteren Einträge vorliegen.

Gültige Werte für die Font ID-Nummer sind:

FF_DONTCARE
FF_ROMAN
FF_SWISS
FF_MODERN
FF_SCRIPT
FF_DECORATIVE

Weitere ID-Nummern können mit neueren Windows-Versionen definiert werden.

20 WordStar-Format

WordStar, ein Produkt der Firma MicroPro, war eines der ersten Textverarbeitungsprogramme für den IBM-PC. Das Programm wurde ursprünglich für das Betriebssystem CP/M 80 entwickelt und dann auf MS-DOS portiert, deshalb enthält es viele Eigenheiten aus der früheren CP/M-Welt. Auch wenn WordStar in Leistung durch WORD oder WordPerfect mittlerweile übertroffen wird, ist er doch noch recht weit verbreitet und kommt deswegen in diesem Buch zur Besprechung. Die folgenden Informationen beziehen sich auf die WordStar-Versionen 2.2 bis 7.x.

WordStar-Dateien verfügen nicht über Kopf- oder Endblöcke mit Formatinformationen. Das Programm legt den Text vielmehr im ASCII-Format in einer eigenen Datei ab. Zwischen den einzelnen Zeichen finden sich dann die Steuerinformationen für die Formatierung.

```
44 69 65 73 20 69 73 74-20 65 69 6E 20 54 65 78
 D  i  e  s     i  s  t    e  i  n    T  e  x
74 20 69 6E 20 57 6F 72-64 53 74 61 72 2C 20 64
 t     i  n     W  o  r  d  S  t  a  r  ,     d
65 72 0D 0A 73 69 63 68-20 61 75 63 68 20 1B 84
 e  r  .  .  s  i  c  h    a  u  c  h     .  ä
1C 6E 64 65 72 6E 20 6C-1B 84 1C 1B E1 1C 74 2E
 .  n  d  e  r  n     l  .  ä  .  .  ß  .  t  .
0D 0A 0D 0A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A
.....
.....
```

Abbildung 20.1 Dump eines WordStar-Textes

Die Datei selbst wird in Blöcke zu 128 Byte aufgeteilt, der letzte Block ist vom Textende ab mit dem Zeichen 1AH, dem normalen DOS-Zeichen für Dateienden, gefüllt.

Die WordStar-Steuercodes

Da WordStar noch aus der CP/M-Zeit stammt, benutzt das Programm lediglich 7 Bit zur Textdarstellung. Die Zeichen unterhalb 20H und oberhalb von 80H werden als Steuercodes für bestimmte Funktionen (Fettdruck, unterstrichen etc.) verwendet.

Das Programm unterscheidet verschiedene Gruppen von Steuercodes. Zunächst existieren paarweise auftretende Zeichen, wobei das Zeichen beim ersten Auftreten eine Funktion ein- und beim zweiten Auftreten wieder abschaltet (Tabelle 20.1):

Zeichen	Code	Steuercode
^PB	02H	Fettdruck ein/aus
^PD	04H	Doppelanschlag ein/aus
^PI	09H	Tabulator (Hard Tab)
^PJ	0AH	LF-Steuerzeichen
^PK	0BH	Seitenoffset Kopf-, Fußzeile
^PL	0CH	Form Feed
^PS	13H	Unterstreichen ein/aus
^PT	14H	hochgestellte Zeichen ein/aus
^PV	16H	tiefgestellte Zeichen ein/aus
^PX	18H	Überdrucken mit – ein/aus
^PZ	1AH	EOF-Marke (Textende)
^P[	1BH	ESC Beginn einer direkten Angabe (Literal)
^P\	1CH	Ende einer direkten Angabe
^P]	1FH	Beginn einer symmetrischen Sequenz
^P^	1EH	weiches Trennzeichen
^P_	1FH	weiches Trennzeichen Zeilenende

Tabelle 20.1 WordStar Kombinations-Steuercodes

Sobald das erste auftretende Steuerzeichen aus Tabelle 20.1 erkannt ist, schaltet WordStar die betreffende Funktion ein. Beim Auftauchen des zweiten Zeichens wird sie entsprechend wieder ausgeschaltet. Zum Ein- und Auschalten wird in der Regel dasselbe Zeichen verwendet.

Probleme gibt es hier bei der Benutzung multinationaler Zeichensätze auf dem PC, die den Bereich oberhalb von 80H nutzen. Dies ist zum Beispiel bei den deutschen Umlauten der Fall. Hier setzt WordStar die Umschaltfunktion (Tabelle 20.1) für direkte Angaben. Die Sonderzeichen werden einfach durch die Steuercodes 1BH und 1CH eingeklammert, und vor dem Zeichen steht der Code 1BH. Das nächste Zeichen ist direkt zu drukken. Code 1CH beendet den *Literalmodus*, und die nachfolgenden Zeichen werden wieder normal interpretiert. Der deutsche Umlaut »ä« beispielsweise erscheint im Text als 1BH 84H 1CH. Ohne diese Klammerung subtrahiert WordStar die Zahl 80H von dem betreffenden Zeichen und benutzt den Code als Steuerkommando. Tabelle 20.2 gibt die durch die Umlaute entstehenden Steuercodes an:

Zeichen	Code	Steuercode
ü	81H	01H entspricht ^A
ä	84H	04H entspricht ^D

Tabelle 20.2 Umsetzung der Umlaute als Steuercodes

Zeichen	Code	Steuercode
Ä	8EH	0EH entspricht ^N
ö	94H	14H entspricht ^T
Ö	99H	19H entspricht ^Y
Ü	9AH	1AH entspricht ^Z
ß	E1H	61H entspricht a

Tabelle 20.2 Umsetzung der Umlaute als Steuercodes

Diese ungewollte Umsetzung tritt zum Beispiel beim Import von ASCII-Dateien nach WordStar auf.

Neben den bereits erwähnten *paarweise* zu benutzenden Steuercodes kennt WordStar einige Steuerzeichen, die in der Regel alleine auftreten und sich sofort auf die Druckausgabe auswirken (Tabelle 20.3).

Zeichen	Code	Steuercode
^@	00H	Fix Printposition
^PA	01H	alternierende Schriftbreite
^PC	03H	Druckpause
^PE	05H	Benutzer-Druckbefehl
^PF	06H	Phantom Space
^PG	07H	Phantom Rubout
^PH	08H	Überdrucke vorhergehendes Zeichen
^PM	0DH	Carriage Return als Zeilenende
^PN	0EH	Standard Druckbreite
^PF	0FH	Drucke Phantomleerzeichen
^PQ	11H	Drucker Sonderfunktion (Benutzer)
^PW	12H	Drucker Sonderfunktion (Benutzer)
^PW	17H	Drucker Sonderfunktion (Benutzer)
^PY	19H	Druckfarbe wechseln (Version 4.x) kursiv ein/aus (ab Version 5)

Tabelle 20.3 WordStar Einzel-Steuercodes

Für die direkten Steuercodes oberhalb von 80H gibt es einige Regeln: Das achte Bit wird gesetzt, falls das letzte Zeichen einer Zeile eine Zeilentrennung enthält. Mit A0H wird ein Phantom-Leerzeichen erzeugt, die Kombination 0DH, 0AH wird in 8DH, 0AH (Zeilenumbruch) umgesetzt. Das Ende eines Abschnitts ist durch ein sogenanntes »hartes Return« (0DH, 0AH) markiert, das Ende einer Seite durch 0DH, 8AH.

Worttrennstellen werden im Text mit *Softhyphen*, dem sogenannten »weichen Bindestrich« (Code 1EH), markiert. An diesen Stellen kann das Programm eine Worttrennung durchführen. Falls eine solche Stelle am Zeilenende auftritt, wird als Trennzeichen der Code 1FH eingesetzt.

Die Punktbefehle

Weiterhin kommen innerhalb von Texten Steueranweisungen im Klartext, die sogenannten *Punktbefehle*, vor. Wie der Name schon sagt, werden diese Anweisungen durch einen Punkt, gefolgt von zwei Buchstaben, eingeleitet.

Bis zur Version 3.0 von WordStar mußten Parameter hinter den Punktbefehlen als ganze Zahlen angegeben werden. Ab WordStar 4.0 sind auch Ausdrücke erlaubt. In der Version 5.0 sind Abmessungen in Inch definierbar. Ab Version 5.5 (C) sind Abmessungen auch in Punkt oder Zentimetern erlaubt. Die Angaben sind ab Version 5.5 (C) in Hochkommas (z. B. 'CM') einzuschließen. Die Befehle der folgenden Tabelle werden überwiegend in der englischen Notation angegeben, da dann ein Rückschluß auf die jeweiligen Abkürzungen möglich ist.

Befehl	Bedeutung
.AV	Abfrage einer Variablen beim Ausdruck (ask variable)
.AW	Ausrichtung/Wortumbruch Ein/Aus (Align and word wrap)
.BN	Auswahl Papierschacht (1 bis 4) (bin select)
.BP	Bidirektional Druck Ein/Aus (bidirectional print)
.CC	Conditional column break
.CO	Spalten (Columns)
.CP	Conditional page break
.CS	Clear screen and display message
.CV	Convert note type
.CW	Character width (in 1/120 inch)
.DF	Data file that will be merged into text Format: CSV, DBF, WKS etc.
.DM	Display a message
.E#	Set endnote value
.EI	End if
.EL	Else
.F#	Set footnote value and numbering type
.FI	File insert (bis 7 Stufen)
.FM	Footer margin (Leerzeilen am Blattende)

Tabelle 20.4 Die Steuerbefehle in WordStar

Befehl	Bedeutung
.FO	Footer (Fußtext)
.F1	Footer (für Folgeseiten)
.F2	Second footer
.F3	Third footer
.GO	Go to top or bottom of document
.HE	Header (Kopfzeile)
.H1	Header (für Folgeseiten)
.H2	Second header
.H3	Third header
.HM	Header margin
.HY	Auto-Hyphenation on/off (WS 6.0 D)
.IF	If
.IG	Ignore (Kommentar bis Zeilenende)
.IX	Index
.KR	Kerning
.L#	Line numbering (Zeilennumerierung)
.LH	Line height (Zeilenhöhe in 1/48 Inch)
.LM	Left margin (linker Rand)
.LQ	Letter quality Ein/Aus
.LS	Line spacing (Zeilenabstand)
.MA	Math (speichert Ergebnis einer Berechnung)
.MB	Bottom margin (unterer Rand)
.MT	Top margin (oberer Rand)Befehl Bedeutung
.OC	Centering (Zentrierung Ein/Aus)
.OJ	Output justification Ein/Aus
.OJ	C center
.OJ	R right flush
.OP	Omit page number
.P#	Paragraph number
.PA	Page break (Seitenwechsel)
.PC	Page column (Position Seitennumer)
.PE	Print endnotes
.PF	Paragraph realignment while printing
.PG	Number pages

Tabelle 20.4 Die Steuerbefehle in WordStar

Befehl	Bedeutung
.PL	Page length (in Zeilen)
.PM	Paragraph margin
.PN	Page number (aktuelle Seitennummer)
.PO	Page offset
.PR	Printer information
.PS	Proportional spacing Ein/Aus
.RM	Right margin
.RP	Repeat
.RR	Ruler
.RV	Read variable (aus Datei)
.SB	Suppress blank lines on/off (WS 6.0 D)
.SR	Sub/superscript roll (in 1/48 Inch)
.SV	Set variable
.TB	Tab stops
.TC	Table of contents (Inhaltsverzeichnis) .TC1 bis .TC9 erlauben bis zu neun andere Tabellen.
.UJ	Micro Justierung
.UL	Underline (unterstreichen Ein/Aus)
.XE .XQ	Custom print control. Die folgende Hexsequenz .XR .XW definiert die Steuercodes.
.XL	Form feed
.XX	Strikeout character (Neudefinieren des Zeichens)

Tabelle 20.4 Die Steuerbefehle in WordStar

Symmetrische Codesequenzen

Diese Codesequenzen wurden ab WordStar 5.0 eingeführt, um Funktionen zu ermöglichen, die mit den Punktkommandos nicht darstellbar sind. Alle diese Sequenzen beginnen mit dem Zeichen 1DH und besitzen folgenden Aufbau:

Offset	Byte	Bedeutung
00H	1	1DH als Beginn der Sequenz
01H	2	Zähler
03H	1	Typ der Sequenz
04H	x	Daten der Sequenz...
..H	2	Zähler
..H	1	1DH Ende der Sequenz

Tabelle 20.5 Symmetrische Codesequenzen

Im Wort für den Zähler wird die Zahl der Zeichen der Sequenz (Zeichenzahl – 3) gespeichert. In der Sequenz dürfen alle Codes (einschließlich EOF 1AH) auftreten. Zur Zeit sind folgende symmetrische Sequenzen definiert:

Header

Diese Sequenz enthält nur einen Typ zur Definition verschiedener Optionen.

Header Sequenz (Typ 00H)

Diese Sequenz definiert die Version sowie den Treibernamen und enthält einen Zeiger auf die Style-Definitionen innerhalb der Datei. Die Sequenz besitzt folgenden Aufbau:

Bytes	Bedeutung
1	Header Sequenz (Typ = 00H)
1	Versionsnummer als BCD-Zahl 50H = WordStar 5.0 55H = WordStar 5.5 60H = WordStar 6.0
9	Name des Treibers als ASCIIZ-String
2	reserviert
4	Zeiger auf Style-Bibliothek der Datei
107	reserviert

Tabelle 20.6 Header Sequenz (Typ 00H)

Der gesamte Satz umfaßt damit 130 Byte (einschließlich des Rahmens von 1DH .. 1DH).

Print Controls

Diese Sequenzen enthalten die Definitionen für die Druckausgabe (Farben und Fonts).

Color Sequenz (Typ 01H)

Diese Sequenz definiert eine Farbe und besitzt folgenden Aufbau:

Bytes	Bedeutung
1	Print Control Color (Typ = 01H)
1	Farbnummer (0 bis 0FH)

Tabelle 20.7 Color Sequenz (Typ 01H)

Bytes	Bedeutung
1	vorhergehende Farbe in der Datei Kodierung: 00H schwarz 01H blau 02H grün 03H cyan 04H rot 05H magenta 06H braun 07H hellgrau 08H dunkelgrau 09H hellblau 0AH hellgrün 0BH helles cyan 0CH hellrot 0DH helles magenta 0EH gelb 0FH weiß auf schwarz

Tabelle 20.7 Color Sequenz (Typ 01H)

Die Sequenz umfaßt damit 9 Byte.

Font Sequenz (Typ 02H)

Diese Sequenz definiert den Font und besitzt folgenden Aufbau:

Bytes	Bedeutung
1	Print Control Font (Typ = 02H)
2	Fontbreite in HMI (1/1800 Inch)
2	Fonthöhe in VMI (1/1440 Inch)
2	Typestyle (Schriftart)
2	vorherige Fontbreite in HMI (1/1800 Inch)
2	vorherige Fonthöhe in VMI (1/1440 Inch)
2	vorheriger Typestyle (Schriftart)

Tabelle 20.8 Font Sequenz (Typ 02H)

Die Schriftart (Typestyle) wird dabei als Wort in einzelne Bits unterteilt (Tabelle 20.9).

Bit	Bedeutung
15	Proportional Flag
14	Letter Quality Flag
13	Symbol Mapping High Bit

Tabelle 20.9 Kodierung des Typestyles

Bit	Bedeutung
12	Symbol Mapping Low Bit 00 = Code Page 437 01 = Code Page 850 10 = Mathemat. Zeichen 11 = Symbol Font
11	Generic Style High Bit
10	Generic Style Low Bit 00 = Sans Serif Font 01 = Serif Font 10 = Script Font 11 = Display Font
9	1 = symmetrische Sequenz ist anders als bei vorherigen Versionen
8 ... 0	Nummer Schriftart (Typestyle)

Tabelle 20.9 Kodierung des Typestyles

Für die Numerierung der Schriftarten gilt zur Zeit folgende Belegung:

Nummer	Schriftart
0	LinePrinter
1	Pica
2	Elite
3	Courier
4	Helv (auch Helvetica, CG Triumvirate und Swiss)
5	Tms Rmn (auch CG Times, Times Roman und Dutch)
6	Gothic (oder 130 Letter Gothic)
7	Script
8	Prestige (oder 48 Prestige Elite)
9	Caslon
10	Orator
11	Presentations
12	Helv Condensed (auch Swiss Condensed)
13	Serifa
14	Blippo
15	Windsor
16	Century (oder 23)
17	ZapfHumanist

Tabelle 20.10 Schriftarten

Nummer	Schriftart
18	Garamond
19	Cooper
20	Coronet
21	Broadway
22	Bodoni
23	Cntry Schlbk (oder 16)
24	Universal Roman
25	Helv Outline
26	Peignot (auch Exotic)
27	Clarendon
28	Stick
29	HP-GL Drafting
30	HP-GL Spline
31	Times
32	HPLJ Soft Font
33	Borders
34	Uncle Sam Open
35	Raphael
36	Uncial
37	Manhattan
38	Dom Casual
39	Old English
40	Trium Condensed
41	Trium UltraComp
42	Trade ExtraCond
43	American Classic (auch Amerigo)
44	Globe Gothic Outline
45	UniversCondensed (auch Zurich Condensed)
46	Univers (auch Zurich)
47	TmsRmnCond (Oki Laserline 6)
48	PrstElite (siehe auch 8 Prestige)
49	Optima
50	Aachen (Postscript)
51	AmTypewriter

Tabelle 20.10 Schriftarten

Nummer	Schriftart
52	Avant Garde
53	Benguiat
54	Brush Script
55	Carta
56	Centennial
57	Cheltenham
58	FranklinGothic
59	FrstyleScrpt
60	FrizQuadrata
61	Futura
62	Galliard
63	Glypha
64	Goudy
65	Hobo
66	LubalinGraph
67	Lucida
68	LucidaMath
69	Machine
70	Melior (auch Zapf Elliptical)
71	NewBaskrvlle (auch Baskerville)
72	NewCntSchlbk
73	News Gothic (auch Trade Gothic)
74	Palatino (auch Zapf Calligraphic)
75	Park Avenue
76	Revue
77	Sonata
78	Stencil
79	Souvenir
80	TrmpMedievel (auch Activa)
81	ZapfChancery
82	ZapfDingbats
83	Stone
84	CntryOldStyle
85	Corona

Tabelle 20.10 Schriftarten

Nummer	Schriftart
86	GoudyOldStyle
87	Excelsior
88	FuturaCondensed
89	HelvCompressed
90	HelvExtraCompressed
91	Helv Narrow
92	HelvUltraCompressed
93	KorinnaKursiv
94	Lucida Sans
95	Memphis
96	Stone Informal
97	Stone Sans
98	Stone Serif
99	Postscript
100	NPS Utility
101	NPS Draft
102	NPS Corr
103	NPS SansSer Qual
104	NPS Serif Qual
105	PS Utility
106	PS Draft
107	PS Corr
108	PS SansSer Qual
109	PS Serif Qual
110	Download
111	NPS ECS Qual (daisy wheel)
112	PS Plastic (daisy wheel)
113	PS Metal (daisy wheel)
114	CloisterBlack
115	Gill Sans (auch Hammersmith)
116	Rockwell (auch Slate)
117	Tiffany (ITC)
118	Clearface
119	Amelia

Tabelle 20.10 Schriftarten

Nummer	Schriftart
120	HandelGothic
121	OratorSC (Star et al)
122	Outline (Toshiba)
123	Bookman Light (Canon)
124	Humanist (Canon)
125	Swiss Narrow (Canon)
126	ZapfCalligraphic (Canon)
127	Spreadsheet (Quadlaser)
128	Broughm (Brother printers)
129	Anelia (Brother printers)
130	LtrGothic (Brother Definition)
131	Boldface (Boldface PS)
132	High Density (NEC)
133	High Speed (NEC)
134	Super Focus (NEC P2200)
135	Swiss Outline (Cordata)
136	Swiss Display (Cordata)
137	Momento Outline (Cordata)
138	Courier Italic (TI 855)
139	Text Light (Cordata)
140	Momento Heavy (Cordata)
141	BarCode
142	EAN/UPC
143	Math-7 (HPLJ)
144	Math-8 (HPLJ)
145	Swiss
146	Dutch
147	Trend (Nissho)
148	Holsatia (Qume Laser)
149	Serif (IBM Pageprinter)
150	Bandit (Cordata)
151	Bookman (Cordata)
152	Casual (Cordata)
153	Dot (Cordata)

Tabelle 20.10 Schriftarten

Nummer	Schriftart
154	EDP (Epson GQ3500)
155	ExtGraphics (Epson GQ3500)
156	Garland (Canon Laser)
157	PC Line
158	HP Line
159	Hamilton (QMS)
160	Korinna (Cordata)
161	LineDrw (QMS)
162	Modern
163	Momento (Cordata)
164	MX (Cordata)
165	PC (Cordata)
166	PI
167	Profile (Quadlaser)
168	Q-Fmt (QMS)
169	Rule (Cordata)
170	SB (Cordata)
171	Taylor (Cordata)
172	Text (Cordata)
173	APL
174	Artisan
175	Triumvirate
176	Chart
177	Classic
178	Data
179	Document
180	Emperor
181	Essay
182	Forms
183	Facet
184	Micro (auch Microstyle, Eurostile)
185	OCR-A
186	OCR-B
187	Apollo (Blaser)

Tabelle 20.10 Schriftarten

Nummer	Schriftart
188	Math
189	Scientific
190	Sonoran (IBM Pageprinter)
191	Square 3
192	Symbol
193	Tempora
194	Title
195	Titan
196	Theme
197	TaxLineDraw
198	Vintage
199	XCP
200	Eletto (Olivetti)
201	Est Elite (Olivetti)
202	Idea (Olivetti)
203	Italico (Olivetti)
204	Kent (Olivetti)
205	Mikron (Olivetti)
206	Notizia (Olivetti)
207	Roma (Olivetti)
208	Presentor (Olivetti)
209	Victoria (Olivetti)
210	Draft Italic (Olivetti)
211	PS Capita (Olivetti)
212	Qual Italic (Olivetti)
213	Antique Olive (auch Provence)
214	Bauhaus (ITC)
215	Eras (ITC)
216	Mincho
217	SerifGothic (ITC)
218	Signet Roundhand
219	Souvenir Gothic
220	Stymie (ATF)
221	Bernhard Modern

Tabelle 20.10 Schriftarten

Nummer	Schriftart
222	Grand Ronde Script
223	Ondine (auch Mermaid)
224	PT Barnum
225	Kaufmann
226	Bolt (ITC)
227	AntOliveCompact (auch Provence Compact)
228	Garth Graphic
229	Ronda (ITC)
230	EngSchreibschrift
231	Flash
232	Gothic Outline (URW)
233	Akzidenz-Grotesk
234	TD Logos
235	Shannon
236	Oberon
237	Callisto
238	Charter
239	Plantin
240	Helvetica Black (PS)
241	Helvetica Light (PS)
242	Arnold Bocklin (PS)
243	Fette Fraktur (PS)
244	Greek (PS (Universal Greek))

Tabelle 20.10 Schriftarten

Alle Angaben in der obigen Liste erfolgen in der dezimalen Notation. Beachten Sie weiterhin, daß diese Liste laufend ergänzt wird.

Notes

Diese Sequenzen definieren Fuß- und Endnoten.

Fußnoten-Sequenz (Typ 03H)

Diese Sequenz definiert die Fußnoten im Text und besitzt folgenden Aufbau:

Bytes	Bedeutung
1	Footnote (Typ = 03H)
2	Zeilennummer Text Fußnote
2	Offset Footnote Number TAG Bit 15: 1= die restlichen Bits definieren den Offset zur internen Sequenz mit dem TAG der Fußnote. 0= Nummer der Fußnote
1	unbenutzt, bzw. intern belegt
x	Textbereich der Fußnote, kann weitere symmetrische Sequenzen enthalten

Tabelle 20.11 Fußnoten Sequenz (Typ 03H)

Falls die Fußnote keinen Tag enthält, wird das sechste Byte als Konvertierungsflag benutzt. Der Text kann dann eine weitere Fußnotensequenz enthalten, die zur Anzeige oder zum Ausdruck des mit der Fußnote verbundenen Tags dient. Die Sequenz benutzt dann den folgenden Aufbau:

Bytes	Bedeutung
1	Footnote (Typ = 03H)
2	unbenutzt (linecount wird als 1 angenommen)
2	Footnote Number
1	Conversion Flag (normalerweise 0) Falls .CV oder .FV# auftritt, gilt: Bits 0-3: 4=Note in Endnote konvertieren 6=Note in einen Kommentar konvertieren Bits 4-7: Format Typ 0=Symbole verwenden 1=Großbuchstaben verwenden 2=Kleinbuchstaben verwenden 3=Nummern verwenden
X	Textbereich der Fußnote

Tabelle 20.12 Fußnoten-Sequenz (Typ 03H) – Variante 2

Wird das Conversion-Flag benutzt, enthalten die unteren 4 Bits die Fußnotennummer. Ist der Wert auf 4 gesetzt, wird die Fußnote in eine Endnote gewandelt. Ist der Wert = 6, wird die Fußnote in einen Kommentar gewandelt. In den oberen Bits steht das Format für die Numerierung (0 = Symbole, 1 = Großbuchstaben, 2 = Kleinbuchstaben, 3 = Nummern).

Endnoten-Sequenz (Typ 04H)

Diese Sequenz definiert die Fußnoten im Text und besitzt folgenden Aufbau:

Bytes	Bedeutung
1	Endnote (Typ = 04H)
2	Zeilennummer Text Endnote
2	Offset Endnote Number TAG Bit 15 = 1: die restlichen Bits definieren den Offset zur internen Sequenz mit dem TAG der Endnote. 0: Nummer der Endnote
1	unbenutzt, bzw. intern belegt
x	Textbereich der Endnote, kann weitere symmetrische Sequenzen enthalten

Tabelle 20.13 Endnoten Sequenz (Typ 04H)

Der Text kann eine weitere Endnoten-Sequenz zur Anzeige oder Ausgabe des mit der Note verbundenen Text-Tags enthalten. Die Sequenz benutzt dann den folgenden Aufbau:

Bytes	Bedeutung
1	Endnote (Typ = 04H)
2	unbenutzt (linecount wird als 1 angenommen)
2	Endnote Number
1	Conversion Flag (normalerweise 0) Falls .CV oder .FV# auftritt, gilt: Bits 0-3: 4=Note in Fußnote konvertieren 6=Note in einen Kommentar konvertieren Bits 4-7: Format Typ 0=Symbole verwenden 1=Großbuchstaben verwenden 2=Kleinbuchstaben verwenden 3=Nummern verwenden
X	Textbereich der Endnote

Tabelle 20.14 Endnoten Sequenz (Typ 04H) – Variante 2

Wird das Conversion-Flag benutzt, enthalten die unteren 4 Bits die Fußnotennummer. Ist der Wert auf 3 gesetzt, wird die Endnote in eine Fußnote gewandelt. Ist der Wert = 6, wird die Endnote in einen Kommentar umgewandelt. In den oberen Bits steht das Format für die Numerierung (0 = Symbole, 1 = Großbuchstaben, 2 = Kleinbuchstaben, 3 = Nummern).

Anmerkung (Annotation) (Typ 05H)

Diese Sequenz definiert Anmerkungen im Text und besitzt folgenden Aufbau:

Bytes	Bedeutung
1	Anmerkungen (Typ = 05H)
2	Zeilennummer Anmerkung
2	Offset TAG der Anmerkungsnummer Bit 15 = 1: Die restlichen Bits definieren den Offset zur internen Sequenz mit dem TAG 0: Wort ist 0
1	unbenutzt
x	Textbereich der Anmerkung, kann weitere symmetrische Sequenzen enthalten

Tabelle 20.15 Anmerkung (Typ 05H)

Der Text kann eine weitere Anmerkungs-Sequenz mit dem gleichen Format enthalten. Die Struktur dieser Sequenz sieht folgendermaßen aus:

Bytes	Bedeutung
1	Anmerkungen (Typ = 05H)
2	unbenutzt (Line Count =1 wird angenommen)
2	unbenutzt
x	Textbereich für die Anmerkung

Tabelle 20.16 Symmetrische Sequenz einer Anmerkung

Kommentar (Typ 06H)

Diese Sequenz definiert Kommentare im Text und besitzt folgenden Aufbau:

Bytes	Bedeutung
1	Kommentare (Typ = 06H)
2	Zeilennummer Kommentar
2	Offset TAG (unbenutzt)
1	Conversion Flag (meist 0 außer bei .CV oder .F#)

Tabelle 20.17 Kommentar (Typ 06H)

Wird das Conversion-Flag benutzt, gelten die gleichen Bedingungen wie bei Fußnoten. Ist der Wert der unteren 4 Bits auf 3 gesetzt, wird der Kommentar in eine Fußnote gewandelt. Ist der Wert = 4, wird der Kommentar in eine Endnote gewandelt. In den oberen Bits steht das Format für die Numerierung (0 = Symbole, 1 = Großbuchstaben, 2 = Kleinbuchstaben, 3 = Nummern). Kommentare enthalten keine weiteren internen Sequenzen.

Die Codes 07H und 08H sind zur Zeit noch reserviert.

Tabs

Diese Sequenz dient zur Festlegung von Tabulatoren und Seiten.

Tabs (Typ 09H)

Diese Sequenz beschreibt Tabulatoren im Text und besitzt folgenden Aufbau:

Bytes	Bedeutung
1	Tabulator (Typ = 08H)
2	Tabulatorgröße in HMI
2	absolute Tabulatorgröße in HMI
1	Tabulator Typ
1	Tabulator Größe in 1/10

Tabelle 20.18 Tabs (Typ 09H) Tabs (Typ 09H)

Die Angaben für die Abmessungen eines Tabulators sind mir zur Zeit nicht ganz klar. Für den Typ des Tabulators werden verschiedene Zeichen eingesetzt:

space	hard tab
soft space	soft tab
!	center line tab
#	decimal tab
[	right align line tab
. oder *	dot leader

Tabelle 20.19 Tabulatortypen

Damit wird die Ausrichtung der Tabulatoren definiert.

Der Code 0AH ist zur Zeit noch reserviert.

End of Page (Typ 0BH)

Diese Sequenz sollte ignoriert werden, da sie durch den WordStar-Editor zur Anzeige des Seitenumbruchs verwendet wird.

Page Offset (Typ 0CH)

Diese Sequenz ist für die Druckertreiber reserviert und sollte in einem Textdokument nicht auftreten.

Andere

Die folgenden Sequenzen definieren weitere Optionen innerhalb einer WordStar-Datei.

Paragraphen-Nummer (Typ 0DH)

Diese Sequenz definiert die Numerierung der Absätze und besitzt folgenden Aufbau:

Bytes	Bedeutung
1	Paragraphen-Nummer (Typ = 0DH)
1	Level erhöhen: 0 = Level lassen 1 = Level erhöhen (2 -> 2.1)
1	Level erniedrigen: 0 = Level lassen 1 = Level erniedrigen
1	Levelnummer aktueller Absatz (1 bis *n*)
8*2	Levelnummern acht Worte (0 bis 9) Jedes Wort führt die Nummer der Stufe.
31	ASCIIZ-String mit Absatzformat

Tabelle 20.20 Paragraphen-Nummer (Typ 0DH)

WordStar kann eine Kapitelnumerierung bis zu acht Stufen durchführen. Die Verwaltung der Numerierung erfolgt in obiger Sequenz.

Indexeintrag (Typ 0EH)

Dieser Record enthält einen Text mit dem jeweiligen Index.

Druckerkontrolle (Typ 0FH)

Diese Sequenz speichert Steuercodes für die Druckeransteuerung.

Bytes	Bedeutung
1	Druckerkontrolle (Typ = 0FH)
2	Breite in HMI für Sequenz
1	Zahl der Zeichen für Bildschirmanzeige
x	Text für Bildschirmanzeige plus der Sequenz für den Drucker

Tabelle 20.21 Druckerkontrolle (Typ 0FH)

Mit diesem Satz kann eine besondere Sequenz an den Drucker geschickt werden. Die Sequenz kann auf dem Bildschirm angezeigt werden. Die Druckerzeichen folgen auf den Text für die Bildschirmanzeige.

Grafik (Typ = 10H)

Diese Sequenz erlaubt es, einen Filenamen für eine einzubindende Grafik anzugeben.

Bytes	Bedeutung
1	Grafik (Typ = 10H)

Tabelle 20.22 Grafik (Typ 10H)

Bytes	Bedeutung
n	Filename der Grafikdatei

Tabelle 20.22 Grafik (Typ 10H)

Die Länge des Dateinamens läßt sich aus der Länge der Sequenz bestimmen.

Paragraph Style (Typ = 11H)

Diese Sequenz beschreibt die Schriftarten für die einzelnen Absätze (Paragraphen).

Bytes	Bedeutung
1	Paragraph Style (Typ 11H)
2	neue Nummer der Schriftart
2	alte Nummer der Schriftart
2	vorhergehende geänderte Nummer
2	vorvorhergehende Schriftart

Tabelle 20.23 Paragraph Style (Typ 11H)

Diese etwas merkwürdige Numerierung ist erforderlich, da verschiedene Attribute etc. übernommen werden. Die Nummern dienen als Index in der Schriftarten-Bibliothek (Documents style libray). Die aktuelle Nummer der Schriftart wird im ersten Wort geführt. Dahinter folgt das Wort mit der Nummer der vorhergehenden Schriftart. Da sich Attribute und Fonts ändern können, wird bei jedem neuen Schriftstil ein temporärer Satz angelegt. Beim Zurückschalten können dann die alten Parameter wieder verwendet werden. Die Speicherung des vorvorhergenden Schriftstils erlaubt es WordStar, beim Zurückblättern die Einstellungen schnell zu erkennen.

Die Codes 12H bis 14H sind zur Zeit reserviert.

Alternate Font Change (Typ 15H)

Diese Sequenz belegt mehrere Bytes, wobei im ersten Byte die Werte

0 = Normaler Font
1 = Alternate Font

abgelegt werden. Die restlichen Bytes enthalten eine weitere symmetrische Sequenz mit der Definition der neuen Fontdaten (Breite Höhe, Fontnummer) und der vorherigen Fontdaten (siehe Typ 02H).

Truncation (Typ = 16H)

Diese Sequenz wird nur verwendet, falls die Zeichen einer symmetrischen Sequenz nicht in den Arbeitsspeicher passen.

Japanese Font-Shift-In/Shift-Out (Typ = 17H)

Diese Sequenz besitzt ein Byte und wird nur in der japanischen WordStar-Version verwendet. Mit Shift-in = 1 wird zum betreffenden Font geschaltet, der Wert Shift-out = 0 schaltet zum normalen Font zurück. Ist der Font eingeschaltet (Shift-in), verwendet WordStar die Codes 1BH/1CH (Zeichenumbruch) nicht mehr. Statt dessen wird der asiatische Zeichensatz mit den Prefix-Codes 81H-9FH, E0H-FEH und den Codes 20-7FH für das Zeichen benutzt.

Die restlichen Codes 18H bis FFH sind reserviert.

Aufbau einer »Paragraph Style Library«

Die Bibliothek mit den Schriftarten läßt sich in

- der Datei WSSTYLE.OVR
- als Kopie in der editierten Datei
- als Kopie in einer temporären Datei

ablegen. Im Header einer symmetrischen Sequenz kann ein 32-Bit-Offset auf die betreffende Schriftart gespeichert sein (siehe oben). Der Wert muß dabei auf eine 128-Byte-Seite mit dem sogenannten Master Index verweisen. Dieser Index besitzt folgenden Aufbau:

Bytes	Bedeutung
1	1AH als EOF-Markierung
2	nächster verfügbarer 512-Byte-Block Offset relativ zum Index Start
1	Zahl der Objekte (meist 1)
	Beginn des Master-Index
2	Zahl der Indexeinträge
2	Größe eines Objektes (102 für Par. style)
4	Zeiger in Objekt-Index
	Beginn des Objekt-Index
1	Zahl der Indexeinträge (14 für Par. styles)
4	Link zum nächsten Index Block relativ zum Anfang des Master-Index
24	Objektname (links justiert mit Blanks aufgefüllt)
5	intern benutzt
4	Zeiger auf Style-Definition
	Style-Definition
3*2	Fontbeschreibung (-1 = abgeleitet)

Tabelle 20.24 Paragraph Style Library

Bytes	Bedeutung
4	reserviert
2	linker Rand in HMI (-2 = abgeleitet)
2	rechter Rand in HMI (-2 = abgeleitet)
2	Paragraph Grenze in HMI (-2 = abgeleitet)
2	reserviert
1	Zahl der Tabulatoren (0 = abgeleitet)
1	Zahl der dezimalen Tabulatoren (0 = abgeleitet)
32*2	Tabulatorpositionen in HMI (erster Eintrag -1 -> abgeleiteter Font)
2	reserviert
1	Justierungsflag 0=keine Justierung -1=abgeleitet von anderem Font 1=Justierung rechts 2=Justierung zentriert 3=Justierung rechts
1	Word Wrap Flag 0=Umbruch Aus -1=abgeleitet von anderem Font 1=Umbruch Ein
2	Zeilenhöhe in VMI (-1 = abgeleitet)
1	Zeilenschaltung (-1 = abgeleitet) (Zwischenraum als Wert 1 – 9)
2	Druckattribute explizit Ein
2	Druckattribute explizit Aus
1	Farbe (-1 = abgeleitet)
6	reserviert

Tabelle 20.24 Paragraph Style Library

Sobald die Bits der Druckerattribute übereinstimmend in beiden Worten auf Aus gesetzt sind, wird das betreffende Attribut von einem anderen Font übernommen (vererbt). Für die einzelnen Bits gilt folgende Kodierung:

Strikeout0000000000000001B
Doublestrike0000000000000010B
Underlining0000000000001000B
Subscript 0000000000010000B
Superscript 0000000000100000B
Bold 0000000001000000B
Italics 0000000010000000B

Die Kodierung der Farben wurde in einer der symmetrischen Sequenzen beschrieben. Damit möchte ich die Beschreibung des WordStar-Formats beenden.

25 Das AMI Pro Dateiformat (Version 3.0/4.0)

AMI Pro ist ein Textverarbeitungsprogramm von Lotus, welches die Dokumente in Dateien mit der Extension .SAM speichert. Damit diese Dateien mit einem Texteditor zu bearbeiten sind, verwendet AMI Pro 7-bit ASCII Zeichen (mit einigen Erweiterungen) zur Sicherung der Daten. Das Dateiformat von AMI Pro ist dabei zwischen den verschiedenen Versionen kompatibel. Lediglich die AMI Pro-Versionen vor 2.0 sichern Grafiken in separaten .Gxx Dateien. Die Zeichen xx stehen dabei für eine fortlaufende Numerierung (00, 01, 02, etc.). Ab AMI Pro Version 2.0 werden die Grafiken am Ende einer .SAM-Datei angefügt. Der betreffende Abschnitt in der Datei wird mit der Abschnittsmarke [embedded] eingeleitet.

AMI Pro verwendet zusätzlich sogenannte Style Sheets (Formatvorlagen) zur Formatierung der Dokumente. Der Aufbau dieser .STY-Dateien ist identisch zu den SAM-Dateien, lediglich die Verweise auf die Formatvorlagen fehlen in diesen Dateien. Weiterhin wird ein neues Kommando [newmac] für Makronamen benutzt. Das Makro wird beim Öffnen der Formatvorlage automatisch ausgeführt.

Der Inhalt einer SAM Datei

Jede AMI Pro-Dokumentdatei besteht aus drei Abschnitten:

- Ein strukturierter Abschnitt im Kopf enthält alle Informationen zur Formatierung des Dokuments.
- Daran schließt sich der Bereich mit dem (unstrukturierten) Text an. Dieser Bereich beinhaltetet auch die Kopf-/Fußzeilen, Anmerkungen etc.
- Der letzte Abschnitt innerhalb der AMI PRO-Datei ist für eingebettete Objekte (Bilder, Gleichungen etc.) reserviert.

Der strukturierte Bereich am Anfang des Dokuments untergliedert sich seinerseits wieder in mehrere Abschnitte mit der Beschreibung der Komponenten im Dokument (Abschnitte, Layout, Formate etc.).

- Jeder Abschnitt wird durch ein Schlüsselwort eingeleitet, welches in eckigen Klammern [] steht. Diese Nomenklatur ist mit dem Aufbau der Windows-INI-Dateien vergleichbar.
- Das Schlüsselwort mit den Klammern [...] muß dabei am Zeilenanfang beginnen.
- An die Zeile mit dem Schlüselwort schließt sich eine variable Anzahl an Zeilen mit Definitionsdaten an. Diese Folgezeilen werden durch ein oder mehrere Tabulatorzeichen nach rechts eingerückt. Hierdurch erhält die Datei eine hierarchiche Struktur.
- Alle numerischen Felder innerhalb einer Zeile werden als vorzeichenbehaftete ASCII-Zahlen vereinbart.

- Einheiten und Abstandsangaben beziehen sich auf die Maßeinheit twips (1440 twips entspricht 1 Zoll oder 20 Punkt).
- Flags und Bitfelder werden ebenfalls als ASCII-Strings gespeichert. Um den Wert eines Bitfeldes zu ermitteln, sind alle Teilwerte zu addieren. Damit erhalten Sie den Binärwert des Flags.
- Enthält eine Zeile einen Schmuckpunkt oder einen Formatnamen (Style) mit 8-Bit-Zeichen, wird eine Escape Sequenz benutzt. Damit werden die 7-Bit-Zeichen in der Datei von diesen Zeichen unterschieden.

Weitere Einzelheiten finden Sie auf den folgenden Seiten.

Der Dokument Abschnitt

Alle Kommandos in diesem Abschnitt werden mit einem Schlüsselwort in der ersten Zeile eingeleitet:

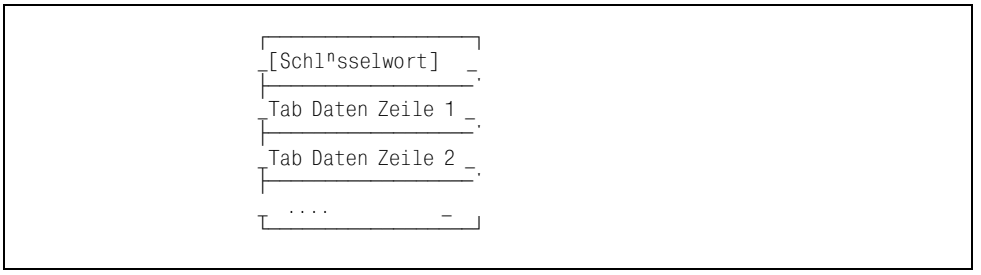


Abbildung 25.1 Struktur eines AMI Pro Records

Alle Folgezeilen werden mit einem oder mehreren Tabulatorzeichen eingerückt. Diese Zeilen können eine unterschiedliche Anzahl an Daten aufnehmen.

[encrypt]

Dieses Schlüsselwort wird nur für verschlüsselte Dateien verwendet. Dann enthält die zweite Zeile das *Password*. In diesem Fall ist der folgende Inhalt der .SAM-Datei verschlüsselt und kann ohne Passwort nicht gelesen werden.

[ver]

Mit diesem Schlüsselwort wird die Versionsnummer des erzeugenden Programmes gespeichert. Der Satz muß vor dem Abschnitt mit dem [sty]-Schlüsselwort kommen. Die Zeile nach dem [ver]-Schlüsselwort enthält dann die Versionsnummer der AMI-Pro-Datei. Beachten Sie hierbei aber, daß es sich nicht um die AMI Pro-Versionsnummer handelt. Vielmehr wird hier die Version der verwendeten Formatstruktur benutzt. AMI 1.0 benutzt die Version 3.0, während AMI Pro 1.1 die Version 4.0 verwendet. Die Version 4.0 stellt eine Übermenge des Fileformats der Version 3.0 dar. Nachfolgend wird die Version

4.0 des AMI Pro-Fileformats beschrieben. Zusätzlich sind aber auch die Differenzen zu den älteren Versionen aufgeführt.

[sty]

Dieses Schlüsselwort muß sich an die [ver]-Beschreibung anschließen und enthält den Verweis auf die Formatvorlage (Style sheet). Handelt es sich um eine Datei mit der Erweiterung .STY, entfällt das Schlüsselwort. Wurde die .SAM-Datei mit der Option *Speichere Format mit dem Dokument* gesichert, bleibt die Zeile nach dem Schlüsselwort leer (da kein STY-File existiert). Fehlt das Feld, gibt AMI Pro eine Fehlermeldung aus.

[files]

Über dieses Schlüsselwort wird ein Bereich mit weiteren Dateinamen eingeleitet. Hier lassen sich die Namen von Importdateien (z. B. Importgrafiken TIFF oder PCX) zur Verwendung im Dokument angeben. Die Dateien müssen dabei mit dem kompletten Pfad angegeben werden, falls Sie auf einem Datenträger gespeichert sind. Bitmap-Grafiken aus der Zwischenablage (Clipboard) erhalten dabei die Erweiterung .Gxx. Weiterhin werden Dateinamen für Zeichnungen und Dateien für DDE-Verbindungen in diesem Abschnitt angegeben.

[revision]

Dieses Schlüsselwort steht vor einem Flag, welches auf 1 gesetzt wird, falls die Revision-Marke eingeschaltet ist. Enthält die Zeile nach dem Schlüsselwort den Wert 0, ist die Markierung für Revisionen nicht gesetzt.

[recfiles]

Tritt dieses Schlüsselwort auf, ist eine externe Datei in das Dokument einzufügen. Die Folgezeilen enthalten dann den Dateinamen und die Optionen zum Zusammenfügen der beiden Teildokumente. Ami Pro verwendet diese Technik zum Beispiel beim Etikettendruck. Ein solches Etikett wird dann als eine Art Miniaturseite behandelt. Der Bereich nach dem Schlüsselwort besitzt folgenden Aufbau:

Parameter
Dateiname (max.80 Byte)
Dateibeschreibung (max. 80 Byte)
(nur falls Datei nicht aus AMI Pro)
Flag (ASCII)
1: Merge Print
2: Merge View Print

Tabelle 25.1 [recfiles]-Struktur[recfiles]-Struktur

4: Merge Save As
8: With Conditions
16: As Label
64: New Wave Objekt
128: New Wave desc. Objekta
Einheiten (1 = Zoll, 2 = Zentimeter, 3 = Pica, 4 = Punkt)
Label Across
Label Down
First Down (<= 32768 twips)
First Right (<= 32768 twips)

Tabelle 25.1 [recfiles]-Struktur[recfiles]-Struktur

Die beiden Einträge *Label Across* und *Label Down* definieren die Zahl der Etiketten (quer, hoch) auf einem Etikettenbogen und werden als Zahlen im Bereich zwischen 0 und 128 angegeben. Beachten Sie aber, daß jeder Wert als ASCII-String kodiert wird.

[toc]

Dieser Abschnitt definiert das Inhaltsverzeichnis (*table of contents*). Das Schlüsselwort tritt nur dann in einer AMI Pro-Datei auf, wenn vom Benutzer ein solches Inhaltsverzeichnis generiert wurde.

Parameter
Format Ebene 1 (style for level one)
Format Ebene 2
Format Ebene 3
Separator für Ebene 1 (String)
Separator für Ebene 2
Separator für Ebene 3
Flags
1: Seitennummern in Ebene 1
2: Nummern rechts justiert (Ebene 1)
4: Seitennummern in Ebene 2
8: Nummern rechts justiert (Ebene 2)
16: Seitennummern in Ebene 3
32: Nummern rechts justiert (Ebene 3)

Tabelle 25.2 [toc]-Abschnitt[toc]-Abschnitt

Über diesen Abschnitt kann AMI Pro jederzeit das Inhaltsverzeichnis aktualisieren.

[master]

Dieser Abschnitt enthält die Informationen für das Hauptdokument (master document information).

Parameter
Inhaltsverzeichnis (max. 78 Zeichen)
Index Name (max. 78 Zeichen)
Index Flag (1 für separate Zeilen im Index)
Länge Dateiliste in Byte
Zahl der Dateien
Dateiliste

Tabelle 25.3 [master]-Abschnitt[master]-Abschnitt

Dieser Abschnitt muß in jedem Hauptdokument (Master Dokument) auftreten.

[sequence]

Dieser Abschnitt beschreibt einen Abschnitt (Section) im Dokument. Hierbei gilt folgender Satzaufbau:

Parameter
Seitennummer Start
Fußnotennummer Start
Notennummer Start
Zahl der Sequenzen
Zahl der Formate (Styles)
Für jeden Abschnitt (Section)
Abschnittsnummer (Section number)
Abschnittsname (Section name)
Für jedes Format (Style)
Formatnummer (Style number)
Formatname (Style name)

Tabelle 25.4 [master]-Abschnitt[master]-Abschnitt

Dieser Abschnitt tritt in jedem Teildokument auf, welches im Hauptdokument zusammengefaßt wird.

[desc]

In diesem Abschnitt können benutzerspezifische Kommentare, Revisionsdatum etc. abgelegt werden.

Parameter
Beschreibung (max. 120 Zeichen)
Benutzerfeld 1 (max 34 Zeichen)
Benutzerfeld 2 (max 34 Zeichen)
Benutzerfeld 3 (max 34 Zeichen)
Benutzerfeld 4 (max 34 Zeichen)
Datum letzte Revision (in Sekunden seit 1.1.1980, 12:00 A.M.)
Zahl der Änderungen (Number of edits)
Datum Erzeugung (creation date, in Sekunden seit 1.1.1980)
Bearbeitungszeit (edit time, akkumuliert in Minuten)
Seitenzahl in der Datei
Zahl der Wörter in der Datei
Zeichenzahl in der Datei
Dateigröße in KByte
Textgröße (unbenutzt)
Schlüsselworte (Keywords)
Locked Flag (falls gesetzt, Benutzername sonst Leerzeile)
Benutzerfeld 5
Benutzerfeld 6
Benutzerfeld 7
Benutzerfeld 8

Tabelle 25.5 [desc]-Abschnitt[desc]-Abschnitt

Dieser Abschnitt wird seit der Version 4.0 (AMI Pro 1.1) verwendet.

[book]

Dieser Abschnitt definiert Marken (bookmarks) in Tabellen. Die Marken werden durch Leerzeichen getrennt.

Parameter
Name der Marke (bookmark name)

Tabelle 25.6 [book]-Abschnitt[book]-Abschnitt

Zahl der Rahmen für Tabellen (frames holding table)
Zeile
Spalte

Tabelle 25.6 [book]-Abschnitt[book]-Abschnitt

[prn]

Enthält den Druckernamen, für den das Dokument formatiert wurde. Als Druckername wird der logische Name aus dem Windows Control Pannel benutzt. Der Treibername darf ebenfalls, durch ein Komma vom Druckernamen getrennt, angegeben werden.

In zukünftigen Versionen von AMI Pro wird der Treibername benutzt. Ist die Zeile hinter [prn] leer, oder fehlt der komplette Abschnitt in der Datei, dann verwendet AMI Pro den Standard-Drucker aus Windows.

[stpg]

Wird in älteren Versionen benutzt, um den Startwert für die Seitennumerierung zu setzen. Fehlt das [stpg]-Schlüsselwort, verwendet AMI Pro den Startwert 1. In Version 4.0 werden die Seitennummern in der Marke für die Seitennummern (*page number mark*) gesichert.

[lang]

Definiert die Sprache (Language) für das Dokument. Dies ist für die automatische Trennung und für die Rechtschreibprüfung erforderlich.

Code	Sprache
1	Amerikanisches Englisch
2	Britisches Englisch
3	Französisch
4	Kanadisches Französisch
5	Italienisch
6	Spanisch
7	Deutsch
8	Niederländisch
9	Schwedisch
10	Norwegisch
11	Dänisch
12	Portugiesisch

Tabelle 25.7 Kodierung der SprachenKodierung der Sprachen

13	Finnisch
14	Medical
15	Legal
16	Griechisch
17	Brasilianisches Portugiesisch
18	Australisches Englisch
19	Polnisch
20	Russisch

Tabelle 25.7 Kodierung der SprachenKodierung der Sprachen

Alle Werte werden dabei im ASCII-Code gespeichert. Fehlt dieser Record, verwendet AMI Pro amerikanisches Englisch als Sprache.

[fopts]

Dieser Satz definiert die Optionen für Fußnoten. Hierbei gilt folgende Struktur:

Parameter
Options Flag
1: sammeln am Seitenende
2: Seitennummer jede Seite zurücksetzen
4: Trennlinie
Startnummer (<= 9999)
Länge Trennlinie (<= 32767 twips)
Einrückung Fußnote (<= 32767 twips)

Tabelle 25.8 [fopts]-Abschnitt[fopts]-Abschnitt

[newmac]

Dieser Abschnitt wird nur in Formatvorlagen (style sheets) gespeichert und vereinbart das Makro, welches beim Öffnen der Vorlage ausgeführt wird.

Parameter
Makroname
Flag
0: Makro beim Öffnen nicht ausführen
1: Makro beim Öffnen ausführen

Tabelle 25.9 [newmac]-Abschnitt[newmac]-Abschnitt

[lnopts]

Dieser Abschnitt vereinbart die Optionen zur Zeilennumerierung in AMI Pro. Es gilt dabei folgende Struktur.

Parameter
Options Flag
1: Zeilen numerieren
2: Jede Zeile
4: Jede ungerade Zeile
8: Jede fünfte Zeile
16: Nach jeder Seite zurücksetzen
32: Jede n-te Zeile
TAG-Name zur Formatierung der Zeilennummern (<= 13 Byte)
Schrittweite bei Numerierung jeder n-ten Zeile

Tabelle 25.10 [lnopts]-Abschnitt[lnopts]-Abschnitt

[docopts]

Mit diesem Satztyp werden Dokumentoptionen in AMI Pro vereinbart. Diese Optionen betreffen Informationen, die nicht zum Absatzformat oder dem Seitenlayout gehören (z. B. Trennungen, Kerning etc.).

Parameter
Trennungszone (Zeichenzahl, max. 9)
Flags:
1: Kerning
2: Widow/Orphan
4: Background printing
8: Background flowing
16: Snap always
32: Snap when open
64: Snap never

Tabelle 25.11 [docopts]-Abschnitt[docopts]-Abschnitt

Diese Optionen werden bei der Formatierung des Dokuments verwendet.

[tag]

Der *tag*-Abschnitt definiert das Absatzformat (paragraph style).

Parameter
Tag Name (max. 13 Zeichen)
Nummer Funktionstaste (<=16, nicht 1 oder 10)

Tabelle 25.12 [tag]-Abschnitt[tag]-Abschnitt

Jede Formatvorlage und jedes Dokument ohne Formatvorlage muß einen Bereich mit der Beschreibung des Textformats (*body text*) aufweisen.

[ftn]

Hier werden Fontinformationen wie Schriftart, Schriftgrad etc. definiert.

Parameter
Name Schriftart (max. 32 Zeichen)
Schriftgröße (in twips)
Farbe (RGB)
Schwarz 0
Blau 16711680
Rot 255
Magenta 16711935
Grün 65280
Gelb 65535
Cyan 16776960
Weiß 16777215
Flags:
1: Fett (Bold)
2: Kursiv (Italic)
4: Unterstrichen
8: Wörter unterstrichen
16: Kapitälchen
32: Kleinbuchstaben (Small Caps, nicht in AMI Pro 1.0)
64: Doppelt unterstrichen (nicht in 1.0)
128: Durchgestrichen (nicht in 1.0)
256: Hochgestellt
512: Tiefgestellt
1024: Alles in Kleinbuchstaben

Tabelle 25.13 [fnt]-Abschnitt[fnt]-Abschnitt

2048: Beginne mit Kapitälchen (Initial Caps)
4096: Erste Zeile fett
16384: Variabler Zeichenabstand (Pitch)
32768: Serifen Fonbsat

Tabelle 25.13 [fnt]-Abschnitt[fnt]-Abschnitt

Die Optionen *Variabler Abstand (pitch)* und *Serifen Font* werden nur verwendet, wenn kein passender Font vorhanden ist. Die Farben für die Zeichen werden als RGB-Werte gemäß den Windows 3.0-Spezifikationen definiert.

[algn]

Mit diesem Abschnitt werden die Informationen zur Ausrichtung (alignment) spezifiziert:

Parameter
Flags:
1: Linksbündig
2: Rechtsbündig
4: Zentriert
8: Blocksatz (Justify)
16: Einrücken beide Seiten gleich
256: Trennung
512: Einrückung 1 Tab
Einheit Ausrichtung (1: Zoll, 2: cm, 3: Pica, 4: Punkt)
Einzug alle (<= 32767 twips)
Erstzeileneinzug (<= 32767 twips)
Einzug Restzeilen (<= 32767 twips)

Tabelle 25.14 [algn]-Abschnitt[algn]-Abschnitt

Der Parameter *Einzug alle* wird nur belegt, wenn der Absatz links und rechts eingerückt wird (Flag beide Seiten einrücken).

[spc]

Dieser Abschnitt definiert den Zwischenraum zwischen den Zeilen.

Parameter
Flags:

Tabelle 25.15 [spc]-Abschnitt[spc]-Abschnitt

1: Zeilenabstand 1 fach
2: 1 1/2 fach
4: Doppelter Zeilenabstand
8: Benutzerspezifischer Abstand
16: Abstand immer anwenden (bezieht sich auf Absatz)
32: Nicht beim Seitenumbruch (bezieht sich auf Absatz)
Zeilenabstand (<= 32767 twips)
Einheit Zeilenabstand (1,2,3,4)
Abstand vorhergehender Absatz (<= 32767 twips)
Abstand folgender Absatz (<= 32767 twips)
Einheit Absatzabstand (1: Zoll, 2: cm,
3: Pica, 4: Punkt)
Linienstärk6(90, 100, 115)

Tabelle 25.15 [spc]-Abschnitt[spc]-Abschnitt

Im Satz werden sowohl die Parameter für den Zeilenzwischenraum im Absatz, als auch die Absatzabstände vereinbart. Für die Liniendicke werden nur die obigen drei Werte akzeptiert.

[brk]

In diesem Abschnitt werden die Seiten- und Spaltenumbrüche definiert.

Parameter
Flags:
1: Seitenumbruch vor (Absatz)
2: Seitenumbruch nach (Absatz)
4: Seitenumbruch (im Absatz)
8: Zusammenhalten (mit vorherigem Absatz)
16: Zusammenhalten (mit nächstem Absatz)
32: Verwende Tabulatoren
64: Nächstes Format (style)
128: Spaltenwechsel vorher
256: Spaltenwechsel nachher

Tabelle 25.16 [brk]-Abschnitt[brk]-Abschnitt

Die Parameter kontrollieren den Seitenumbruch im Zusammenhang mit der Absatzformatierung.

[line]

Über diesen Abschnitt werden Linien in einem Absatz definiert.

Parameter
Flags:
1: Linie oberhalb des Absatzes
2: Linie unterhalb des Absatzes
4: Breite Textzeile
8: Breite Marginalie (Margin line)
16: Zeilenbreite benutzerdefiniert
Zeilenlänge (<= 32767 twips, falls benutzerdefiniert)
Einheit Länge (1: Zoll, 2: cm, 3: Pica, 4: Punkt)
Linienfarbe (gültiger Windows RGB Wert)
Schwarz 0
75 % grau 12566463
50 % grau 8355711
25 % grau 4144949
10 % grau 1644825
Blau 16711680
Rot 255
Magenta 16711935
Grün 65280
Gelb 65535
Cyan 16776960
Weiß 16777215
Reserviert
Linienform oberhalb Absatz
1: 1/2 Punkt
2: 1 Punkt
3: 2 Punkt
4: 4 Punkt
5: 5 Punkt
6: 8 Punkt
7: 12 Punkt

Tabelle 25.17 [line]-Abschnitt[line]-Abschnitt

8: Doppelt 1 pt – 1 pt
9: Doppelt 2 pt – 2 pt
10: 1 pt – 2 pt – 1pt
11: 2 pt – 1 pt
12: 1 pt – 2 pt
Linienform unterhalb Absatz (Werte wie oben)
Linienabstand oberhalb (<= 32767 twips)
Linienabstand unterhalb (<= 32767 twips)
Einheit Linienmaß (1: Zoll, 2: cm,
3: Pica, 4: Punkt)

Tabelle 25.17 [line]-Abschnitt[line]-Abschnitt

Die Parameter definieren die Linienform und den Abstand der Linie zum Absatztext.

[spec]

Spezialzeichen (Schmuckpunkte, Aufzählungszeichen etc.) im Dokument werden in diesem Abschnitt vereinbart.

Parameter
Unbenutzt (Typ Schmuckpunkt in Version 3.0)
Outline Level (0-6 für Numerierung)
Text Schmuckpunkt (Bullet, 31 Zeichen)
Unbenutzt
Einheit Abstand (Space units 1: Zoll, 2: cm, 3: Pica, 4: Punkt)
Tab After Flag (unbenutzt in 2.0)
0: Ja
1: Nein
Attribute Schmuckpunkt (Bullet)
Numerierungs Flag
2: After lesser level
4: After intervening level
8: For legal (cumulative)

Tabelle 25.18 [spec]-Abschnitt[spec]-Abschnitt

Als Schmuckpunkte dürfen auch ANSI-Zeichen mit den Dezimalcodes 8 und 10-30 benutzt werden. Das Attribut für den Schmuckpunkt wird im Flag der Formatbeschreibung (Tabelle 25.14) definiert.

[nfmt]

Dieser Abschnitt definiert das Format numerischer Werte in einer Zelle einer Tabelle.

Parameter
Flag Ausrichtung in der Zelle:
1: Oben (nicht in AMI Pro 1.1)
2: Mitte (nicht in AMI Pro 1.1)
4: Unten (nicht in AMI Pro 1.1)
8: Verwende Tausendertrennzeichen
16: Minuszeichen voranstellen
32: Minuszeichen anhängen
64: Klammer () für negative Werte
128: Negative Werte rot
256: Währungssymbol voranstellen
Zellformat Flag
1: Standardformat (General)
2: Feste Stellenzahl (Fixed)
3: Währung (Currency)
4: Prozent (Percent)
Zahl der Dezimalstellen (max. 15)
Symbol Dezimalpunkt (1 Zeichen)
Symbol Tausendertrennzeichen (1 Zeichen)
Währungssymbol (3 Zeichen)
Name nächstes Format (Style)
Tabulatorzahl
Für jeden Tabulator:
Typ
Offset in twips
Einrückung von rechts

Tabelle 25.19 [nfmt]-Abschnitt[nfmt]-Abschnitt

Die Kodierung des Tabulatortyps ist identisch mit dem [frmlay]-tab-Abschnitt.

[frm]

Dieser Abschnitt beschreibt einen Rahmen (frame) zur Aufnahme einer Grafik oder eines Objekts.

Parameter
Seitennummer (<= 32767)
Flags:
1: Bitmap (importiert oder Clipboard)
2: Draw (nicht in AMI Pro 1.0)
4: Tabelle (nicht in AMI Pro 1.0)
8: Intern benutzt
16: Revisionsmarke erzeugt
32: In einer Zelle einer Tabelle
64: Opact
128: Zeilenumbruch (wrap around)
256: Wiederholung (repeating)
512: Textrahmen
folgende Bits werden nur bei Textrahmen benutzt:
1024: Intern benutzt (initialisiert mit 0)
2048: Kopfzeile
4096: Fußzeile
8192: Ungerade Seite (nicht in AMI Pro 1.0)
16384: Intern benutzt (initialisiert mit 0)
32768: Importiert
65536: Linie um den Rahmen
121072: kein Umbruch am Rand
262144: Metafile (nicht in AMI Pro 1.0)
524288: Verankert (nicht in AMI Pro 1.0)
1048576: Seite Tabelle (Tabellendatei 0, nicht in AMI Pro 1.0)
2097152: DDE-Rahmen (nicht in AMI Pro 1.0)
4194394: Gerade Seiten wiederholen (nicht in AMI Pro 1.0)
8388608: Gruppe (nicht in AMI Pro 1.0)
16777216: 1. Rahmen in Gruppe (Gruppenfeld muß gesetzt sein, nicht in AMI Pro 1.0)
33554432: Letzter Rahmen in Gruppe (Gruppenfeld muß gesetzt sein, nicht in AMI Pro 1.0)

Tabelle 25.20 [frm]-Abschnitt[frm]-Abschnitt

67208864: Inhaltsverzeichnis (Tabelle und Seitentabelle müssen gesetzt sein, nicht in AMI Pro 1.0)
134217728: Intern benutzt (initialisiert auf 0)
268435456: Rahmen für New Wave
536870912: Rahmen für ISD (Image Structure Descriptor)
1073741824: Rahmen für EPS-Bild
2114783648: Revisionsmarke gelöscht
Offset linker Rand (<= 32767 twips)
Offset oberer Rand (<= 32767 twips)
Offset rechter Rand (<= 32767 twips)
Offset unterer Rand (<= 32767 twips)
Flag Linie um den Rahmen
1: Alle Seiten
2: Links
4: Rechts
8: Oben
16: Unten
Unbenutzt
Position Randlinie
1: Mitte
2: innen
3: außen
4: geschlossen innen
5: geschlossen außen
Typ Randlinie (siehe Tabelle 25.18)
Schattierung links (in twips)
Schattierung oben (in twips)
Schattierung rechts (in twips)
Schattierung unten (in twips)
Flag abgerundete Ecken (% der Breite und Höhe)
Farbe Schattierung (RGB Wert)
Farbe Randlinie (siehe Tabelle 25.18)
Hintergrundfarbe (siehe Tabelle 25.18)

Tabelle 25.20 [frm]-Abschnitt[frm]-Abschnitt

Falls der Rahmen verankert ist:
ordinale Nummer des Anker Escape-Feldes im Text
Einrückung (in twips) ab linker Rand
Rahmenbreite (in twips)
Falls der Rahmen ein DDE-Ziel enthält:
Anwendung
Topic
Item

Tabelle 25.20 [frm]-Abschnitt[frm]-Abschnitt

Die Werte für die Schattierung sind immer positiv. Die vorgegebene Schattierung des Rahmens ist gemäß der Spezifikation in allen Richtungen anzuzeigen.

[frmmac]

Dieser Abschnitt dient zur Aufnahme eines Makronamens. Dieses Makro wird ausgeführt, sobald der Rahmen angewählt wird.

[frmname]

Dieser Abschnitt enthält den Namen des Rahmens (Rahmennummer) als ASCII-String.

[frmlay]

Dieser Abschnitt definiert das Layout einer Seite (page frame) mit Breite, Seitenränder, Bundsteg etc. Für die Seite wird dabei ein Rahmen zur Aufnahme des Textes definiert. Dieser Rahmen wird dann innerhalb der Seite positioniert.

Parameter
Rahmenhöhe (<= 32767 twips)
Rahmenbreite (<= 32767 twips)
Nicht benutzt
Linker Rand (<= 32767 twips, Abstand zum Rahmen)
Unterer Rand (<= 32767 twips, Abstand zum Rahmen)
Einheit Rand (1: Zoll, 2: cm,
3: Pica, 4: Punkt)
Oberer Rand (<= 32767 twips)
Rechter Rand (<= 32767 twips)
Flags:

Tabelle 25.21 [frmlay]-Abschnitt[frmlay]-Abschnitt

1: Spalten ausgleichen
2: Linie für Bundsteg
4: ---
Typ für Bundsteg (siehe Tabelle 25.18)
Farbe Bundsteg (siehe Tabelle 25.18)
Nicht benutzt
Nicht benutzt
Nicht benutzt
Spaltenzahl (<= 32767)
Für jede Spalte:
Links (Offset linker Seitenrand,<= 32767 twips)
Rechts (Offset linker Seitenrand,<= 32767 twips)
Tabulatoren (<= 32767 Tabs in der Spalte)
Für jeden Tabulator der Spalte
Typ Flag:
1: Tabulator links
2: Tabulator zentriert
4: Tabulator numerisch
16384: - voranstellen (Leader Hyphen)
32768: . voranstellen (Leader Dot)
49152: unterstreichen (Leader underline)
Offset von aktueller Spalte (<= 32767 twips)
Rahmenhöhe (<= 32767 twips)
Rahmenbreite (<= 32767 twips)
Nicht benutzt
Linker Rand (<= 32767 twips, Abstand zum Rahmen)
Unterer Rand (<= 32767 twips, Abstand zum Rahmen)
Einheit Rand (1: Zoll, 2: cm,
3: Pica, 4: Punkt)
Oberer Rand (<= 32767 twips)
Rechter Rand (<= 32767 twips)
Flags:
1: Spalten ausgleichen

Tabelle 25.21 [frmlay]-Abschnitt[frmlay]-Abschnitt

2: Linie für Bundsteg
4: ---
Typ für Bundsteg (siehe Tag Line Above Style)
Farbe Bundsteg (siehe Tag Line Color)
Nicht benutzt
Nicht benutzt
Nicht benutzt
Spaltenzahl (<= 32767)
Für jede Spalte:
Links (Offset linker Seitenrand,<= 32767 twips)
Rechts (Offset linker Seitenrand,<= 32767 twips)
Tabulatoren (<= 32767 Tabs in der Spalte)
Für jeden Tabulator der Spalte
Typ Flag:
1: Tabulator links
2: Tabulator zentriert
4: Tabulator numerisch
16384: - voranstellen (Leader Hyphen)
32768: . voranstellen (Leader Dot)

Tabelle 25.21 [frmlay]-Abschnitt[frmlay]-Abschnitt

An die Definition des Rahmens schließt sich die Definition mit dem Rahmeninhalt an.

[txt]

Dieser Abschnitt folgt einem [frmlay]-Abschnitt, sofern es sich um einen Textrahmen handelt. Der Text im Rahmen wird gemäß den Formatregeln für normale Texte formatiert. Die *EndOfText*-Marke muß aber linksbündig ohne führende Tabulatorzeichen geschrieben werden.

[btmap]

Enthält der Rahmen ein Bitmap-Bild, folgt dieser Record. Der Abschnitt wird in der Version 2.0 nicht verwendet.

Parameter
Bitmap Typ (0, verwendet durch Windows)

Tabelle 25.22 [btmap]-Abschnitt[btmap]-Abschnitt

Breite (in Pixel, <= 32767)
Höhe (in Pixel, <= 32767)
Breite in Byte (gerundet auf gerade Pixelzahl)
Bitebenen (planes für Farben)
Bits/Pixel (muß 1 sein)
X-Offset zum Ausschneiden (-32716 bis 32767 twips)
Y-Offset zum Ausschneiden (-32716 bis 32767 twips)
X-Breite des Rahmens (scaling)
Y-Höhe des Rahmens (scaling)
X-Breite Original Bitmap
Y-Höhe Original Bitmap
Flags Größe:
1: Originalgröße
2: Einpassen in Rahmen
4: Prozent
8: benutzerdefiniert
16: Seitenverhältnis (Aspekt) erhalten
32: Intern benutzt (0)
64: Intern benutzt (0)
128: Intern benutzt (0)
256: Rotiert
512: Intern benutzt (0)
1024: EPS Datei
2048: Graustufen TIFF Datei
4096: EPS ohne Vorschau
Einheit (1: Zoll, 2: cm,
3: Pica, 4: Punkt)
Name (für Kompatibilität mit älteren Versionen)
Pfadname (leer, falls Bitmap aus Zwischenablage, Pfad-Name bei importierter Bitmap)
Flag Bildbearbeitung: (nicht AMI Pro 1.0)
1: Edge enhancements
2: Smooth edge

Tabelle 25.22 [btmap]-Abschnitt[btmap]-Abschnitt

4: Negieren
8: Helligkeit
16: Kontrast
-- restliche Bits intern benutzt
Helligkeitswert (0-255)
Kontrastwert (0-255)
Graustufe Bildbearbeitung (2,4,6,8 Bits/Sample)
Skalierung % (1-Seitengröße%)
Bei einer Zeichnung (nicht benutzt in Version 2.0)
Intern benutzt
X-Größe Original Rahmen – X-Größe Bild
Y-Größe Original Rahmen – Y-Größe Bild
Einheit (1: Zoll, 2: cm,
3: Pica, 4: Punkt)
X-Größe Bild (zur Skalierung)
Y-Größe Bild (zur Skalierung)
Skalierung % (1-Seitengröße%)
Falls Rahmen ein New Wave-Objekt enthält
Intern benutzt (0)
Objekt Referenznummer
View (definiert durch New Wave)
Objekt vergrößert (zoomed mode)
Opened (Objekt ist geöffnet)

Tabelle 25.22 [btmap]-Abschnitt[btmap]-Abschnitt

[ISD]

Dieser Abschnitt folgt der Tabellenstruktur, falls diese einen ISD (Image Structure Descriptor) enthält.

Parameter
Dateiname Queelle .Gxx oder .Xxx (xx ist eine Zahl)
Datenformat (Fileextension)
Name Schnappschuß (leer oder .Gxx oder .Xxx)
Skalierungs Flag:

Tabelle 25.23 [isd]-Abschnitt[isd]-Abschnitt

1: Originalgröße
2: in Rahmen einpassen
4: Prozent
8: benutzerdefiniert
Einheit Skalierung (1: Zoll, 2: cm,
3: Pica, 4: Punkt)
Skalierung Rechteck links (-32757 bis 32767 twips)
Skalierung Rechteck oben (-32757 bis 32767 twips)
Skalierung Rechteck rechts (-32757 bis 32767 twips)
Skalierung Rechteck unten (-32757 bis 32767 twips)
Skalierung in % (nur falls Skalierung in %)
Rotationswinkel (1/10 Grad im Uhrzeigersinn)
Contextgröße (library context size) in Worten
Context
Contextformat (context creator format)
Bildausschnitt (Crop X in twips von links, -32767 bis 32767)
Bildausschnitt (Crop Y in twips von oben, -32767 to 32767)
Filtercontextgröße
Filtercontext

Tabelle 25.23 [isd]-Abschnitt[isd]-Abschnitt

[tbl]

Dieser Abschnitt folgt nur auf eine Tabellenstruktur. Die folgenden Felder werden in einer Zeile, getrennt durch Leerzeichen, ausgegeben.

Parameter
Zeilenzahl (1 – 4000)
Spaltenzahl (2, init. 0)
Standard Zeilenhöhe (<= 32767 twips)
Standardhöhe Bundsteg (Gutter) (<= 32767 twips)
Standard Spaltenbreite (<= 32767 twips)
Standardbreite Bundsteg (Gutter) (<= 32767 twips)
Flags:
1: Zeilenhöhe automatisch (auto row grow)

Tabelle 25.24 [tbl]-Abschnitt[tbl]-Abschnitt

2: Intern benutzt (2, init 0)
4: falls Tabelle zentriert auf Seite
8: Auszeichnungen geschützt(honor protection)

Tabelle 25.24 [tbl]-Abschnitt[tbl]-Abschnitt

Mit diesen Zeilen wird die Tabelle beschrieben.

[h]

Dieser Abschnitt folgt auf eine Tabellenstruktur und beschreibt benutzerspezifische Zeilen der Tabelle.

Parameter
Zeilennummer (basiert auf 0)
Höhe (<= 32767 twips)
Gutter Höhe (gutter height, <= 32767 twips)
Flags:
1: enthält Zellverbindung (connected cell)
2: einheitliche Zeilenhöhe (unique row height)
4: einheitliche Gutterhöhe (unique gutter heigth)
16: Kopfzeile der Tabelle
32: Seitenumbruch nach Zeile
64: Zeile mit Revisionsmarke
128: gelöschte Zeile mit Revisionsmarke
Intern benutzt (initialisiert mit 0)
Intern benutzt (initialisiert mit 0)
Intern benutzt (initialisiert mit 0)
.. zusätzliche Zeilen mit den gleichen Feldern
[e] ende des [h] records

Tabelle 25.25 [h]-Abschnitt[h]-Abschnitt

Jedes der sieben Felder wird in einer Zeile, getrennt durch Leerzeichen, ausgegeben. Die Struktur enthält für jede benutzerspezifische Zeile einen Eintrag. Die Struktur wird durch das Schlüsselwort [e] beendet.

[w]

Diese Struktur folgt einer Tabellendefinition und beschreibt benutzerspezifische Spalten.

Parameter
Spaltennummer (0 basierend)
Breite (<= 32767 twips)
Gutter Breite (<= 32767 twips)
Flags:
1: Enthält Zellverbindung (connected cell)
2: einheitliche Zellbreite
4: einheitliche Gutterbreite (unique gutter width)
16: Spalte mit Spaltentitel
32: Seitenumbruch nach Zelle
64: Zelle mit Revisionsmarke
128: Gelöschte Zelle mit Revisionsmarke
Intern benutzt (initialisiert mit 0)
.. zusätzliche Zeilen mit gleichen Feldern
[e] ende des [w] record

Tabelle 25.26 [w]-Abschnitt[w]-Abschnitt

Die Felder werden in einer Zeile, getrennt durch Leerzeichen, ausgegeben. Für jede benutzerdefinierte Spalte enthält die Struktur eine Zeile mit der Definition. Die Struktur wird durch das Schlüsselwort [e] beendet.

[data]

Die Daten der einzelnen Zellen einer Tabelle werden in [data]-Abschnitten abgelegt.

Parameter
Zeilennummer (0 basierend)
Spaltennummer (0 basierend)
Flag 1
1: Intern benutzt
2: Intern benutzt
4: Intern benutzt
8: Zellinhalt links ausgerichtet
16: Zellinhalt rechts ausgerichtet
24: Zellinhalt zentriert
32: Zellinhalt justiert

Tabelle 25.27 [data]-Abschnitt[data]-Abschnitt

64: Zelle mit Formel
128: Teil einer Zellverbindung (connected cell)
256: Zelle mit Verbindung nach oben links
512: Zelle mit Hintergrundschattierung
1024: Intern benutzt
2048: Intern benutzt
4096: Intern benutzt
Formelreferenz Flag
8192: Zelle ungültig
16384: Zelle mit Leader Dots
32768: Zelle mit Leader Hyphen
49152: Zelle mit Leader underline
Row Info zusammengefaßt (joined row info)
Column Info zusammengefaßt (joined column info)
Schattierung Zelle (1 basierender Index zum Tag mit den Linienfarben, 1-8)
Zellrand (Wort basierend auf dem Tag Line Style)
untere vier Bits: linke Linie
nächste vier Bits: rechte Linie
nächste vier Bits: obere Linie
nächste vier Bits: untere Linie
Flag 2
1: Zelle enthält text
Protectflag 1= Zelle geschützt

Tabelle 25.27 [data]-Abschnitt[data]-Abschnitt

Das Feld *joined row info* enthält die Zahl der Zeilen bis zur verbundenen Zelle. Dieser Wert ist jedoch nur gültig, wenn es sich um die Zelle in der obersten linken Ecke handelt. Zellen innerhalb der Tabelle enthalten die Zeilenzahl von der obersten Reihe bis zur verbundenen Zelle. Für das Feld *joined column info* gilt ähnliches, wobei hier jedoch die Spaltenzahl angegeben wird.

[lay]

Dieser Abschnitt definiert das Blattformat und das Seitenlayout (Page Layout) des Dokuments. AMI verwendet nur ein Format für das komplette Dokument.

Parameter
Layoutname(<= 13 Zeichen, STANDARD für 1.0)
Flags:
1: Letter
2: Legal
3: A3
4: A4
5: A5
6: B5
7: benutzerdefiniert
128: Intern benutzt
256: Landscape
512: Non-Alternating
1024: Mirror-imaged
2048: 2nd header
4096: 2nd footer

Tabelle 25.28 [lay]-Abschnitt[lay]-Abschnitt

In AMI Pro lassen sich jedoch mehrere Layouts mit unterschiedlichen Kopf-/Fußzeilen für gerade und ungerade Seiten definieren.

[rght]

Dieser Abschnitt dient zur Aufnahme der Informationen für die rechte (ungerade) Seite).

Parameter
Länge (<= 32767 twips)
Breite (<= 32767 twips)
Einheit (1: Zoll, 2: cm, 3: Pica, 4: Punkt)
Linker Rand (<= 32767 twips)
Unterer Rand (<= 32767 twips)
Oberer Rand (<= 32767 twips)
Rechter Rand (<= 32767 twips)
Flags:
1: Spalten gleich (columns balanced)
2: Linie für Bundsteg (gutter line)

Tabelle 25.29 [rght]-Abschnitt

4: Linie Seitenrand (page border line)
Linientyp Bundsteg (gutter line type)
Farbe Bundsteg (gutter color)
Typ Seitenrand (page border)
Einheit Rand (1-4, siehe oben)
Randabstand (<= 32767 vom Text)
Spaltenzahl (<= 32767)
Für jede Spalte:
Linker Rand (Offset von linker Seite <= 32767 twips)
Rechter Rand (Offset von linker Seite <= 32767 twips)
Tabulatoren in Spalte (<= 32767 twips)
Für jeden Tabulator:
Flag Tabulatortyp:
1: links
2: zentriert
3: rechts
4: numerisch (dezimal)
16384: Leader Hyphen
32768: Leader Dot
49152: Leader underline
Offset (in Spalte <= 32767 twips)

Tabelle 25.29 [rght]-Abschnitt

Das Feld *gutter line type* verwendet die gleiche Kodierung wie der Eintrag *Linienform* in Tabelle 25.18. Die Farbe des Bundstegs (*gutter color*) entspricht der Kodierung der Linienfarbe in Tabelle 25.18. Der Typ des Seitenrandes ist gemäß der Linienform in Tabelle 25.18 kodiert.

[lft]

Dieser Abschnitt enthält die Informationen für das Layout der linken Seite. Es wird die gleiche Struktur wie bei der rechten Seite verwendet (siehe oben). In Ami Pro 1.0 wird dieser Record nicht verwendet.

[hrght]

Dieser Abschnitt beschreibt die rechte Kopfzeile (*right header*) und benutzt die gleiche Struktur wie der Abschnitt zur Definition eines Textrahmens (siehe Tabelle 25.21).

[frght]

Dieser Abschnitt beschreibt die Fußzeile der rechten Seite und verwendet die Struktur des Textrahmens (siehe Tabelle 25.21).

[hlft]

Dieser Abschnitt beschreibt die linke Kopfzeile (*left header*) und benutzt die gleiche Struktur wie der Abschnitt zur Definition eines Textrahmens (siehe Tabelle 25.21). In Version 1.0 existiert dieser Record nicht.

[flft]

Dieser Abschnitt beschreibt die Fußzeile der linken (geraden) Seite. In Version 1.0 existiert dieser Record nicht. Hier wird das Layout der rechten Kopf-/Fußzeile verwendet. Dabei gilt die gleiche Struktur wie bei einem Textrahmen (siehe Tabelle 25.21). Lediglich das Schlüsselwort wechselt von *[frm]* nach *[lyfrm]* und die Layoutdaten werden mit *[frmlay]* an Stelle von *[lay]* eingeleitet.

[repfrm]

Rahmen die sich wiederholen, werden mit dem Schlüsselwort [repfrm] eingeleitet. Das Flagbit wird auf Wiederholung gesetzt. Die Struktur des Abschnitts entspricht dem Aufbau der Kopf-/Fußzeilen (header, footer).

[l1]

Der Abschnitt definiert den Aufbau der Layoutnummer der ersten Seite.

Parameter
`Nummer des verwendeten Layouts (<= 13 bytes)`

Tabelle 25.30 [l1]-Abschnitt

Ist der Wert im Feld größer 0, wird das Layout an Anfang der ersten Seite eingefügt.

[pg]

Dieser Abschnitt enthält Zusatzinformationen (page hints) um schneller durch das Dokument zu blättern. Das Format hängt von der AMI Pro-Version ab und wird nicht offengelegt.

[edoc]

Dieses Schlüsselwort zeigt das Ende des Dokumentabschnitts *(end of document section)* an. An diesen Abschnitt schließt sich der Dokumenttext an. Enthält die Datei Text, muß dieser [edoc]-Record auftreten.

Der Textbereich

Der eigentliche Text wird in der AMI Pro-Datei an das Ende des Dokumentbereichs absatzweise angehängt. Der Text wird dabei im ASCII-Zeichencode gespeichert. Zur Formatierung und für Zusatzinformationen fügt AMI Pro zusätzliche Daten im Text ein. Um diese von den Textzeichen zu unterscheiden, nutzt das Programm sogenannte *Escape Sequenzen*.

- Eine Escape Sequenz wird mit '<' eingeleitet und mit '>' beendet. Damit werden alle Zeichen der Escape Sequenz in '<...>' eingebettet.
- Wird ein '<' als normales Zeichen im Text verwendet, muß es verdoppelt '<<' gespeichert werden. Endet eine Escape Sequenz ohne '>' Zeichen, sollten folgende '<<' und '>>' Zeichen als Escape Limiter interpretiert werden (Bsp.: Seitenumbruch Index <:p<*!>>).

Ein Textabsatz (Paragraph) wird von einer Leerzeile gefolgt (Ausnahme bildet nur der Spezialfall eines leeren Absatzes). Jeder Absatz beginnt mit den Formatnamen. Der Formatname (Style) wird dabei durch ein @-Zeichen eingeleitet.

Die öffnende Klammer (ist ein weiteres Spezialzeichen. Soll dieses Zeichen im Text auftreten, wird es zwischen die Escape Zeichen <(> gesetzt. Ein <-Zeichen im Text wird dagegen als <;> in der AMI Pro-Datei gespeichert.

Für eingebettete Escape Zeichen werden zwei Bereiche mit Codes unterhalb 20H und oberhalb 80H verwendet. Diese Zeichen sind aber im 7-Bit-ASCII-Zeichensatz nicht definiert oder als Steuercodes reserviert. Tritt ein solches Zeichen auf, wird dieses vor dem Speichern in die Datei konvertiert:

- Ist der Zeichencode kleiner 20H, wird das Zeichen * vor dem Wert gespeichert. Als Wert wird der Zeichencode+20H gespeichert, d.h. vom gelesenen Zeichen muß 20H wieder subtrahiert werden.
- Zeichen mit Codes zwischen 80H und BFH verwenden das /-Zeichen als Escape Sequenz, wobei der Wert sich daran anschließt. Als Wert wird der Zeichencode-40H gespeichert. Das Zeichen muß deshalb nach dem Lesen um 40H erhöht werden, um den ursprünglichen Code zu erhalten.
- Liegt der Zeichencode zwischen C0H und FFH, wird die Escape Sequenz mit dem Zeichen \ eingeleitet. Daran schließt sich der Wert an, der aus dem Zeichencode-80H berechnet wird.

Mit dieser Kodierung werden die gespeicherten Daten auf 7 Bit ASCII-Codes reduziert.

Weiterhin werden andere Escape-Zeichen eingebettet. Diese dienen als Specialcodes für die Zeichenattribute und Formatinformationen. Die erste dieser Escape Sequenzen beginnt mit + und wird vom Escape Code gefolgt. Das Ende der Sequenz wird mit dem Zei-

chen – und dem Escape Code markiert. Hierbei sind die folgenden Escape Codes definiert:

Code	Bemerkung
' '	Plain text (space)
!	Text fett
22H	Text kursiv
#	Text unterstrichen
$	Text wortweise unterstrichen
%	Text durchgestrichen
&	Text hochgestellt
27H	Text tiefgestellt
(	Text mit Kleinbuchstaben (smallcaps)
)	Doppelt unterstrichen
*	Text geschützt
+	Nur Großbuchstaben
,	Nur Kleinbuchstaben
-	Initial caps
@	Absatz linksbündig justiert
A	Absatz rechtsbündig justiert
B	Absatz zentriert
C	Absatz Blocksatz
P	Trail (nur Indexseite)
Q	Lead (nur Indexseite)
R	im gleichen Absatz (nur Indexseite)

Tabelle 25.31 Escape Codes in AMI Pro

Das Semicolon (;) dient als spezielles Escapezeichen. Es wird im Dokument in ein > umgesetzt.

Alle anderen Escape Sequenzen beginnen immer mit einem Doppelpunkt (:), gefolgt vom Escape Code mit dem Typ. Die folgende Tabelle gibt diese Codes an.

Code	Bedeutung
D	Systemdatum und -format
P	aktuelle Seitennummern und -format
S	Zeilenzwischenraum (line spacing)
p	Seitenwechsel (page break)

Tabelle 25.32 Escape Codes in AMI Pro

Code	Bedeutung
c	Spaltenwechsel (column break)
f	Wechsel Zeichensatz
R	Tabulator Lineal (Tab ruler)
M	zusammenfügen (Merge)
N	Anmerkung (Note)
F	Fußnote
H	Kopfzeile
h	Fußzeile
O	Überschreiben (Overstrike)
A	Anker (Anchor)
t	Tabelle
s	Rechschreibprüfung (Spelling)
v	Revisionsmarke Absatz
T	Eintrag Inhaltsverzeichnis (Table of contents entry)
X	Exponentfeld (Power field)
Z	Marke (Bookmark)
d	Add Document Beschreibungsvariable
e	DDE-Link
n	New Wave-Object
r	Datum letzte Revision einfügen
k	Creation Zeit einfügen
V	Revisionsmarke
x	Tabelle mit Fußnoten
?	Unbekannt

Tabelle 25.32 Escape Codes in AMI Pro

Eine Escape Sequenz leitet einen Satz mit Feldern und Formatcodes ein. Der folgende Abschnitt beschreibt die Syntax der Escape Sequenz. Großbuchstaben in eckigen Klammern [STYLE] stehen für Felder in einer Escape Sequenc und müssen ausgefüllt werden.

<:D[STYLE]>

Dieser Satz definiert das Datum, gefolgt vom Datumsformat (Style) Das Format wird als Buchstabe von a bis l angegeben. Tabelle 25.34 definiert die unterschiedlichen Formate.

Style	Format
a	8/1/94

Tabelle 25.33 Datumsformate

Style	Format
b	Juni 3, 1995
c	2 Oktober 1993
d	Freitag, Mai 1, 1994
e	August 2
f	Samstag August 2
g	8/2
h	8/2/1991
i	2. August
j	2. August 1994
k	1993 July 9
l	July, 1993

Tabelle 25.33 Datumsformate

Die Buchstaben a, g und f treten ausschließlich als Kleinbuchstaben auf. Alle anderen Buchstaben dürfen klein und groß geschrieben werden. Falls möglich, wird das Datum an die Windows-Einstellung angepaßt.

<:P[STYLE&PAGENUM]m[PHYSCALPAGE],[PREFIX]>

Dieser Satz definiert die Seitennumerierung und das zugehörige Format. Die drei Felder werden durch Kommas getrennt. Das erste Feld enthält zwei Informationen. Das erste Zeichen hinter dem Buchstaben P definiert das Format der Seitennummer:

Style	Format
1	arabische Ziffern
I	Römische Ziffern (Großbuchstaben)
i	Römische Ziffen (Kleinbuchstaben)
A	alphanumerische Großbuchstaben
a	alphanumerische Kleinbuchstaben

Tabelle 25.34 Format der Seitennummern

Der zweite Teil des ersten Feldes enthält die Startnummer für die erste Seite als ASCII-String. Das nächste Feld enthält die ASCII-Nummer der physikalischen Seite, an der die Numerierung beginnt. Das letzte Feld enthält einen ASCII-String (max. 31 Zeichen), der vor der Seitennummer ausgegeben wird.

<:S+[SPACING]> oder <:S->

Diese Sequenz definiert den Zeilenabstand (*line spacing*). Das Kommando *<:S+xxxx>* schaltet den Zeilenabstand ein, wobei *xxxx* die ASCII-Zahl für den Zeilenabstand (in twips) angibt.

Wert	Line Spacing
0xffff	einfach
0xfffe	1,5
0xfffd	doppelt

Tabelle 25.35 Werte für den Zeilenabstand

Mit dem Kommando *<:S->* wird der Zeilenabstand wieder ausgeschaltet.

<:p[FLAG]>

Dieses Kommando signalisiert einen Seitenumbruch (*page break*) und wird von einem Flag Feld gefolgt. Das Feld kann wieder eine Escape Sequenz sein, da einzelne Werte außerhalb des gültigen Wertebereiches für ASCII-Zeichen(20H bis 80H) liegen.

Wert	Bemerkung
0	Text (Plain text)
1	Indexseite
2	Inhaltsverzeichnis
3	Endnote
64	Seite vertikal zentriert
128	Formatwechsel nächste Seite

Tabelle 25.36 Flag Seitenumbruch (page break)

<:f[PTSIZE][WINPITCH+FACE],[RED,GREEN,BLUE]>

Diese Escape Sequenz leitet einen Fontwechsel (*font exchange*) ein und besitzt zwei Varianten. Beim <:f>-String wird der Font auf das Absatzformat zurückgesetzt. In anderen Fällen enthält der Satz drei durch Kommas getrennte Felder. Das erste Feld definiert die Fontgröße in Punkt (ASCII-Wert). Das zweite Feld definiert den *font pitch* für Windows im ersten Zeichen. Hierbei wird der Offset 20H zum Wert addiert, um ein gültiges ASCII-Zeichen zu erhalten. Die nächsten Zeichen definieren den Namen des Zeichensatzes (type face). Das letzte Feld enthält drei ASCII-Werte, die durch Kommas getrennt werden. Diese definieren die Schriftfarbe (rot, grün, blau) im Bereich zwischen 0 und 255. Fehlt eines der drei Felder, ist die Standardeinstellung für den Absatz für den betreffenden Parameter zu verwenden.

<:R> oder <:R[COLS],[NUMTS1],[TABTYPE,OFFST],..,[NUMTS2]..>

<:R> setzt die Tabulatoren auf die Einstellungen im Seitenlayout zurück (*tab ruler escape*). Die zweite Variante definiert neue Tabulatorpositionen und enthält verschiedene ASCII-Felder. Die Felder werden durch Kommas separiert. Das erste Feld enthält die Zahl der Spalten. Für jede Spalte existiert ein Feld mit der Zahl der Tabulatorstopps. Für jeden Tabulator in der Spalte existiert einen Struktur mit zwei Feldern (TABTYPE, OFFSET), die den Tabulator beschreiben.

Bemerkung
TABTYPE
1: Tab links
2: Tab zentriert
3: Tab rechts
4: Tab numerisch (dezimal)
16384: Leader Hyphen
32768: Leader Dot
49152: Leader Underline
OFFSET (<= 32767 twips)

Tabelle 25.37 Tabulator Definitonsfeld

Die ersten vier Werte im Feld TABTYPE schließen sich gegenseitig aus. Der Wert 4 und die Einstellungen für die Tabulatorzeichen (Leader Hyphen etc.) können addiert werden. Das Ergebnis wird dabei als Integerwert in der Datei gespeichert. Dabei können auch negative Werte auftreten. Das Feld OFFSET definiert den relativen Offset vom Beginn der Spalte zur Tabulatorposition (in twips).

<:M[MERGEVARNAME]>

Dieser Befehl leitet eine *merge*-Escape Sequenz ein und wird gefolgt vom Namen der merge-Variable.

<:N[EDITDATE]\n-Text-\n\n>

Diese Escape Sequenz definiert eine Anmerkung (*note*), gefolgt von einer ASCII-Zahl für das Datum der Änderung (Editdate) der Anmerkung. Die nächste Sequenz definiert den Zeilenvorschub, gefolgt vom Anmerkungstext. Die Anmerkung kann dabei mehrere Absätze umfassen.

<:F\n-Text-\n\n>

Der Satz leitete eine Fußnote ein. Der Fußnotentext ist innerhalb der Escape Sequenz enthalten und darf mehrere Absätze umfassen.

<:H[FLAG]-Text-\n\n>

Mit dieser Escape Sequenz wird ein Flag für Kopf-/Fußzeilen (*header* H, *footer* h) gesetzt. Daß Flag definiert, um welches Sequenz es sich handelt:

Flag	Bemerkung
1	Fußzeile
2	Kopfzeile
4	Ungerade Seite header/footer
8	Gerade Seite header/footer
16	Odd/even für header/footer

Tabelle 25.38 Das Header/Footer-Flag

Die ersten beiden Einträge aus obiger Tabelle schließen sich gegenseitig aus. An das Flag schließt sich der Text für die Kopf- oder Fußzeile an. Dabei können im Text weitere Escape Sequenzen auftreten. Geschachtelte Escape Sequenzen mit Kopf-/Fußzeilen sind aber nicht zulässig.

<:O+[OVERSTRIKECHAR]> oder <:O->

Die erste Variante schaltet den Überschreibmodus ein. Die Sequenz enthält ein Feld mit dem Zeichen, welches zum Überschreiben verwendet wird. Die Sequenz <:O-> beendet den Modus *überschreiben*.

<:t[FRAMENUM]> oder <:A[FRAMENUM]>

Diese Sequenz definiert einen Rahmen zur Aufnahme einer Tabelle (<:t..>) und kann zur Verankerung der Tabelle (anchor escapes) benutzt werden (<:A..>). Das auf den Escape Code folgende Feld enthält eine (0 basierende) ASCII-Zahl mit dem Namen des Rahmen, auf den sich die Verankerung bezieht. Damit läßt sich eine Tabelle fest an einer Position auf einer Seite fixieren.

<:X[TYPE],[FLAG],[FIELDTEXT]>

Die Escape Sequenz für Felder (*fields* X) besteht aus mehreren Varianten:

- Enthält die X-Escape Sequenz einen Text, folgt dieser im Dokument. Der Text wird dann durch eine weitere X-Escape Sequenz beendet.
- Bei der zweiten Variante folgt eine Tilde (~) auf das Zeichen X. Diese Tilde markiert das Ende der Escape Sequenz.

Das Feld TYPE definiert den Feldtyp:

Wert	Bemerkung
1	DDE-Feld
2	Makre (Bookmark)
3	Standard (General result)
4	Sequenz Feld
5	Setzt eine globale Variable
6	Schaltfläche (Button field)
7	Printer Escape
8	Indexeintrag
9	benutzerdefinierte Marke

Tabelle 25.39 Feldtypen

Die Feldtypen 4, 5, 7 und 8 enthalten keinen Text und verzichten deshalb auf eine zweite Escape Sequenz zum Beenden.

Das Feld FLAG enthält die folgenden Einträge:

Flag	Bemerkung
0x1000	Auto evaluation field
0x2000	Locked
0x8000	Field is in main body

Tabelle 25.40 Kodierung Feld FLAG

Der Abschnitt FIELDTEXT ist optional und enthält den Text. Die Struktur des Textes wird im AMI Pro-Makrohandbuch beschrieben.

<:Z[BOOKMARK]>

Diese Escape Sequenz definiert den Namen einer Marke (bookmark). Der Name folgt direkt auf den Sequenzcode.

<:e[APPNAME],[TOPIC],[ITEM]>

Mit dieser Sequenz werden DDE-Verbindungen gespeichert. Für eine DDE-Verbindung wird die DDE-Sequenz dreimal geschrieben:

- Die erste Sequenz wird als <:e>, gefolgt vom Dokumenttext, gespeichert.
- Die nächste <e:>-Sequenz zeigt das Ende des betreffenden Textes an.
- Die dritte DDE-Escape Sequenz enthält schließlich die Parameter Anwendungsname (Application), das Thema (topic) und das Item.

<:d[FORMAT][DESCFIELD][NULL]>

Diese Escape Sequenz enthält das Feld für die Dokumentbeschreibung (*document description field*). Hier finden sich verschiedene Parameter wie Dateinamen, Datum, Zeit etc. zur Beschreibung des Dokuments. Das Feld FORMAT enthält nur das Zeichen 'a'. Das Byte im Feld DESCFIELD besitzt folgende Einträge:

Wert	Format
1	Dateiname
2	Pfad
3	Name Formatvorlage (Stylesheet)
4	Datum Erzeugung (created)
5	Datum letzte Änderung
6	Revisionszähler
7	Dokumentbeschreibung
8-15	Benutzerfelder
16-19	---
20	Zeit Erzeugung
21	Zeit letzte Änderung
22	Bearbeitungszeit (total)
23	Seitenzahl
24	Zahl der Worte
25	Zahl der Zeichen
26	Dokumentgröße

Tabelle 25.41 Inhalt des Feldes DESCFIELDInhalt des Feldes DESCFIELD

Die Einträge in diesem Feld werden als ASCII-Codes, gemäß den Escape Code-Konventionen abgespeichert. Das NULL-Feld enthält einfach den Wert 00H oder <*>.

<:n[OBJREFNUM]>

Diese Escape Sequenz definiert ein New Wave-Objekt. Daran schließt sich eine ASCII-Zahl mit der Referenznummer des New Wave-Objekts an. Die Referenznummer wird im Programm als DWORD kodiert.

<:k[FORMAT]>

Die *creation date* Escape Sequenz wird von einem Formatbyte gefolgt. Das Formatfeld wird im Rahmen der <d..>-Sequenz dokumentiert.

<:b[NULL]>

Die Escape Sequenz *creation time* wird durch 00H oder <*> gefolgt.

<:r[FORMAT]>

Dieser Satz folgt auf die *last revision date* Escape Sequenz (r). Das FORMAT-Feld ist in der <:d..>-Escape Sequenz dokumentiert.

<:V[S oder E][+ oder -]>

Die Escape Sequenz *revision marking* (V) wird von zwei Byte gefolgt. Der Buchstabe S markiert die Escape Sequenz als Anfang eines geänderten (revised) Textes. Das Zeichen E markiert das Ende dieses Textbereichs mit der Änderung. Das zweite Byte markiert die Änderungen als eingefügt (+) oder als gelöscht (-).

<:v[+ oder -]>

Absatzrevisionen werden über die *paragraph revision marking* Escape Sequenz (v) eingeleitet. Darauf folgt ein Byte, wobei das Zeichen + eingefügte Änderungen markiert. Das Zeichen – steht für gelöschte Korrekturen.

<:x[FOOTNOTENUM],[FOOTNOTEROW]>

Diese Escape Sequenz definiert Tabellen mit Fußnoten und wird von zwei ASCII-Zahlen gefolgt. Die erste Zahl definiert die Nummer der Fußnote. Die zweite Zahl definiert die Zeile in der Fußnotentabelle, in der die Fußnote erscheint.

Eingebettete Grafiken

In den Versionen 1.1 und 1.2 speichert Ami Pro alle eingebetteten Grafiken in separate Dateien. Diese werden mit dem Namen der Textdatei und den Erweiterungen .G00, .G01 etc. versehen.

Ab der Version 2.0 hängt AMI Pro alle Ojektdaten an das Ende der Dokumentdatei (hinter dem Text an). Alle AMI Pro-Versionen behandeln dabei ein > ohne ein vorhergehendes <-Zeichen als das Ende des Dokuments. An diese Marke schließen sich die Objektdaten im anwendungsspezifischen Format an. Hier wird auch die Begrenzung auf 7 Bit ASCII-Zeichen unterbrochen. AMI Pro hängt aber hinter allen Objektdaten eine sogenannte *final object directory* im ASCII-Format an. Für jedes Objekt wird dabei eine Zeile geschrieben. Die Zeile enthält den Objektnamen, den Dateityp (Numerierung beginnt mit 1), den Offset zu den Daten in der Datei (0=externe Referenz). Weiterhin werden die Felder Offset und Länge für einen Schnappschuß und ein weiterer Dateioffset und die Länge des Schnappschusses für das Objekt geführt. Der Eintrag 0 definiert, daß kein Schnappschuß vorliegt. Die sechs Felder pro Zeile werden als ASCII-Text kodiert und durch Leerzeichen getrennt. Alle Zahlen werden als Dezimalwerte geschrieben. Bei externen Objekten muß der Pfadname am Zeilenende mit angegeben werden.

Um dieses Inhaltsverzeichnis der Objekte zu finden, ist der Lesezeiger vom Dateiende 10 Byte in Richtung Dateianfang zu bewegen. Dann sind acht ASCII-Hexziffern, gefolgt von

einem CR/LF zu lesen. Die Hexzahl definiert den Offset in die Datei, an der die erste ASCII-Zeile des Objektverzeichnisses beginnt. Diese Zeile sollte dann mit einer eckigen Klammer [als Abschnittstitel (Section Title) anfangen. Dieser Titel kann folgendes Format aufweisen:

- [embedded] für das Objektverzeichnis
- [glossary] für das Glossar, das nach dem Textbereich gespeichert wird
- [macro] für einen P-Ccode Makro, der am Dateiende gespeichert wird.

Um den nächsten/vorhergehenden Abschnitt zu finden, positionieren Sie den Lesezeiger weitere 10 Byte vor dieser Zeile und wiederholen den Suchprozess. Die Suche ist zu beenden, falls die 10 Byte keine ASCII-Hexsequenz beinhalten, oder falls dieser Zeiger nicht auf eine Zeile mit einem [-Zeichen verweist.

Für den Image Structure Descriptor (ISD) sind folgende Dateierweiterungen zulässig:

.ole OLE-Link
.bmp Windows Bitmap
.wmf Windows Metafile
.tex AMI Equation
.sdw AMI Draw oder Chart
.tgf AMI Image Processor (Graustufen TIFF)

Tabelle 25.42 Dateierweiterungen für eingebettete Objekte

Der Dateityp kann jede gültige Erweiterung für externe Objekte mit 3 Zeichen sein (1.pcx 0 0 12334 1024 c:\pcx\pict1.pcx).

In zukünftigen Versionen soll diese Struktur beibehalten werden. Lediglich neue Felder werden hinzugefügt. Damit kann ein AMI Pro-Leser unbekannte Kommandos überspringen.

28 Das Animatic Film-Format (FLM)

Das Animatic Film Format wird auf dem Atari zur Speicherung von Bildsequenzen mit einer niedrigen Auflösung (320*200) und 16 Farben verwendet. Die FLM-Dateien enthalten dabei einen 62-Byte-Header, gefolgt von einem oder mehreren Teilbildern.

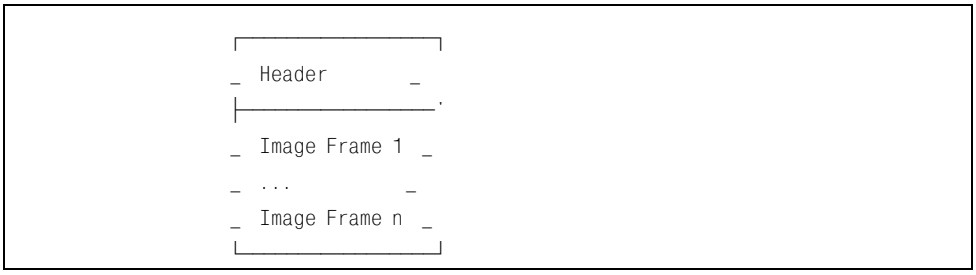


Abbildung 28.1 Struktur einer Animatic Film-Datei (FLM)

Die Daten werden im Motorola-Format (big endian) gespeichert.

Der Animatic Film-Header (FLM)

Der Header einer FLM-Datei wird gemäß Tabelle 28.1 aufgeteilt.

Offset	Bytes	Bemerkungen
00H	2	Einzelbilder (Number of Frames)
02H	16*2	Farbpalette
22H	2	Film Speed
24H	2	Play Direction
26H	2	Actionafter last Frame
28H	2	Frame width in Pixel
2AH	2	Frame height in Pixel
2CH	2	Versionsnummer (Major)
2EH	2	Versionsnummer (Minor)
30H	4	Signatur
32H	3*4	Reserviert (00H)

Tabelle 28.1 Struktur eines FLM-Headers

Das erste Wort enthält die Zahl der Bilder (Frames) in der Datei. Daran schließt sich eine Palette mit 16 Einträgen a 2 Byte an. Die Farbanteile werden dabei mit je 3 Bit abgelegt (siehe NEOchrome NEO-Format).

Die Abspielgeschwindigkeit (Film Speed) wird ab Offset 22H angegeben. Der Wert liegt im Bereich zwischen 0 bis 99 und gibt die Zahl der *Blank Frames* zwischen zwei Bildern an.

Der Wert 0 für die Abspielrichtung (Play Direction) definiert die Bildwiedergabe in der gespeicherten Reihenfolge. Mit dem Eintrag 1 in dem betreffenden Feld werden die Bilder in umgekehrter Reihenfolge wiedergegeben.

Das Wort ab Offset 26H definiert die Aktion, die nach der Ausgabe des letzten Bildes auszuführen ist.

Wert	Aktion
00H	Pause und neu abspielen
01H	sofort wiederholen (loop)
02H	--
03H	Playback in umgekehrter Richtung

Tabelle 28.2 Aktion nach dem letzten Bild

Die beiden Worte ab Offset 28H definieren die Abmessungen der Bilder (Frames) in Pixeln.

Die folgenden beiden Worte enthalten die Versionsnummer der Animatic Software, mit der die Animation erstellt wurde. Die Signatur findet sich ab Offset 30H und umfaßt vier Byte mit den Werten 27 18 28 18H. Die letzten 12 Byte sind reserviert und enthalten den Wert 00H.

An den Header schließt sich der Datenbereich mit den Bildern (Frames) an. Dieser Bereich enthält einen Abzug des Bildschirminhalts für jedes Bild (Frame).

29 Das ComputerEyes Raw Data Format (CE1,CE2)

Das ComputerEyes Raw Data-Format wird nur auf dem Atari zur Speicherung von Grafiken verwendet. Die Erweiterung CE1 steht dabei für Bilder mit einer niedrigen Auflösung (320*200), während CE2 Bilder mit mittlerer Auflösung (640*200) speichert. Die Daten werden dabei im Motorola-Format (big endian) abgelegt.

Der ComputerEyes Raw Data-Header (CEx)

Der Header einer CEx-Datei wird gemäß Tabelle 29.1 aufgeteilt.

Offset	Bytes	Bemerkungen
00H	4	Signatur 'EYES'
04H	2	Auflösung
06H	8*2	Reserviert

Tabelle 29.1 Struktur eines CEx-Headers

Die ersten vier Byte enthalten die Signatur 45 49 45 53, was dem String 'EYES' entspricht. Daran schließt sich das Wort mit der Auflösung an. Hierbei sind nur die Einträge:

```
0:Auflösung 320*200
1:Auflösung 640*200
```

erlaubt. Die restlichen 8 Worte des Headers sind für interne Zwecke reserviert.

An den Header schließt sich der Bilddatenbereich an. Bei einer Auflösung von 320*200 Bildpunkten werden die Daten in drei Ebenen gespeichert:

- 64000 Byte Ebene rot, 1 Pixel per Byte
- 64000 Byte Ebene grün, 1 Pixel per Byte
- 64000 Byte Ebene blau, 1 Pixel per Byte

Falls die mittlere Auflösung verwendet wird, enthält der Bilddatenbereich 128000 Bytes, die in 640*200 Bytes organisiert sind. Jedes Byte enthält den Wert für einen Bildpunkt (Pixel).

30 Das Cyber Paint Sequence Format (SEQ)

Dieses Format wurde ebenfalls für Atari-Rechner entwickelt und erlaubt die Speicherung von Bildsequenzen mit 16 Farben und einer niedrigen Auflösung (320*200). Die Daten werden dabei im Motorola-Format (big endian) gespeichert.

Der Cyber Paint Sequence-Header (SEQ)

Der Header einer SEQ-Datei wird gemäß Tabelle 30.1 aufgeteilt.

Offset	Bytes	Bemerkungen
00H	2	Signatur (FEDBH oder FEDCH)
02H	2	Versionsnummer
04H	4	Einzelbilder (Number of Frames)
08H	2	Display Speed
0AH	118	Reserviert
..H	n*4	Array mit Frame Offsets

Tabelle 30.1 Struktur eines SEQ-Headers

Die ersten zwei Byte enthalten die Signatur FEDBH oder FEDCH. Daran schließt sich eine Versionsnummer an. Ab Offset 04H wird die Zahl der Bilder (Frames) innerhalb der Datei angegeben. Das folgende WORD definiert die Verzögerung zwischen zwei Frames. Die 128 Byte ab Offset 0AH sind reserviert und werden auf 00H gesetzt. Der Header wird mit einem 4-Byte-Array abgeschlossen, welches die Offsets auf die einzelnen Bilder (Frames) enthält.

Der Aufbau der Frames

Jeder Frame beginnt wieder mit einem Header, gefolgt von den Bilddaten. Dieser Header besitzt den Aufbau aus Tabelle 30.2.

Offset	Bytes	Bemerkungen
00H	2	Frame Typ
02H	2	Frame Auflösung
04H	16*2	Color Palette
24H	12	Filename
30H	2	Color Animation Limits
32H	2	Color Animation Speed & Direction
34H	2	Zahl der Farbschritte (color steps)

Tabelle 30.2 Struktur eines Frame Headers

Offset	Bytes	Bemerkungen
36H	2	Frame x Offset (0)
38H	2	Frame y Offset (0)
3AH	2	Frame Breite (width)
3CH	2	Frame Höhe (height)
3EH	1	Graphikoperation
3FH	1	Komprimierung
40H	4	Frame Länge in Byte
44H	60	Reserviert (00)

Tabelle 30.2 Struktur eines Frame Headers

Der Header lehnt sich stark an das NEOchrome-Header-Format an. Das erste Wort identifiziert den Typ des Frames. Daran schließt sich ein Wort mit dem Code für die Auflösung an. Dieser Wert wird auf 00H (320*200 Pixel) gesetzt. Die Palette wird in 16 Worten im Atari-Format (siehe NEOchrome Format) abgelegt. Der Filename wird meist mit Leerzeichen belegt. Die Felder *color animation speed* und *number of color steps* bleiben unbenutzt.

Der x- Offset des Frames kann zwischen 0 und 319 liegen, der y-Offset ist im Bereich zwischen 0 und 199 zulässig. Die Bildabmessungen dürfen auf 0 gesetzt werden, da die Abmessungen des Frames fest auf 320*200 Bildpunkte gesetzt werden. Das Feld Operation beschreibt, wie die Daten eines Frames in den Bildschirm geschrieben werden.

```
0:kopieren
1:Exclusiv oder
```

Das Komprimierungs-Byte signalisiert mit dem Wert 1 einen komprimierten Datenbereich. Mit dem Eintrag 0 liegen die Bilddaten unkomprimiert vor.

Das Längenfeld bezieht sich immer auf den Datenbereich. Bei komprimierten Daten wird diese Länge angegeben. Die letzten 60 Byte im Frame-Header sind reserviert und füllen den Header auf 128 Byte auf.

Der Datenbereich folgt hinter dem Header und kann die Daten nach der Delta Compression-Methode aufnehmen. In diesem Fall wird lediglich der Unterschied zum vorhergehenden Frame gespeichert.

Hierbei wird ein Rechteck um den Bildbereich gelegt, der alle geänderten Bildpunkte enthält. Dieses Rechteck wird dann als Frame gespeichert. Die Abmessung und die Lage des Rechtecks werden im Frame-Header angegeben.

- Ein solcher Delta Frame kann dabei unkomprimiert abgelegt werden. Dann ist er mit einer einfachen Copy-Operation in den Bildbereich ab der Koordinate X,Y einzublenden.
- Alternativ lassen sich die Bilddaten mit XOR in den Bildbereich einblenden. Dieser Mode wird ebenfalls im Header vereinbart.
- Bei einem komprimierten Delta Frame müssen die Daten vor dem Einblenden in den Bildbereich decodiert werden. Dann ist ebenfalls eine copy- oder eine XOR-Operation möglich.
- Ein Null-Frame enthält eine Höhe und Breite von 0 Pixeln, d.h. der Frame entspricht genau dem vorhergehenden Frame.

Die Bilddaten werden nach diesen Kriterien im Datenbereich abgelegt, so daß der geringste Speicherplatz benötigt wird.

Das Komprimierungsverfahren

Zur Komprimierung wird ein einfaches Verfahren benutzt:

- Als erstes ist ein Word zu lesen. Dieses Wort dient als Kontrollelement.
- Ist der Wert dieses Wortes negativ (Bit 15=1), definiert der absolute Wert (0..32736) die Zahl der folgenden Worte, die aus dem Datenbereich unkomprimiert zu lesen sind.
- Bei einem Wert größer 0 dient das Kontrollwort als Zähler (0..32736). Das folgende Wort ist dann n mal zu kopieren.

Der Wert 00 ist zwar möglich, bleibt aber ohne Bedeutung.

Die entpackten Daten liegen in vier separaten Bitplanes vor. Jede Bitplane wird dabei vertikal organisiert. Die Bilddaten sind dann in den angegebenen Bildausschnitt einzufügen (Copy oder XOR).

31 Das Atari DEGAS-Format (PI*,PC*)

Zur Darstellung von Animationen mit Farbbildern wird auf dem Atari das DEGAS-Format benutzt. Von diesem Format existieren mehrere Varianten, die durch die Endungen .PI1, .PI2, .PI3, .PC1, .PC2 und .PC3 angezeigt werden. Die Auflösung wird durch die letzte Ziffer in der Erweiterung (1 low, 2 medium, 3 high) definiert. Der Buchstabe C in der Extension steht für komprimierte Bilddateien, während PI*-Dateien unkomprimiert vorliegen.

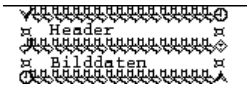


Abbildung 31.1 Struktur einer DEGAS-Datei

Der Header besitzt eine feste Größe von 140 Byte, gefolgt von einem Bilddatenbereich mit 32.000 Byte. Die Daten werden dabei im Motorola-Format (big endian) gespeichert.

Die DEGAS PI*-Dateien

Der Header einer PI*-Datei wird gemäß Tabelle 31.1 aufgeteilt.

Offset	Bytes	Bemerkungen
00H	2	Auflösung 0 = low 1 = medium 2 = high
02H	16*2	Color Palette

Tabelle 31.1 Struktur eines DEGAS PI*-Headers

Das erste Wort enthält die Auflösung, die in den unteren zwei Bit kodiert wird. Hierbei gelten die beim NEOchrome-Format beschriebenen Auflösungen.

Ab Offset 02H folgen 16 Worte mit der Farbpalette (color palette) für die im Bild benutzten Farben. Die Kodierung erfolgt dabei mit drei Bit pro Farbanteil (siehe NEOchrome Format).

An den Header schließt sich bei PI1-, PI2- und PI3-Dateien der Datenbereich mit den unkomprimierten Bilddaten an. Dieser besitzt immer eine feste Länge von 16000 Worten. Die Daten werden als Bildschirmabzug gespeichert.

Die DEGAS Elite PC*-Dateien

Bei den DEGAS Elite PC1-, PC2- und PC3-Dateien handelt es sich um komprimierte Bilddateien. Diese bestehen ebenfalls aus einem Header, gefolgt von einem Bilddatenbereich. Eine PC*-Datei wird gemäß Tabelle 31.2 aufgeteilt.

Offset	Bytes	Bemerkungen
00H	2	Auflösung 8000H low 8001H medium 8002H high
02H	16*2	Color Palette
22H	n	Bilddaten
..H	4*2	Start Farbnummern
..H	4*2	Ende Farbnummern
..H	4*2	Direction
..H	4*2	Animation Kanal Delay

Tabelle 31.2 Struktur einer DEGAS Elite PC*-Datei

Die Datei besteht aus einem Header, gefolgt von dem Datenbereich. Am Ende der Datei befindet sich eine Erweiterung der Headerdaten mit Informationen zur Animation.

Das erste Wort enthält die Auflösung, die in den unteren zwei Bit kodiert wird. Zur Markierung, daß es sich um eine DEGAS Elite-Datei mit komprimierten Daten handelt, wird Bit 15 = 1 gesetzt. Für die Auflösung gelten die beim NEOchrome-Format beschriebenen Auflösungen.

Ab Offset 02H folgen 16 Worte mit der Farbpalette (Color Palette) für die im Bild benutzten Farben. Die Kodierung erfolgt dabei mit drei Bit pro Farbanteil (siehe NEOchrome Format).

An den Header schließt sich bei PC1-, PC2- und PC3-Dateien der Datenbereich mit den unkomprimierten Bilddaten an. Dieser enthält einen Bildschirmabzug, der im Packbit-Verfahren komprimiert wurde.

- Die Daten werden dabei zeilenweise komprimiert.
- Die erste Bildzeile (0) beginnt am obereren Bildrand. Die nächsten Bildzeilen liegen dann unterhalb dieser Zeile.
- Die Daten einer Scanline werden ebenenweise (plane) komprimiert. Die erste Scanline enthält die Daten der niedrigsten Ebene.
- Das Packbit-Verfahren unterteilt die Werte in Records. Das erste Byte (n) dient als Steuerbyte. Ist der Wert kleiner 80H folgen n unkomprimierte Datenbytes. Falls das Steuerbyte Werte größer 80H enthält, enthält das folgende Byte den komprimierten

Wert. Dieser Wert ist dann n mal zu kopieren. Der Wert n errechnet sich als dem Steuerbyte zu: *n = 100H – Steuerbyte + 1*.

Nach diesem Verfahren sind n Zeilen mit m Bildpunkten zu dekodieren. Die Zahl der Zeilen und Bildpunkte ergibt sich aus der Auflösung des Bildes.

An den Bilddatenbereich schließen sich bei DEGAS-Elite Bildern noch zusätzliche Informationen zur Animation an. Die Animation ist für vier Kanäle ausgelegt. Die folgenden Informationen beziehen sich deshalb auf diese vier Kanäle. Als erstes findet sich eine Tabelle mit 4 Worten, die die linken Grenzen (*left color animation limit*) für die Farbanimation der Kanäle enthält. Die nächste Tabelle mit 4 Worten enthält die rechten Grenzen (*right color animation limit*) für die Farbanimation. Ein weiters Feld mit 4 Worten gibt die Richtung der Animation (*Animation channel direction*) an:

Code	Richtung
00H	Links
01H	Aus
02H	Rechts

Tabelle 31.3 Richtung der Animation

Das letzte Feld mit 4 Worten definiert die Verzögerungszeiten für die Kanäle während der Animation. Dieser Wert wird in 1/60 Sekunden angegeben und muß von 128 subtrahiert werden. Die Farbanimation ist speziell auf die Grafikmöglichkeiten der Atari-Rechner abgestimmt.

32 Das Atari Tiny-Format (TNY, TN*)

Das Tiny-Formt wurde ebenfalls zur Speicherung von Bildern auf dem Atari entwickelt. Es gibt mehrere Varianten, die sich durch die Erweiterung des Dateinamens unterscheiden:

*.TNY	beliebige Auflösung
*.TN1	niedrige Auflösung
*.TN2	mittlere Auflösung
*.TN3	hohe Auflösung

Tiny-Dateien bestehen aus einem Header, gefolgt von einem Bilddatenbereich. Die Daten werden dabei im Motorola-Format (big endian) gespeichert.

Eine Tiny-Datei wird gemäß Tabelle 32.1 aufgeteilt.

Offset	Bytes	Bemerkungen
00H	1	Auflösung 0 = low 1 = medium 2 = high >2 = Rotation Info
01H	4	Falls Auflösung > 2: 1 Byte left/right Color Limit 1 Byte Direction/Speed 1 Word Rotation Duration
..H	16*2	Color Palette
..H	2	Zahl der Kontrollbytes
..H	2	Zahl der Datenworte
..H	n	Kontrollbytes
..H	m	Datenbereich

Tabelle 32.1 Struktur einer Tiny-DateiStruktur einer Tiny-Datei

Das erste Byte enthält die Auflösung, die in den unteren zwei Bit kodiert wird. Hierbei gelten die beim NEOchrome-Format beschriebenen Auflösungen.

Bei Werten größer 2 enthält der Header zusätzliche Informationen zur Farbrotation. Diese Informationen werden nach dem ersten Byte im Header eingefügt. Das Byte ab Offset 01H definiert dann die Grenzen für die Farbanimation. Die oberen 4 Bits geben die linke Grenze (start limit) an, während die unteren 4 Bit die rechte Grenze (end limit) definieren. Das folgende Byte enthält den Code für die Animationsgeschwindigkeit (speed of color animation) und die Richtung. Negative Werte geben die Animationsrichtung links vor, während positive Werte die Richtung auf rechts setzen. Der absolute Wert

gibt die Animationsgeschwindigkeit in 1/60 Sekunden an. Das letzte Wort definiert die Dauer der Animation in Schritten (Number of iterations).

Auf diesen Einschub folgt ein Feld mit 16 Worten, welches die Farbpalette enthält. Die Kodierung wird beim NEOchrome-Format beschrieben. An die Palette schließt sich ein Wort an, welches die Zahl der Kontrollbytes angibt. Diese (3..10667) Kontrollbytes befinden sich zwischen Header und Datenbereich. Diese Kontrollbytes enthalten Informationen zur Entkomprimierung der Bilddaten. Das folgende Wort gibt die Zahl der Datenworte (1..16000) für den Bilddatenbereich an.

Die Bilddaten sind mit einer Lauflängenkodierung (RLE) komprimiert. Als Besonderheit werden die Kontrollbytes von den komprimierten Daten getrennt. Die Zahl der Kontrollbytes findet sich im Header. Diese Kontrollbytes dienen der Entschlüsselung der komprimierten Bilddaten. Hierbei gilt für jedes gelesene Kontrollbyte:

x < 0Ist der Wert negativ (größer 80H), gibt der absolute Wert die Zahl der Worte an, die aus dem Datenbereich zu lesen sind.

x = 0In diesem Fall ist das folgende Wort aus dem Kontrolldatenbereich zu lesen. Dieses Wort enthält einen Wiederholungszähler (128..32767). Dann ist das Wort aus dem Bilddatenbereich n mal zu wiederholen.

x = 1 Das folgende Wort des Kontrolldatenbereiches ist zu lesen. Dieses Wort enthält die Zahl der aus dem Bilddatenbereich zu lesenden Worte (128..32768).

x > 0Das Kontrolldatenbyte enthält einen Wiederholungszähler (2..127). Das folgende Wort aus dem Bilddatenbereich ist dann n mal zu wiederholen.

Die mit diesem Verfahren entschlüsselten Bilddaten stellen keinen Bildschirmabzug dar. Vielmehr werden die Daten des Bildschirms vertikal in Spalten aufgeteilt. Hierbei gelten vier Bereiche für die Spalten:

```
1: Spalten 1, 5, 9 etc.
2: Spalten 2, 6, 10 etc.
3: Spalten 3, 7, 11 etc.
4: Spalten 4, 8, 12 etc.
```

Eine Spalte enthält dann ein Wort aus der Scanline. Dann wird ein Wort aus der nächsten Spalte gelesen. So wird das Bild Zeile für Zeile komprimiert.

33 Das Atari Imagic Film/Picture-Format (IC*)

Das Imagic Film/Picture-Format wurde ebenfalls zur Speicherung von Bildern auf dem Atari entwickelt. Es gibt mehrere Varianten, die sich durch die Erweiterung des Dateinamens (.IC1, IC2, IC3) unterscheiden. Die letzte Ziffer gibt dabei die Auflösung (1 = low, 2 = medium, 3 = high) an.

Imagic Film/Picture-Dateien bestehen aus einem Header, gefolgt von einem Bilddatenbereich. Die Daten werden dabei im Motorola-Format (big endian) gespeichert. Der Header wird gemäß Tabelle 33.1 aufgeteilt.

Offset	Bytes	Bemerkungen
00H	4	Signatur 'IMDC'
02H	2	Auflösung (Resolution) 0 = low 1 = medium 2 = high
04H	2*16	Color Palette
24H	2	Date Stamp
26H	2	Time Stamp
28H	8	Filename
30H	2	Länge Datenbereich
32H	4	Registrationsnummer
34H	8	Reserviert
3CH	1	Kompressionsflag
3DH	3	IF Kompressionflag = 01H 1 Byte Kompression 1 Byte -- 1 Byte Escape Byte

Tabelle 33.1 Struktur eines Imagic Film/Picture-HeadersStruktur eines Imagic Film/Picture-Headers

Die ersten vier Byte enthalten eine Signatur, daran schließt sich ein Wort mit der Auflösung des Bildes an. Die Color Palette besteht aus 16 Worten. Die Kodierung erfolgt wie beim NEOchrome Format.

Ab Offset 24H findet sich das Datum und die Zeit der Dateierstellung (creation) im GEMDOS-Format.

Der Dateiname wird ab Offset 28H abgelegt. Das Wort ab Offset 30H definiert die Länge des Datenbereiches. Das Komprimierungs-Flag findet sich ab Offset 3CH. Mit dem Wert 00H liegen die Daten unkomprimiert vor. Ein Wert 01H weist auf komprimierte Bilddaten hin.

Ab Offset 32H steht eine Registriernummer, die nur für das erzeugende Programm von Bedeutung ist.

Sind die Bilddaten komprimiert, folgen drei Byte ab Offset 3DH. Ist das erste Byte auf FFH gesetzt, liegt eine RLE-Komprimierung vor. Ein anderer Wert zeigt die Delta-Kompression an, d.h. es werden nur Änderungen der Einzelbilder gesichert.

Für die RLE-Komprimierung läßt sich folgender Algorithmus zur Decodierung der gepackten Daten verwenden:


```
Escape Byte
Lese ein Byte
IF Byte >= 2 THEN
 kopiere nächstes Byte n mal
ENDIF

IF Byte = 1 THEN
 o = 0
 While n = 1 Then
  o = o+1
  Lese Byte n
 End
 k = n
 d = nächstes Byte
 kopiere d (256*o+k) mal
ENDIF
Not (Escape Byte)
```

Abbildung 33.1 Decodierung der RLE-Daten

Wurden die Daten dagegen im Delta Frame-Verfahren komprimiert, ergibt sich folgender Algorithmus zur Dekodierung:

```
Escape Byte
Lese ein Byte
IF Byte >= 3 THEN
 duplizieren das Folgebyte n mal
ENDIF
```

Abbildung 33.2 Decodierung Delta Frame-Daten

```
IF Byte = 2 THEN
 n = nächstes Byte
 IF n = 0 THEN End of Picture
 IF N >= 2 THEN
  Lese n Bytes aus Basisbild
 ENDIF
 IF N = 1 THEN
  o = 0
  While n = 1 Then
   o = o+1
   Lese Byte n
  End
  k = n
  d = nächstes Byte
  dupliziere d (256*o+k) mal
 ENDIF
ENDIF
Not (Escape Byte)
```

Abbildung 33.2 Decodierung Delta Frame-Daten

Beim Delta Frame-Verfahren wird im Header der Dateiname des Basisbildes (base picture) angegeben. Enthält dieses Feld keinen Eintrag (00H), liegt kein Basisbild vor.

34 Das Atari NEOchrome-Format (NEO)

Zur Darstellung von Farbbildern wird auf dem Atari das NEOchrome-Format benutzt. Die Dateien erhalten die Endung NEO. Wegen der Besonderheiten des Atari mit reduzierter Farbdarstellung blieb dieses Format auf diesen Rechnertyp begrenzt.

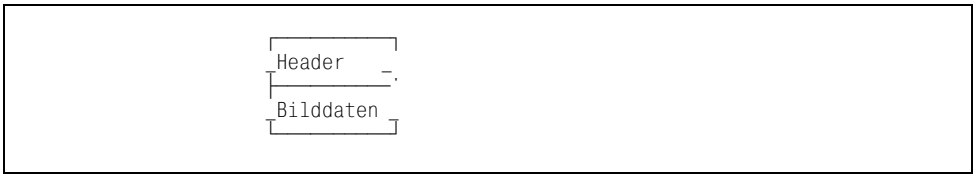


Abbildung 34.1 Struktur einer NEOchrome (NEO)-Datei

Das Format besteht aus einem Header mit einer festen Größe von 140 Byte, gefolgt von einem Bilddatenbereich mit 16.000 Byte. Die Daten werden dabei im Motorola-Format (big endian) gespeichert.

Der NEOchrome Header

Der Header einer NEO-Datei wird wortweise gemäß Tabelle 34.1 aufgeteilt.

Offset	Bytes	Bemerkungen
00H	2	Flags (0)
02H	2	Auflösung Bild
04H	16*2	Farbpalette
24H	12	Animation Filename
30H	2	Color Animation Limits
32H	2	Color Animation Speed & Direction
34H	2	Schrittzahl
36H	2	Bild x Offset (0)
38H	2	Bild y Offset (0)
3AH	2	Bildbreite
3BH	2	Bildhöhe
3CH	33*2	Reserviert (00)

Tabelle 34.1 Struktur eines NEO-Header

Das erste Wort enthält ein Flag, welches aber immer auf 0 gesetzt wird.

Das zweite Wort definiert die Bildauflösung. Bei NEOchrome-Dateien sind drei feste Auflösungen vereinbart.

Code	Auflösung
00H	Low (320*200, 16 Farben)
01H	Medium (640*200, 4 Farben)
02H	High (640*200, 2 Farben)

Tabelle 34.2 Auflösung NEOchrome-Bilder

Auf Grund der reduzierten Möglichkeiten zur Farbdarstellung nimmt die zulässige Anzahl an Farben mit höherer Auflösung ab.

Ab Offset 06 folgen 16 Worte, die die Farbpalette für das Bild enthalten. Die Farbpalette wird dabei mit 3 Bit pro Farbanteil gemäß Tabelle 34.2 gespeichert.

Bit	Farbe
0-2	blau
3	--
4-6	grün
7	--
8-10	rot
11-15	--

Tabelle 34.3 Kodierung Atari-Farbpalette

Soll ein solches Bild auf PCs übertragen werden, ist der angegebene Farbwert mit 32 zu multiplizieren.

An die Palette schließt sich der Bereich mit dem Filenamen der Datei an. Dieser Eintrag wird in der Regel mit Leerzeichen » . » aufgefüllt.

Das Wort ab Offset 30H definiert die Grenzen (color animation limits) innerhalb der sich die Farben eines Bildes variieren lassen:

Bits	Bemerkungen
0-3	Farblimit rechts/oben
4-7	Farblimit links/unten
15	1 = Animationsdaten gültig

Tabelle 34.4 Kodierung der Farblimits

Die (Farb-) Animation ist nur gültig, falls Bit 15 des Wortes auf 1 gesetzt ist. Andernfalls wird das Bild mit einer festen Farbkombination ausgegeben.

Das Wort ab Offset 32H definiert die Ausgabegeschwindigkeit für das Bild (Animation speed) und die Ausgaberichtung. Tabelle 34.4 gibt die Kodierung der einzelnen Bits des Wortes wieder.

Bits	Bemerkungen
0-7	Speed
15	Direction (0 normal, 1 reverse)

Tabelle 34.5 Kodierung Speed und Direction

Die Wiedergabegeschwindigkeit (playback speed) definiert die Zahl der Leerbilder (Blank frames) pro Animation Frame. Bit 15 definiert dabei die Richtung der Wiedergabe. Die Zahl der Frames innerhalb der Animation wird ab Offset 34H abgelegt.

Die Felder für den Offset der X- und Y-Achse des Bildes sind unbelegt und werden auf 0 gesetzt. Die Bildbreite ist immer fest mit 320 und die Bildhöhe mit 200 Pixel belegt. Die restlichen Felder werden zum Füllen des Headers benutzt.

Der Datenbereich der NEOchrome Datei

Der Datenbereich enthält die Bilddaten in unkodierter Form als Bildschirmabzug. Es werden Bilder mit 1, 2 und 4 Bit pro Bildpunkt gespeichert.

Bei 1 Bit pro Pixel liegen die Daten in einer Ebene vor. Ein Byte beschreibt dann 8 Bildpunkte, die im angegebenen Bildrasten auszugeben sind.

Bei 2 Bit pro Pixel lassen sich 4 Farben darstellen. Die Daten werden in zwei Ebenen abgelegt. Das erste Byte nimmt 8 Punkte der ersten Ebene auf. Das zweite Byte enthält die 8 Punkte der zweiten Ebene. Die zwei Byte müssen (dann z. B. beim Transfer auf PCs) bitweise zu den Farbwerten zu kombiniert werden.

Bei 4 Bit pro Pixel werden 16 Farben angezeigt. Die Daten liegen dann in vier Ebenen vor, wobei ein Byte immer 8 Bildpunkte einer Ebene enthält. Nachdem vier Byte gelesen wurden, sind diese bitweise für die Farbinformation eines Pixel zu kombinieren.

35 Das NEOchrome-Animation-Format (ANI)

Die NEOchrome Animations Files besitzen die Erweiterung ANI und erlauben die Wiedergabe von Bildsequenzen. Die ANI-Dateien enthalten einen 22-Byte-Header, gefolgt von einem oder mehreren Teilbildern zur Animation.

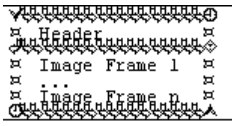


Abbildung 35.1 Struktur einer NEOchrome (ANI)-Datei

Die Daten werden dabei im Motorola-Format (big endian) gespeichert.

Der NEOchrome ANI-Header

Der Header einer ANI-Datei wird gemäß Tabelle 35.1 aufgeteilt.

Offset	Bytes	Bemerkungen
00H	4	Signatur
04H	2	Bildbreite
06H	2	Bildhöhe
08H	2	Bildgröße + 10
0AH	2	Bild X Koordinate-1
0CH	2	Bild Y Koordinate-1
0EH	2	Einzelbilder (Frames)
10H	2	Playback Speed
12H	4	Reserviert (00H)

Tabelle 35.1 Struktur eines ANI-Headers

Das erste Wort enthält die Signatur, die immer auf BA BE EB EA gesetzt wird. Daran schließt sich die Bildbreite an. Diese Bildbreite soll in Byte angegeben werden und der Wert muß durch 8 dividierbar sein. Die Bildhöhe findet sich ab Offset 06H und wird in Bildzeilen (scan lines) definiert. Das Wort ab Offset 08H definiert die Bildgröße + 10 in Byte. Ab Offset 0AH finden sich die Bildkoordinaten X,Y des Bildes. Der Wert der X-Koordinate definiert den linken Bildrand (in Pixel-1) und muß durch 16 teilbar sein. Die Y-Koordinate definiert den oberen Bildrand in Pixel-1.

Das Wort ab 0EH definiert die Zahl der Einzelbilder (Frames) in der ANI-Datei. Das folgende Feld gibt die Abspielgeschwindigkeit der einzelnen Bilder (Frames) an. Dieser Wert definiert die Zahl der Leerbilder (Blank frames) zwischen zwei Teilbildern.

Die letzten vier Byte des Headers sind reserviert und werden zu 0 gesetzt.

An den Header schließt sich der Datenbereich mit ein oder mehreren Bildern in der abzuspielenden Reihenfolge an. Hierbei gilt die im vorhergehenden Kapitel beim NEO-Format beschriebene Speicherung.

36 Das Atari STAD-Format (PAC)

Das STAD-Format wurde zur Speicherung von Monochrom-Einzelbildern mit 640*400 Bildpunkten auf dem Atari entwickelt. Eine PAC-Datei enthält einen Header und einen Datenbereich, der mit dem RLE-Verfahren gepackt wurde. Die Daten werden dabei im Motorola-Format (big endian) gespeichert. Der Header besitzt folgenden Aufbau:

Offset	Bytes	Bemerkungen
00H	4	Packrichtung
04H	1	RLE-ID-Byte als Packbyte
05H	1	Packbyte
06H	1	spezial RLE-ID-Byte

Tabelle 36.1 Struktur eines PAC-File-Headers

Die ersten vier Byte enthalten eine Signatur, die die Packrichtung der Daten angibt. Mit 'pM86' werden die Daten spaltenweise (vertikal) gepackt. Mit 'pM85' packt der Algorithmus die Daten zeilenweise (horizontal).

Die Bilddaten werden nach dem RLE-Verfahren hinter dem 7-Byte-Header abgelegt. Dabei können gepackte und ungepackte Records abwechseln. Gepackte Records werden durch eines der zwei RLE-ID-Bytes eingeleitet.

Das Byte ab Offset 05H im Header gibt den Wert des am häufigsten auftretenden Datenbytes (most frequently occurring byte) an. Dann muß dieses Byte nicht mehr im gepackten Datenbereich untergebracht werden.

Der Datenbereich kann damit drei unterschiedliche Recordtypen enthalten:

- Die erste Recordvariante umfaßt zwei Datenbytes. Das erste Datenbyte entspricht dem RLE-ID-Byte, welches ab Offset 04H im Header gespeichert ist. Das zweite Byte definiert einen Wiederholungszähler. Dann ist das Packbyte (im Header ab Offset 05H gespeichert) n mal zu kopieren.
- Die zweite Recordvariante umfaßt drei Datenbytes. Das erste Byte entspricht dem RLE-ID-Byte, welches im Header ab Offset 06H als spezial RLE-ID-Byte gespeichert wurde. Dann enthält das zweite Byte den Wiederholungszähler und das dritte Byte den eigentlichen Bildwert. Dieses dritte Byte ist dann n mal zu wiederholen.
- Die dritte Recordvariante enthält nur ein Byte, welches einen unkomprimierten Bildwert darstellt. Diese Variante tritt immer auf, wenn der Wert des gelesenen Bytes nicht mit den im Header angegebenen RLE-ID-Bytes übereinstimmt.

Die komprimierten Bilddaten sind zeilen- oder spaltenweise zu decodieren und im Raster 640*400 auszugeben. Jedes entpackte Datenbyte beschreibt 8 Bildpunkte. Die Rich-

tung, in der die Bildpunkte auszugeben sind (horizontal, vertikal) wird im Header angegeben.

37 Drawing Web-Format (DWF)

Die Darstellung von Grafiken im World Wide Web (WWW) beschränkt sich zur Zeit auf Pixelgrafiken im GIF- und JPEG-Format. Nur über spezielle Plug-In-Programme für die Browser von Netscape oder Microsoft lassen sich auch andere Bitmap-Formate anzeigen. Was bisher gänzlich fehlte, ist ein Format zur Anzeige von 2D-Vektorgrafiken in HTML-Dokumenten (die HTML 3.2-Spezifikation sieht hier nichts vor). Da die existierenden Vektorformate sehr spezifisch an bestimmte Anwendungen angepaßt sind, wurde 1996 von der Firma Autodesk das *Drawing Web Format (DWF)* für diesen Zweck definiert. Die bisher vorliegende Version 1.0 beschreibt die Basisbefehle, um 2D-Vektorgrafiken in einem WWW-Dokument darzustellen. Die Spezifikation unterstützt 2D-Vektorgrafiken, Beschriftungen im Unicode-Zeichensatz, Text- und Font-Operationen sowie URL-Adressen. (Das Format wurde aber nicht zum Datenaustausch von Vektordaten entworfen, d.h. ein Export komplexerer Vektorgrafiken in eine DWF-Datei mit anschließendem Re-Import in eine CAD-Anwendung ist nicht vorgesehen.) Zukünftige Revisionen des DWF-Formats sollen lediglich Zusatzbefehle für bestimmte Operationen festlegen.

Von Autodesk ist der WHIP!DWF File Toolkit erhältlich. Hierbei handelt es sich um eine C++-Bibliothek, die das Erzeugen und Parsen von DWF-Dateien unterstützt. Aufbauend auf diesem Toolkit stellt Autodesk Plug-In-Module für Netscape und den Microsoft Internet Explorer zur Verfügung, mit denen sich DWF-Dateien in WWW-Dokumenten anzeigen lassen. (Die bisherigen Implementierungen unterstützen jedoch erst einen Subset der definierten DWF-Befehle.)

Eine DWF-Datei besitzt einen blockorientierten Aufbau und besteht aus einem Header, dem Datenbereich und dem abschließenden Trailer-Record (Bild 37.1).

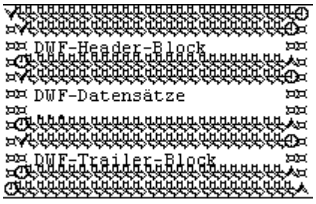


Abbildung 37.1 Struktur einer DWF-Datei

Die Daten des Headers und des Trailers sind dabei als ASCII-Text definiert. Der DWF-Datenbereich besteht aus einzelnen Datensätzen, wobei die Daten der DWF-Datei dabei durch jeweils einen Opcode, gefolgt von den Operanden, gebildet werden. Als Besonderheit kann der Opcode samt der zugehörigen Operanden entweder als ASCII-String oder als Binärdatensequenz auftreten. Dies kompliziert m.E. die Dekodierung der DWF-Datei, da keine direkte Unterscheidung der Opcodes anhand des Recordaufbaus möglich ist. Ein DWF-Reader muß zwar nicht alle Opcodes implementieren. Damit das Programm

aber die Datensätze korrekt lesen und trennen kann, ist unbedingt erforderlich, daß der Reader die betreffenden Opcodes samt der zugehörigen Recordlänge kennt.

Der DWF-Header

Jede DWF-Datei beginnt mit einem Header, der aus einem 12 Byte langen ASCII-String besteht. Der Header wird mit zwei Klammern eingeschlossen und besteht aus der Zeichenkette:

```
(DWF V01.00)
```

Die ersten sechs Byte »(DWF V« sind dabei immer konstant. Die restlichen Bytes enthalten einen Versionscode, der bisher mit »01.00)« definiert ist (steht für die Version 1.0). Die Zahl vor dem Dezimalpunkt gibt dabei die Hauptversion an, während die Ziffern hinter dem Punkt die Unterversion festlegen. Eine Anwendung, die eine DWF-Datei erzeugt, sollte als Versionsnummer immer die notwendigerweise niedrigste Versionsnummer angeben, die zum Lesen der Datei erforderlich ist. Ein Reader besitzt dabei kaum eine Chance, DWF-Dateien zu lesen, deren Hauptversion höher ist als die Versionsnummer, deren Opcodes implementiert wurden.

Der DWF-Trailer

Eine DWF-Datei wird mit dem Opcode *(EndOfDWF)* abgeschlossen. Damit erkennt der DWF-Reader, daß keine weiteren Bilddaten mehr folgen, obwohl die Anwendung durchaus noch Daten hinter den DWF-Trailer schreiben kann. Ein Beispiel für eine sehr einfache DWF-Datei könnte folgendermaßen aussehen:

```
(DWF V01.00)
(DrawingInfo
  (SourceFilename Haus.DWG)
  (Description 'Entwurf für das Haus')
  (Creator 'AutoCAD R13C4')
  (Created 820497600 'January 1, 1997')
  (Author 'G. Born')
  (Bounds 1, 1 1000,1000)
  (Scale 0.001 0.001 meter)
)
(Comment Dies ist ein einfacher DWF-Kommentar! )
(View 30,30, 800,800)
(Layer 1 Grundriss)
  C 132
  L 25,30 200,300  L 320,320 500,500
(Layer 2 Installation)
```

```
C 12
  L 20,20 150,300  L 320,320 500,500
 (URL 'http://www.addison-wesley.de')
(EndOfDWF)
```

Der DWF-Datenbereich

Der DWF-Datenbereich besteht aus einzelnen Records, die ab dem dreizehnten Byte hinter dem Header beginnen. Jeder Datensatz besteht dabei aus dem Opcode, gefolgt von den zugehörigen Operanden. Eine DWF-Datei kann dabei sowohl lesbare ASCII-Befehle als auch Binärdaten enthalten. Beim Einlesen der Opcodes wird daher folgende Unterscheidung getroffen:

- *Single-Byte-Opcodes:* Diese wurden aus Effizienzgründen eingeführt und enthalten Binärdaten. Der Reader muß den Opcode kennen, um die Länge des zugehörigen Datensatzes mit den Operanden zu bestimmen.
- *Extended ASCII-Opcodes:* Alle lesbaren Opcodes werden als ASCII-Strings geschrieben, wobei die Operanden durch Separatoren getrennt werden. Solche Datensätze werden durch Klammern (..) eingeschlossen. Der Befehl *(Origin 100 100)* stellt zum Beispiel einen solchen Befehl dar. Als Besonderheit dürfen mehrere Opcodes geschachtelt werden. Der Record *(Owner (Person 'Born Guenter') (Company 'Addison Wesley'))* enthält zum Beispiel mehrere Extended ASCII-Opcodes. Ein DWF-Leser besitzt die Möglichkeit, diese Datensätze samt den zugehörigen Operanden zu überlesen, indem er für jede öffnende Klammer nach der schließenden Klammer »)« sucht. Wird im Zeichenstring ein Hochkomma ' gefunden, muß der Reader das den String abschließende Hochkomma ' suchen. Schließende Klammern innerhalb dieser String-Sequenz sind zu ignorieren. Mit dem Backslash \ wird angezeigt, daß das folgende Zeichen nicht als Literal zu interpretieren ist. Damit können Sie Hochkommas ' oder Backslash-Zeichen im Befehl einbinden. Die Zeichenkette *Dies ist/war das 'Sonderangebot' des Sommers* muß dann folgendermaßen kodiert werden: (Comment *'Dies ist\\war das (\'Sonderangebot\') des Sommers'*). Beachten Sie, daß hier die beiden Klammern, die das Wort *'Sonderangebot'* einhüllen, zum String gehören und nicht den Klammern im ASCII-Record entsprechen.
- *Extended Binary Opcodes:* Dieser Opcode enthält eine Längenangabe, mit der sich die Länge des Datensatzes ermitteln läßt. Ein solcher Datensatz wird immer mit zwei geschweiften Klammern {..} eingeschlossen. Sobald also ein Datensatz mit dem Zeichen { beginnt, liegt ein Extended Binary Opcode vor. Der Datensatz besitzt dabei folgende Struktur: {cccceexxxxxxxx}, wobei die Zeichen *cccc* für eine 4-Byte-Integerzahl stehen, die die Länge des Binärdatenbereichs in Byte angibt. An die Längenangabe schließt sich ein 2-Byte-Opcode an, der hier mit den beiden Zeichen *ee* dargestellt wurde. Die Zeichen *xxxx* stehen für eine Binärdatensequenz von n Byte, die anschließend durch

ein }-Zeichen abgeschlossen wird. Bei der Längenangabe werden der 2-Byte-Opcode und das }-Zeichen mitgezählt. Die Werte werden dabei im *Little-endian*-Format in der Datei gespeichert, d. h. auf Motorola-Systemen müssen Binärdaten erst byteweise vertauscht werden. Ein DWF-Reader kann anhand der Längenangabe einen *Extended Binary Record* überlesen. Allerdings gibt es Sonderfälle, in denen als Längenangabe 0 vorgegeben ist. Dann konnte das schreibende Programm die Länge nicht bestimmen. Erkennt der DWF-Reader den Opcode nicht, kann die Datei nicht dekodiert werden.

Den einzelnen Opcodes darf dabei eine beliebige Anzahl an *White-Space*-Zeichen (Leerzeichen, Tabulator, CR, LF) vorangehen. Weiterhin können zwischen einzelnen Parmetern eines Extended ASCII-Records solche *White-Space*-Zeichen stehen. Damit können DWF-Dateien in Zeilen aufgeteilt werden, und die jeweiligen Anweisungen in einer Zeile lassen sich einrücken. Dies ist hilfreich, falls eine DWF-Datei aus lesbaren Zeichen besteht und ggf. ausgedruckt werden soll.

DWF-Koordinatensystem

DWF-Dateien benutzen ein logisches Koordinatensystem, welches auf die Koordinaten des Anzeigegeräts abzubilden ist. Die logischen Koordinaten dürfen dabei im Bereich zwischen 0 und 2.147.483.647 Einheiten (entspricht dem positiven Bereich einer 4-Byte-Integerzahl) liegen. Durch die Verwendung einer Integerzahl zur Koordinatendarstellung erlaubt Ihnen die DWF-Spezifikation Karten mit einer Breite von 21.000 Kilometern und einer Auflösung von 1 Zentimeter darzustellen. Abhängig vom Opcode werden dabei absolute oder relative Koordinaten vorgegeben. Beachten Sie hierbei, daß absolute Koordinatenangaben dabei nur den positiven Bereich einer 4-Byte-Integerzahl umfassen dürfen. Relative Koordinatenangaben umfassen dagegen den positiven und negativen Zahlenbereich, der mit einem 32-Bit-Integerwert darstellbar ist. Zusätzlich sind noch 16-Bit-Integerwerte für die Koordinatenangaben vieler Datensätze zulässig. Der Nullpunkt für ein Bild liegt in der linken unteren Ecke.

Die DWF-Opcodes

Das erste Byte eines Datensatzes legt den Opcode fest. Die beiden Zeichen '(' und '{' stehen dabei für die Opcodes, die erweiterte Datensätze im ASCII- und Binärdatenformat einleiten (siehe oben). Verschiedene andere Zeichen sind als *White-Spaces* definiert, dürfen folglich nicht als Opcodes auftreten. In einer DWF-Datei sind die folgenden ASCII-Zeichen nicht als Opcodes zugelassen:

Leerzeichen (20H), Tabulator (09H), Bindestrich – (2DH), die Ziffern 0 – 9 (30H bis 39H), Carriage Return (0AH), Anführungszeichen » (22H), Hochkomma ' (27H), Punkt . (2EH), Klammern (..) (28H, 29H), geschweifte Klammern {..} (7BH, 7DH), eckige Klammern [..] (5BH, 5DH), Backslash \ (5CH).

Die in der Revision 1.0 der DWF-Spezifikation definierten Opcodes werden nachfolgend beschrieben.

Comment

Dieser (Extended ASCII-) Datensatz erlaubt Ihnen die Speicherung von Kommentaren. Der Datensatz besitzt dabei folgenden Aufbau:

```
(Comment <Kommentartext>)
```

Der Datensatz wird durch runde Klammern eingeschlossen. Der Kommentartext kann durch mehrere White-Space-Zeichen vom Befehl *Comment* getrennt werden. Die Angabe *(Comment Dies ist ein Kommentar)* ist ein Beispiel für diesen Befehl. Der Comment-Befehl wird nicht in der Vektorgrafik angezeigt. Der DWF-Reader kann aber eine Option zur Anzeige der Kommentare bereitstellen.

Define Compressed Data

Dieser (Extended Binär-) Datensatz enthält einen komprimierten DWF-Datenstrom, der aus Opcode-Operandenpaaren besteht. Diese werden dann nach einem RLE-Verfahren komprimiert. Der Datensatz beginnt mit der geschweiften öffnenden Klammer {, gefolgt von dem 32-Bit-Längenwert und dem 2-Byte-Opcode 01H 23H. An den Opcode schließen sich dann die komprimierten Binärdaten an. Diese enthalten andere DWF-Records (Opcodes und Operanden). Der Datensatz wird mit einer geschweiften Klammer } abgeschlossen.

Zur Komprimierung wird ein einfacher RLE-Algorithmus verwendet, der sich wiederholende Zeichenketten zusammenfaßt. Ein Datensatz besteht aus folgenden sich ggf. wiederholenden Bytes:

Bytes	Bemerkung
1	Compression-Code
1	Extended Literal Run-Length (optional)
n	Literal-Datenbereich (optional)
1	Extended Compressed Run-Lenght (optional)
2	Offset-Code (optional)

Tabelle 37.1 Aufbau eines komprimierten Records

Auf den *Define Compressed Data*-Opcode folgen die komprimierten Daten. Abgesehen vom ersten Byte sind die anderen in Tabelle 37.1 aufgeführten Bytes optional und werden je nach Kontext im Datensatz eingefügt.

Das erste Byte enthält den *Compression-Code*, der in zwei 4-Bit-Hälften unterteilt wird. Die unteren vier Bit des *Compression Code* geben an, wie viele Bytes im *Literal-Datenbereich* folgen. Dieser Literal-Datenbereich folgt ab Byte 3 (oder Byte 2, falls der Wert des *Extended Literal Run-Length* fehlt) und enthält unkomprimierte Datenbytes. Die unteren vier Bits im *Compression-Code* werden daher auch als *Literal Data Run Length* bezeichnet. Die vier Bit können dabei folgende Werte aufweisen:

0:	Der Wert 0 signalisiert, daß keine Daten im Literal-Datenbereich vorliegen. Damit fehlt auch das *Extended Literal Run-Length* Byte. Als nächstes Byte kann entweder ein *Offset-Code* (16 Bit) oder ein *Extended compressed Run Length*-Byte folgen.
1-14:	Werte zwischen 1 bis 14 zeigen an, daß eine entsprechende Anzahl an Datenbytes im Literal Datenbereich vorliegt.
15:	Der Wert 15 signalisiert, daß ein *Extended Literal Run Length*-Byte folgt. Damit lassen sich auch *Literal-Datenbereiche* mit mehr als 14 Byte Länge darstellen. Der Wert im *Extended Literal Run Length*-Byte kann dabei zwischen 0 und 255 liegen, was einer Run-Length von 15 bis 270 entspricht (auf das *Extended Literal Run Length*-Byte wird die Zahl 15 addiert).

Die oberen vier Bit des *Compression-Code* geben an, wie viele Bytes im Originaldatenbereich durch den angegebenen Offset-Code komprimiert wurden. Dieser als *Compressed Data Run Length* bezeichnete Wert ist folgendermaßen zu interpretieren:

0:	Der Wert 0 signalisiert, daß keine Daten komprimiert werden konnten. Damit ist das auf den Datenbereich folgende Byte ein weiterer *Compression-Code* (dann ist der Dekomprimier-Algorithmus ab diesem Byte zu wiederholen).
1-14:	Werte zwischen 1 bis 14 zeigen an, daß im Quelldatenbereich ein String zwischen 4 und 17 Byte Länge komprimiert wurde. Der komprimierte String wird durch den 2-Byte-Offset-Code adressiert, der sich hinter dem *Literal-Datenbereich* anschließt.
15:	Der Wert 15 signalisiert, daß ein *Extended Compressed Run Length*-Byte auf den Literal-Datenbereich folgt. Auf dieses Byte folgt dann der 2-Byte-Offsetwert. Das *Extended Compressed Run Length*-Byte kann Werte zwischen 0 und 255 aufnehmen. Diese Werte stehen für Run-Length-Werte von 18 bis 273 (auf das *Extended Compressed Run Length*-Byte wird die Zahl 18 addiert).

An den Offset-Wert schließt sich ggf. der nächste zu dekomprimierende RLE-Record an. Zum Komprimieren schreibt das Programm nicht komprimierbare Zeichen in den Record. Gleichzeitig werden diese unkomprimierten Zeichen in einem *History-Puffer* angehängt. Findet das Programm einen (mindestens vier Zeichen umfassenden) Teilstring, der bereits im *History-Puffer* steht, wird nur noch der Offset auf den History-Puffer und die Länge des Originalstrings hinterlegt.

Bei der Dekomprimierung benutzt das Programm den Offset-Code als einen Index in einen History-Puffer. Befindet sich ein Teilstring in diesem History-Puffer, muß die Anwendung nur *n* Bytes aus dem Puffer in den Ausgabestream schreiben, um die Originaldaten zu restaurieren. Der Offset 0 steht dabei für das älteste im Puffer gespeicherte Byte. Werden unkomprimierte Bytes gelesen (Literal Data) und in den Ausgabedatenstrom ge-

schrieben, sind diese ebenfalls an den History-Puffer anzuhängen. Dadurch wird der History-Puffer automatisch aufgebaut und erlaubt, komprimierte Daten zu ermitteln.

Die Sequenz zum Dekomprimieren endet, sobald das erste Byte mit dem *Compression-Code* den Wert 0 aufweist. Damit wird der Datensatz beendet, und die schließende Klammer } muß folgen. Die DWF-Spezifikation enthält ein Beispiel, welches die Komprimierung des Strings:

SHE SELLS SEASHELLS BY THE SEASHORE

verdeutlicht. Dieser String wird dann folgendermaßen kodiert:

Bytes(Hex)	Bemerkungen
2F	*Compression Code*, 5 komprimierte Bytes (5 = 3 + Wert 2 aus den oberen 4 Bits), und ein *Extended Literal-Datenbereich* folgt (untere vier Bit sind auf FH gesetzt)
00	*Extended Literal Run Length* von 0 ergibt eine Länge von 15 Zeichen im Literal-Datenbereich.
53 ...	Literal-Datenbereich mit dem String: SHE SELLS SEASH
05 00	Zwei Byte (Little-endian) *Offset-Code*, der Wert 0005 zeigt in den History-Puffer (auf das Zeichen E im Literal-Datenbereich).
24	*Compression-Code*, 5 komprimierte Datenbytes (2+3), 4 unkomprimierte Bytes im *Literal-Datenbereich*.
42 ...	Literal-Datenbereich mit dem String: BY T
01 00	Zwei Byte Offset-Code 0001 in den History-Puffer (zeigt auf den Buchstaben H im ersten Literal-Datenbereich).
06	*Compression-Code*, 0 komprimierte Bytes, 6 unkomprimierte Bytes im Literal-Datenbereich.
41	Literal-Datenbereich mit dem String: ASHORE
00	*Compression Code* 0 terminiert die RLE-Kodierung
7D	}-Zeichen zum Abschluß des Records

Tabelle 37.2 Beispiel für einen komprimierten RLE-Datensatz

Beachten Sie aber, daß DWF-Daten standardmäßig nicht komprimiert abgespeichert werden. Damit dürfte dieser Recordtyp nicht innerhalb der DWF-Datei auftreten.

Define Drawing Aspect Ratio

Dieser (Extended ASCII-) Datensatz definiert das als *Aspect Ratio* bezeichnete Seitenverhältnis (Höhe/Breite) der Vektorgrafik. Die Verhältniswerte sind auf die logischen Koordinatenangaben anzuwenden, um ein korrektes Bild zu erhalten. Das Seitenverhältnis wird als Fließkommazahl (normalerweise 1.0) angegeben. Ein gültiger Befehl besitzt dann beispielsweise das Format:

```
(Aspect 1.0)
```

Der Wert für das Aspect Ratio sollte nur auf die *X/Y*-Koordinatenangaben angewandt werden (und nicht auf andere logische Einheiten wie z. B. Radien). Der Wert läßt sich auch aus den Werten *ScaleY/ScaleX* bestimmen, die mit dem Opcode *Define Drawing Scale* definiert werden. Eine DWF-Datei sollte einen *Drawing Aspect Ratio*-Opcode aufweisen, der im *Define Drawing Information Block*-Opcode geschachtelt ist.

Define Drawing Author

Dieser (Extended ASCII-) Opcode gibt einen String mit dem Namen des Autors an, der die DWF-Datei erzeugt hat. Ein Record könnte folgenden Aufbau haben:

```
(Author 'G. Born, xy-Company')
```

Standardmäßig ist kein Autor definiert, und der Datensatz sollte im *Define Drawing Information Block*-Opcode geschachtelt werden.

Define Drawing Background

Dieser (Extended ASCII-) Datensatz gibt die Hintergrundfarbe an, mit der geometrische Primitiven durch die DWF-Anwendung zu zeichnen sind. Ein solcher Datensatz besitzt das Format:

```
(Background <Rot>,<Grün>,<Blau>,<Alpha>)
(Background <Index>)
```

Hinter dem Opcode *Background* folgen ein oder mehrere Leerzeichen. Dann wird die Hintergrundfarbe durch die Angabe der Farbanteile *Rot, Grün, Blau* und den *Alpha*-Kanal festgelegt. Die Werte können dabei zwischen 0 und 255 liegen (entspricht 0 bis 100% Farbanteil; bzw. beim Alpha-Kanal bedeutet der Wert 0, daß die Hintergrundfarbe vollständig transparent ist). Alternativ kann die Hintergrundfarbe über einen Indexwert in eine Farbpalette definiert werden. Bevor eine Anwendung mit der Ausgabe graphischer Primitive beginnt, sollte der Zeichenbereich gelöscht werden. Die Hintergrundfarbe des Fensters wird dabei durch den *Define Drawing Background* Record definiert. In einer DWF-Datei sollte dieser Recordtyp nur einmal auftreten. Standardmäßig wird der Befehl auf *(Background 0,0,0,0)* gesetzt.

Define Drawing Bounds

Die (Extended ASCII-) Funktion *Define Drawing Bounds* definiert ein Rechteck mit den logischen Koordinaten, innerhalb dessen alle graphischen Primitive liegen sollen. Der Befehl besitzt das Format:

```
(Bounds X1,Y1 X2,Y2)
```

Die Koordinaten *X1,Y1* legen die logischen Koordinaten der linken unteren Ecke des Rechtecks fest. Die rechte obere Ecke wird durch die Koordinaten *X2,Y2* definiert. Der

betreffende Befehl teilt einem DWF-Reader mit, in welchen Bereich des Koordinatensystems die Ausgaben erfolgen sollen. Ein solcher Befehl könnte zum Beispiel folgende Angaben enthalten:

```
(Bounds 10,20 1500,1500)
```

Innerhalb der DWF-Datei sollte dieser Record nur einmal innerhalb des *Define Drawing Information Blocks* auftauchen. Standardmäßig wird der gesamte verfügbare Koordinatenraum für die Ausgaben genutzt.

Define Drawing Creation Time

Dieser Datensatz enthält Informationen über das Datum, an dem die DWF-Datei erzeugt wurde. Es gilt folgende Syntax:

```
(Created <Zeit> <Beschreibung>)
```

Der Record wird als *Extended ASCII*-Opcode kodiert. Im Feld *Time* findet sich die Zeitdifferenz zwischen der Systemzeit beim Erzeugen der Datei bis zum 1. Januar 1970 (00:00:00). Diese Zeitdifferenz wird dabei in Sekunden angegeben. Das Feld *Beschreibung* kann dagegen einen beliebigen lesbaren Ausdruck enthalten, der das Datum und die Zeit angibt, zu der die Datei erzeugt wurde. Ein Beispiel für einen solchen Record könnte folgendermaßen aussehen: *(Created 820497600 '1.Jan.1997')*. Innerhalb der DWF-Datei sollte dieser Datensatz nur einmal innerhalb des *Define Drawing Information Blocks* auftreten. Standardmäßig ist keine Creation-Time definiert, was dem Befehl *(Created 0 unknown)* entspricht.

Define Drawing Creator

Dieser (Extended ASCII-) Datensatz enthält Informationen über das Programm, welches die DWF-Datei erzeugt hat. Hierbei wird folgende Syntax verwendet:

```
(Creator <Name>)
```

Im Feld *Name* kann der Name des erzeugenden Programms stehen. Innerhalb einer DWF-Datei sollte dieser Datensatz nur einmal innerhalb des *Define Drawing Information Blocks* auftreten. Ein solcher Datensatz könnte zum Beispiel folgenden Eintrag *(Creator 'AutoCAD R13c4')* aufweisen. Standardmäßig wird kein solcher Eintrag definiert, was dem Befehl *(Creator unknown)* entspricht.

Define Drawing Description

Dieser Datensatz wird im Extended ASCII-Format definiert und kann Informationen über den Inhalt der DWF-Datei beschreiben. Hierbei gilt folgende Syntax:

```
(Description <Beschreibung>)
```

Das Feld *Beschreibung* kann dabei einen einfachen ASCII-Text enthalten (z. B. *(Description 'Entwurf Fertigungshalle 19')*). Die DWF-Datei sollte diesen Datensatz nur einmal im *Define Drawing Information Block* enthalten.

Define Drawing Information Block

Dieser (Extended ASCII-) Block wirkt als Container, um andere Opcodes mit Informationen über den Inhalt der DWF-Datei aufzunehmen. Der Block besitzt dabei folgende Syntax:

```
(DrawingInfo <Info>)
```

Das Argument *Info* enthält die Liste mit den jeweiligen *Define Drawing ....* Opcodes (wurden teilweise auf den vorhergehenden Seiten vorgestellt. In diesem Block dürfen folgende Opcodes auftreten: *Define Drawing Author, Define Drawing Initial View, Define Drawing Creator, Define Drawing Subject, Define Drawing Creation Time* und *Define Drawing Modification Time*. Die Informationen im *Drawing Information*-Block kann der Reader auswerten, bevor er mit der Ausgabe der Zeichnung beginnt. Daher muß dieser Block, falls vorhanden, direkt hinter dem DWF-Header folgen. Damit kann eine Anwendung dem Benutzer z. B. die Suche nach einem bestimmten Autor erlauben. Hierzu muß sie lediglich die Header der jeweiligen DWF-Dateien lesen.

Define Drawing Modification Time

Dieser (Extended ASCII-) Datensatz enthält Informationen über das Datum und die Zeit, zu der die DWF-Datei zuletzt geändert wurde. Hierbei gilt folgende Syntax:

```
(Modified Time Beschreibung)
```

Im Feld *Time* werden die Sekunden seit dem 1. Januar 1970 0:00:00 (GMT) bis zur letzten Änderung angezeigt. Im Feld *Beschreibung* kann das Änderungsdatum in lesbarer Form stehen. Ein gültiger Datensatz wäre z. B. *(Modified 820497600 '1.Januar.1996')*. Das Argument *Beschreibung* sollte nicht durch den DWF-Reader ausgewertet, sondern lediglich angezeigt werden. Der Record muß im *Define Drawing Information* Block eingefügt werden. Standardmäßig ist keine Änderungszeit definiert, was dem Befehl *(Modified 0 unknown)* entspricht.

Define Drawing Scale

Dieser (Extended ASCII-) Datensatz beschreibt die Skalierung (Real-World Scale) der logischen DWF-Koordinaten. Hierbei gilt folgende Syntax:

```
(Scale ScaleX ScaleY Units)
```

Der Parameter *ScaleX* gibt eine Fließkommazahl (normale oder doppelte Genauigkeit, aber in ASCII-Darstellung) mit der Größe der Real-World-Achse (horizontale Achse) an.

In *ScaleY* folgt die ASCII-Darstellung der Größe der (vertikalen) Real-World-Y-Achse. Beide Größenangaben erfolgen in einer bestimmten Einheit, die im Parameter *Units* festgelegt wird. Durch diesen Befehl lassen sich die logischen Koordinatenangaben der Zeichnung in reale Maße umrechnen. Ein Befehl könnte zum Beispiel folgenden Aufbau haben: *(Scale 0.001 0.001 meter)*. Wird anschließend ein Zeichenkommando wie z. B. *(L 0,0 5000,0)* gefunden, bedeutet dies, daß die vom Befehl *L* gezogene Linie genau 5 Meter lang ist.

Der DWF-Reader muß die in der Datei benutzte Einheit aber nicht zwingend kennen, eine Maßangabe in einer Grafik könnte sich ja auch auf physikalische Größen wie Temperatur, Druck etc. beziehen. Nur wenn eine exakte Skalierung der auszugebenden Grafik (z. B. bei Bemaßungen in technischen Zeichnungen) erforderlich ist, wird der Datensatz benötigt. (Die meisten DWF-Reader kennen jedoch nur die Einheiten *inch* und *meter*.) Wird dieser Datensatz verwendet, sollten die Parameter aber als *double-precision*-Fließkommazahlen in der entsprechenden ASCII-Darstellung hinterlegt werden. Die Konvention sieht vor, daß der Opcode im *Define Drawing Information* Block auftritt. Standardmäßig ist keine Skalierung definiert, was dem Befehl *(Scale 1.0 1.0 undefined)* entspricht.

Define Initial View

Dieser (Extended ASCII-) Datensatz liefert dem DWF-Reader einen Hinweis, welcher Ausschnitt des logischen Koordinatenraums zu Beginn anzuzeigen ist. Der Datensatz besitzt folgende Syntax:

```
(View X1,Y2 X2,Y2)
```

Die Parameter *X1,Y1* liefern die logischen Koordinaten der linken unteren Ecke des anzuzeigenden Rechteckbereichs. *X2,Y2* legen dagegen die logischen Koordinaten der rechten oberen Ecke des anzuzeigenden rechteckigen Bereichs fest. Der Datensatz ist hilfreich, wenn nur Ausschnitte des logischen Koordinatenraums mit Zeichenobjekten belegt sind. Der Datensatz sollte daher möglichst am Anfang der DWF-Datei (hinter dem *Define Drawing Information* Block) stehen. Allerdings erlaubt die DWF-Spezifikation, daß eine Datei mehrere (ggf. sich widersprechende) *Define Initial View*-Befehle enthält. Der Reader kann dann folgende Regeln zur Anwendung eines Ausschnitts verwenden:

- Wird die DWF-Datei als Resultat einer URL-Angabe geladen und enthält die URL-Anweisung eine Angabe für die Bildgröße, sollte diese verwendet werden.
- Enthält eine DWF-Datei einen *view*-Opcode, ist dessen Darstellungsbereich zu verwenden.
- Enthält die DWF-Datei einen *Define Bounds*-Opcode, sind die in diesem Opcode definierten *drawing extents* für den anzuzeigenden Abschnitt zu verwenden.

- Liegen keine Informationen vor, ist nach dem Lesen der gesamten DWF-Datei die Ausdehnung der Grafikobjekte zu bestimmen. Aus deren Koordinaten ist dann der anzuzeigende Bildbereich zu ermitteln.
- Alternativ kann der gesamte logische Koordinatenraum zur Anzeige verwendet werden.

Die Verwendung von View-Informationen aus URLs ist in der DWF-Spezifikation 1.0 aber noch nicht festgelegt. Da obige Regeln recht komplex sind, wird es bei der praktischen Implementierung wohl darauf hinauslaufen, daß entweder ein View-Kommando verwendet oder der gesamte Koordinatenraum angezeigt wird. Die Möglichkeit, mehrere View-Opcodes mit unterschiedlichen Anzeigeausschnitten in einer DWF-Datei zu hinterlegen, könnte allerdings auch zur Animation einer anzuzeigenden Grafik benutzt werden. Standardmäßig legt der erste auftretende View-Opcode im *Define Drawing Information* Block den Anzeigebereich für die Ausgabe fest. Fehlt die View-Angabe, ist der gesamte Koordinatenraum (Default) zur Anzeige zu verwenden.

Define Marker Glyph

Die Funktion *Define Marker Glyph* fügt ein neues Grafikobjekt (Glyph) zu einer Sammlung graphischer Primitiven hinzu oder ändert die Definition eines Objekts in dieser Sammlung. Der Datensatz enthält dabei eine Sammlung von DWF-Opcodes, um das betreffende Objekt (den Marker bzw. das Symbol) zu zeichnen. Solche graphischen Objekte könnten zum Beispiel Kreuze, Sternchen, Pfeile etc. sein. Jedes graphische Objekt (Glyph) wird dabei als eine Sequenz an Zeichenfunktionen definiert. Bei der Ausgabe der Zeichnung reicht es dann, das betreffende Objekt anzugeben, der DWF-Reader muß das Objekt dann aus den in der Bibliothek hinterlegten Opcodes erzeugen. Der *Define Marker Glyph*-Datensatz kann entweder als Extended ASCII-Code oder als Extended Binary-Sequenz kodiert werden. Für die Extended ASCII-Darstellung gilt folgende Syntax:

```
(Glyph <Index> <X,Y> <Unit> <Definition1>)
```

Wird der Datensatz im Extended Binary-Format hinterlegt, besitzt dieser das eingangs beschriebene Format. Der Datensatz beginnt mit einer geschweiften öffnenden Klammer {, an den sich die 32-Bit-Längenangabe anschließt. Der Glyph-Opcode besitzt den Wert 0003H. Anschließend folgen die Parameter als Binärdatensequenz:

```
{xxxx0003<Index><X><Y><Unit><Definition2>}
```

Die Parameter sind für beide Recordtypen folgendermaßen definiert:

- Der Parameter *Index* definiert das Element in der Marker Glyph-Tabelle, welches hinzuzufügen oder zu ersetzen ist. Gültige Werte des *Index* liegen zwischen 0 und 65535. Im Binärformat liegt der Parameter als *unsigned short Integer* (2 Byte) vor.

- Die beiden Parameter *X,Y* geben den logischen Koordinatenursprung für das zu definierende Symbol (Glyph) an. Im Binärformat besteht jeder Parameter aus einer vorzeichenbehafteten 4-Byte-Integerzahl.
- Im Parameter *Unit* wird die Größe der Einheit (in logischen Koordinaten) für das Symbol (Glyph) definiert. Der Wert tritt bei der binären Darstellung als vorzeichenlose 32-Bit-Integer auf.
- Bei Extended ASCII-Datensätzen steht der Parameter *Definition1* für eine geschachtelte Serie mit lesbaren ASCII-Opcodes, die zum Zeichnen des betreffenden Symbols zu benutzen sind. Diese sind in der Marker Glyph-Tabelle an der durch den Parameter *Index* spezifizierten Position zu hinterlegen.
- Bei Extended Binary-Datensätzen steht der Parameter *Definition2* für eine Serie an geschachtelten Opcodes, die das Symbol beschreiben. Die Opcode-Sequenz ist in der Marker Glyph-Tabelle an der durch *Index* spezifizierten Position zu hinterlegen.

Mit dem *Define Marker Glyph*-Opcode kann ein Programm innerhalb der DWF-Datei eine Symbolbibliothek definieren. Die so vereinbarten graphischen Primitiven lassen sich anschließend mit dem *Draw Polymarker*-Opcode mehrfach ausgeben. Mit dem *Set Marker Glyph*-Opcode läßt sich das Symbol auswählen, und der *Set Marker Size*-Opcode definiert die Symbolgröße. Da die *Define Marker Glyph*-Operanden aus einer Folge von Opcodes bestehen, kann der DWF-Writer beliebig komplexe Symbole erzeugen und in der Bibliothek hinterlegen. Hierbei wird ein Satz an logischen Koordinatenangaben zur Definition benutzt. Beim Zeichnen des Symbols durch die *Draw Polymarker*-Funktion werden diese Koordinaten mittels der Argumente *Origin* und *Unit* umgerechnet. Nachfolgende Anweisungen definieren ein Kreuz als Symbol:

```
(Glyph 12 300,300 100 (
L 250,300 350,300(Comment Horizontale Line)
L 300,250 300,350(Comment Vertikale Line)
))
```

Hier wird das Symbol mit dem Index 12 in der Tabelle hinterlegt. Der Nullpunkt für das Symbol wird auf dem Punkt 300,300 festgelegt, wobei als Größe (Unit) 100 Einheiten im logischen Koordinatensystem benutzt werden. Beim Zeichnen dieses Symbols mittels *Draw Polymarker* wird der Nullpunkt des Symbols auf den angegebenen Ausgabepunkt umgerechnet. Dann wird das Symbol gemäß den Angaben im *Unit*-Parameter skaliert (die Skalierung wird so ausgeführt, daß die Größe des Symbols zum definierten *Current Marker Size*-Attribut paßt).

Innerhalb der Symboldefinition können beliebige geometrische Primitiven (*Draw* Opcodes) und/oder Attribute (*Set* Opcodes) verwendet werden. Die meisten Symbole enthalten jedoch keine Attribute. Dann werden die aktuellen Attribute bei der Ausgabe des Symbols mittels *Draw Polymarker* verwendet. Die nachfolgende Sequenz nutzt das ge-

rade definierte Symbol (das Kreuz mit dem Index 12) und gibt dieses mit 30 Einheiten in den Farben Rot und Blau aus:

```
G 12(Comment Wähle das Symbol Nummer 12)
S 30(Comment Kreuz soll 30 Einheiten groß sein)
 (Color 255,0,0,255)(Comment Farbe Rot)
M 1 500,400(Comment Kreuz auf 500,400 zeichnen)
 (Color 0,0,255,255)(Comment Farbe Blau)
M 1 600,700(Comment Kreuz auf 600,700 ausgeben)
```

Soll das Symbol direkt Attribute verändern (z.B. ein rotes Kreuz), können Sie diese Attribute innerhalb der Symboldefinition setzen. In der oben gezeigten *Glyph*-Definition wäre dann vor den *L*-Befehlen ein Opcode zur Farbauswahl einzufügen:

```
(Glyph 13 300,300 100 (
   (Color 255,0,0,255)(Comment Farbe Rot wählen)
   L 250,300 350,300(Comment Horizontale Line)
   L 300,250 300,350(Comment Vertikale Line)
))
```

Sobald dieses Symbol mit *Draw Polymarker* ausgegeben wird, setzt der DWF-Reader das Farbattribut auf *Rot*. Vorher ist allerdings das aktuelle Farbattribut zu sichern, und nach der Ausgabe des Symbols ist das alte Attribut zu restaurieren.

Bezieht sich ein *Define Marker Glyph*-Opcode auf den *Index* eines bereits bestehenden Eintrags, muß der DWF-Reader die bestehende Definition überschreiben. Damit läßt sich ein Symbol innerhalb der Tabelle redefinieren. Bei den meisten Symbolen wird der Koordinatennullpunkt in der Bounding Box des Symbols zentriert sein. Die *Unit*-Angabe legt dann die Höhe und Breite der Bounding Box fest. Dies muß aber nicht zwangsweise so sein, denken Sie zum Beispiel an Zeichen, die ebenfalls als *Glyph* definiert werden. Jedes Zeichen wird dann als ein solches Symbol vereinbart. In diesem Fall gibt die *Unit* die Höhe der Zeichen an (wobei sich die Höhe am höchsten Zeichen der Schriftart orientiert und für alle Zeichen gleich ist). Der Nullpunkt für ein solches Zeichen stimmt in der Regel mit der Grundlinie der Schrift überein, d.h. es wird der Punkt in der linken unteren Ecke der Grundlinie angegeben, an dem das Zeichen einzufügen ist. Unterlängen für ein Zeichen liegen dabei unterhalb des jeweiligen Nullpunkts (d.h. die Unit gibt den Abstand von der Grundlinie zum oberen Rand der Buchstaben der jeweiligen Schriftart an).

Wird die Extended ASCII-Version des *Define Marker Glyph*-Opcode benutzt, müssen auch die Opcodes zum Zeichnen des Symbols im Parameter *Definition1* als ASCII-Opcodes hinterlegt werden. Damit kann der DWF-Reader ggf. den gesamten Datensatz überspringen. Treten Extended Binary-Opcodes im Parameter *Definition2* auf, sollte auch die binäre Version des *Define Marker Glyph*-Befehls benutzt werden. Beim Aufbau der *Marker Glyph*-Tabelle sollten folgende Symbole standardmäßig definiert werden:

Index	Bemerkung
0	Einzelner Punkt
1	Plus-Zeichen
2	X-Zeichen
3	Stern
4	Kreis (nicht gefüllt)
5	Kreis (gefüllt)

Tabelle 37.3 Standard-SymboleStandard-Symbole

Die Implementierung dieser Symbole hängt jedoch von der schreibenden Anwendung ab.

Define Source Drawing Creation Time

Dieser (Extended ASCII-) Datensatz teilt dem DWF-Leser mit, wann die Quelldatei angelegt wurde, aus der die DWF-Datei generiert wurde. Damit kann eine Versionskontrolle erfolgen, die unabhängig von der Version der DWF-Datei ist (es wird das Erzeugungsdatum der ursprünglichen 2D-Vektorgrafik angegeben). Dieser Befehl besitzt folgende Syntax:

```
(SourceCreated <Time> <Beschreibung>)
```

Der Parameter *Time* gibt die Zeit in Sekunden an, die seit dem 1. Januar 1970 0:00:00 (GMT) bis zum Erzeugen der Quelldatei verstrichen ist. Im Parameter *Beschreibung* kann ein lesbarer String für den Benutzer hinterlegt sein. Ein Anweisung sieht dann z. B. so aus:

```
(SourceCreated 820497600 '1.Januar.1996')
```

Der Opcode sollte im *Define Drawing Information*-Block auftreten. Standardmäßig ist dieser Parameter nicht definiert.

Define Source Drawing Filename

Der (Extended ASCII-) Datensatz *Define Source Drawing Filename* enthält die Information, welche Datei zum Generieren der DWF-Datei benutzt wurde. Damit kann der DWF-Reader dem Benutzer ggf. den Namen der Quelldatei anzeigen. Der Befehl besitzt folgende Syntax:

```
(SourceFilename <Name>)
```

Im Parameter *Filename* wird der Name der Quelldatei, die zum Erzeugen der DWF-Grafik verwendet wurde, hinterlegt. Ein gültiger Befehl könnte zum Beispiel folgenden Eintrag aufweisen:

```
(SourceFilename 'Fabrikhalle.DWG')
```

Die Spezifikation empfiehlt, daß lediglich der Dateiname ohne weitere Pfadangaben anzugeben ist. Der Opcode sollte innerhalb des *Define Drawing Information*-Blocks auftreten, und es sollte auch mindestens ein *Source Creation Time*-Opcode vorhanden sein. Standardmäßig ist kein Eintrag für *SourceFilename* definiert, d.h. es gilt *(SourceFilename unknown)*.

Define Source Drawing Modification Time

Dieser (Extended ASCII-) Datensatz enthält die Information, wann die Quelldatei, die zum Erzeugen der DWF-Datei verwendet wurde, zum letzten Mal modifiziert wurde. Hierbei gilt folgende Syntax:

```
(SourceModified <Time> <Beschreibung>)
```

Der Parameter *Time* gibt dabei die Sekunden an, die seit dem 1.Januar.1970 0:00:00 (GMT) bis zur letzten Modifikation verstrichen sind. Im Parameter *Beschreibung* ist dann ein String mit der Zeitangabe zu hinterlegen, der vom Benutzer gelesen werden kann. Ein gültiger Befehl wäre zum Beispiel *(SourceModified 820497600 '1.January.1996')*. Die Angabe sollte im *Define Drawing Information*-Block auftreten. Standardmäßig ist der Eintrag nicht definiert und entspricht dem Befehl *(SourceModified 0 unknown)*.

Draw Circle/Circular Arc/Circular Wedge

Dieser Datensatz wird mit einem Opcodebyte gespeichert und kann einen Kreis, einen Kreisbogen oder einen Kreisausschnitt (Tortenstück bzw. Circular Wedge) darstellen. Die Parameter für diesen Opcode können dabei im ASCII-Format oder als Binärwert hinterlegt werden. Die ASCII-Variante des Befehls besitzt dabei die folgende Syntax:

```
(R <X,Y> <R> <Start,Ende>)
```

Damit wird entweder ein Kreis oder ein Kreisbogen gezeichnet. Der Buchstabe *R* legt fest, daß es sich um die ASCII-Variante des Kreisbefehls handelt. Die Parameter *X,Y* geben die absoluten Koordinaten des Kreismittelpunkts an, während *R* den Radius in absoluten Koordinaten definiert. Mit *Start* und *Ende* wird der Anfangs- und Endpunkt auf dem Kreisbogen definiert. Die Werte definieren Winkel in der Einheit 360/65536 Grad. Der Kreisbogen wird dabei gegen den Uhrzeigersinn gezeichnet. Die Werte dürfen zwischen 0 und 65536 liegen.

Bei dem Single-Byte-Opcode mit Binäroperanden gibt es mehrere Varianten, die verschiedene Kreiselemente zeichnen. Für einen Vollkreis mit relativen Koordinatenangaben gilt folgende Syntax:

```
r<X><Y><Radius>
```

Der kleine Buchstabe *r* (Code 72H) definiert, daß die folgenden Parameter als Binärdaten hinterlegt sind. Mit *X, Y* und *Radius* werden absolute Koordinatenangaben definiert. *X,Y*

beziehen sich auf den Kreismittelpunkt und werden als vorzeichenbehaftete 32-Bit-Integer gespeichert. Der Radius wird als vorzeichenlose 32-Bit-Integer festgelegt. Alternativ existiert ein Befehl, um einen Vollkreis mit relativen Koordinatenangaben zu zeichnen. Dieser besitzt folgende Syntax:

```
12H<X><Y><Radius>
```

Der Opcode 12H entspricht dem Tastencode Ctrl-R und wird gefolgt von relativen Koordinatenangaben für den Kreismittelpunkt (*X,Y* vorzeichenbehaftete 16-Bit-Integerwerte) sowie dem Radius (vorzeichenlose 32-Bit-Integer). Die dritte Variante erlaubt, einen Kreis oder einen Kreisbogen mit relativen Koordinatenangaben zu zeichnen. Hierbei gilt folgende Syntax:

```
92H<X><Y><Radius><Start><Ende>
```

Bei diesem Datensatz wird der Opcode 92H verwendet. Daran schließen sich die relativen Koordinaten für den Kreismittelpunkt *X,Y* als vorzeichenbehaftete 32-Bit-Integerzahlen an. Weitere Parameter sind der Radius (32 Bit unsigned Integer) und die Winkelangaben für den Start- und Endpunkt des Kreisbogens (vorzeichenlose 16-Bit-Integer).

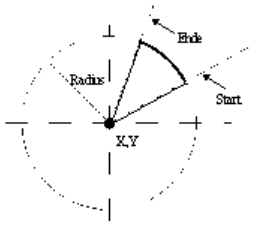


Abbildung 37.2 Parameter für einen Kreis

Ist das *Fill Mode*-Attribut inaktiv (Standardeinstellung), zeichnet die *Draw Circle*-Funktion nur die Kreislinien (basierend auf den aktuellen Zeichenattributen *Color*, *Visibility*, *Line Weight*, *Line Cap* und *Line Pattern*). Bei aktiviertem *Fill Mode*-Attribut wird die Fläche des Kreises oder des Kreisausschnitts mit der auf den Polygon-Zeichenattributen (*Color*, *Visibility*) basierenden Farbe gefüllt.

Einige der Opcodes erlauben es, einen Kreisabschnitt über die Parameter *Start* und *Ende* zu zeichnen. Ist der *Fill*-Modus ausgeschaltet, wird nur der Kreisbogen dargestellt. Bei eingeschaltetem Füllmodus muß der DWF-Reader dagegen ein »Tortenstück« für einen Kreisausschnitt zeichnen. Die Start- und Endwerte werden als Integer im Einheiten von 360/65536 Grad angegeben. Der DWF-Reader muß den Kreisbogen dabei entgegengesetzt zum Uhrzeigersinn vom Startwinkel zum Endwinkel zeichnen. Ist der *Fill Mode* aktiv, wird der außen liegende Teil der Kreislinie nicht dargestellt. Daher muß der DWF-

Writer ggf. einen weiteren *Draw Circle*-Datensatz mit abgeschaltetem Füllmodus generieren, um die Kreislinie zu zeichnen.

Draw Ellipse/Elliptical Arc/Elliptical Wedge

Diese Funktion ermöglicht das Zeichnen einer Ellipse oder eines Ellipsenausschnitts oder einen elliptischen Kreisbogens. Ähnlich wie bei der Funktion zum Zeichnen von Kreisen gibt es auch hier verschiedene Varianten von Opcodes. Die ASCII-Variante des Befehls besitzt dabei die folgende Syntax:

```
(E <X,Y> <Rh,Rv> <Start,Ende> <Tilt>)
```

Damit wird eine Ellipse mit absoluten Koordinatenangaben gezeichnet. Der Buchstabe *E* legt fest, daß es sich um die ASCII-Variante des Ellipsenbefehls handelt. Die Parameter *X,Y* geben die absoluten Koordinaten des Ellipsenpunkts an, während *Rh* den horizontalen Radius der Ellipse in absoluten Koordinateneinheiten definiert. Mit dem Parameter *Rv* wird dagegen der vertikale Radius der Ellipse festgelegt. Die Parameter *Start* und *Ende* legen den Anfangs- und Endpunkt auf dem Ellipsenumfang fest. Die Werte definieren Winkel in der Einheit 360/65536 Grad. Der Ellipsenbogen wird gegen den Uhrzeigersinn gezeichnet. Die Werte dürfen dabei zwischen 0 und 65536 liegen. Der Parameter *Tilt* gibt den Neigungswinkel der Ellipse gegen den Uhrzeigersinn an. Dieser Parameter wird erst angewandt, nachdem die anderen Parameter bearbeitet wurden. Die Werte dürfen dabei zwischen 0 und 65536 liegen.

Bei dem Single-Byte-Opcode mit Binäroperanden gibt es eine Variante zum Zeichnen der Ellipse oder eines Ellipsenbogens mit relativen Koordinatenangaben. Hierbei gilt folgende Syntax:

```
e<X,Y> <Rh,Rv> <Start,Ende> <Tilt>
```

Der kleine Buchstabe *e* (Code 65H) definiert, daß die folgenden Parameter als Binärdaten hinterlegt sind. *X,Y* werden als vorzeichenbehaftete 32-Bit-Integer definiert, während es sich bei *Rh,Rv* um vorzeichenlose 32-Bit-Integer handelt. Die Parameter *Start, Ende* und *Tilt* werden als vorzeichenlose 16-Bit-Integerwerte abgelegt. Es gilt die gleiche Parameterbelegung wie bei der ASCII-Version (siehe oben).

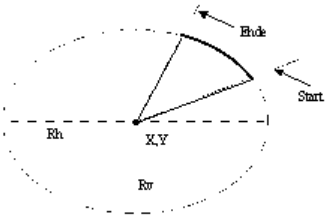


Abbildung 37.3 Parameter der Ellipse

Die Funktion zeichnet entweder einen Ellipsenbogen (Füllmodus ist aus) oder eine elliptische Fläche (das *Fill Mode*-Attribut ist auf *Ein* gesetzt). Bei abgeschaltetem Füllmodus werden die Attribute *Color*, *Visibility*, *Line Weight*, *Line Cap* und *Line Pattern* übernommen. Bei eingeschaltetem Füllmodus werden dagegen Flächen gezeichnet, der DWF-Reader darf dann die Randlinie nicht ausgeben.

Draw Gouraud Polytriangle

Diese Funktion zeichnet eine komplexe Figur, die aus aneinandergelegten Dreiecksflächen besteht. Anschließend werden die außen liegenden Linien der Figur gezeichnet.

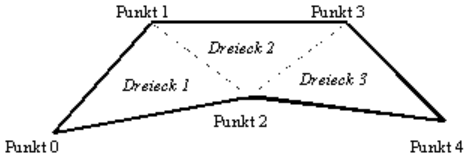


Abbildung 37.4 Polytriangle mit fünf Stützpunkten

Über diese Dreiecke lassen sich beliebige Flächen erzeugen. Die Funktion besitzt zwei Opcodes. Die (Extended ASCII-) Variante besitzt folgende Syntax:

```
(Gouraud <Count> <X1>,<Y1> <R1>,<G1>,<B1>,<A1> <X2>,<Y2>
<R2>,<G2>,<B2>,<A2> [<Xi>,<Yi> <ri>,<gi>,<Bi>,<ai>] )
```

Der Opcode zeichnet die miteinander verbundenen Dreiecke in absoluten Koordinaten. Für den *Single-Byte-O*pcode mit binären Parametern gibt es zwei Varianten:

```
g <Count> <Ecount> <X1><Y1><C1><X2><Y2><C2>[<Xi><Yi><Ci>]
```

Dieser Opcode gibt die Daten in absoluten Koordinaten an. Der folgende Opcode spezifiziert die Koordinaten der Dreiecke in relativen Koordinaten.

```
07H <Count> <Ecount> <X1><Y1><C1><X2><Y2><C2>[<Xi><Yi><Ci>]
```

Der Code 07H steht dabei für *Ctrl-G*. Bei der ASCII-Variante werden alle Parameter als Integer definiert. In [ ] stehende Parameter wiederholen sich ggf. mehrfach. Die Parameter werden teilweise durch Kommas und durch White Space-Zeichen separiert. Bei den binären Varianten besitzen die Parameter unterschiedliche Längen. Für die Parameter gilt folgende Belegung:

- In *Count* wird die Zahl der zu zeichnenden Dreiecke angegeben. (Dieser Wert ist immer um zwei kleiner als die Zahl der Scheitelpunkte.) In der binären Variante umfaßt dieser Parameter ein (vorzeichenloses) Byte. Ist das Byte 0, zeigt dies an, daß die Zahl der Dreiecke im folgenden Parameter *Ecount* steht.

- Der Parameter *Ecount* wird als vorzeichenlose 16-Bit-Integer angegeben und erlaubt, die Zahl der Dreiecke zwischen 256 bis 65791 festzulegen (zum Wert sind 256 zu addieren).
- In *X,Y* stehen die Koordinaten des ersten Scheitelpunkts des ersten Dreiecks der Liste. Beim *g*-Datensatz werden alle Koordinatenangaben als vorzeichenbehaftete 32-Bit-Integergrößen gespeichert. Beim Datensatz mit dem Code 07H sind die Koordinaten als vorzeichenbehaftete 16-Bit-Integerwerte anzugeben.
- Die Parameter *R1*, *G1*, *B1* und *A1* legen die Farbe für die vom ersten Scheitelpunkt zu ziehende Linie als Anteile *Rot*, *Grün*, *Blau* und *Alpha*-Kanal fest. Jeder Parameter kann Werte zwischen 0 und 255 aufweisen.
- Bei den binären Varianten wird im Parameter *C1* die Farbe des ersten Scheitelpunkts als 4-Byte-Wert hinterlegt. Diese vier Byte legen die Farben *Rot*, *Grün*, *Blau* und den *Alpha*-Kanal fest.

Für jeden weiteren Scheitelpunkt enthält der Datensatz die Parameter *X,Y* für die Koordinaten, und *C* definiert die Farbe (bzw. die Farbe läßt sich als *R*, *G*, *B*, *A* angeben). Hierbei werden die gleichen Datentypen verwendet (siehe oben). Die zeichnende Funktion muß die außen liegenden Scheitelpunkte miteinander verbinden. Hierzu wird vom ersten Scheitelpunkt eine Linie durch alle ungeraden Scheitelpunkte gezogen. Anschließend wird die Linie vom letzten ungeraden Scheitelpunkt durch die geraden Scheitelpunkte bis zum ersten Scheitelpunkt gezogen. Damit ergibt sich eine Figur in Form eines Vielecks. Die Figur wird abhängig vom Füllmodus als Umrißlinie oder als Fläche gezeichnet. Hier gelten die gleichen Bedingungen wie bei den Funktionen zum Zeichnen von Kreisen. Um eine *Gouraud Polytriangle*-Funktion mit mehr als 65791 Dreiecken anzuwenden, müssen Sie mehrere *Draw Gouraud Polytriangle*-Opcodes speichern.

Draw Image

Diese Funktion erlaubt es, ein ein- oder zweidimensionales Pixelmuster zu zeichnen. Damit lassen sich Bilder im GIF-, TIFF- oder JPEG-Format in diesem Datensatz hinterlegen und ausgeben. Die Daten für die Funktion können in einer ASCII-Variante oder als Binärdaten vorliegen. Für die ASCII-Version gilt folgende Syntax:

```
(Image <Col>,<Row> <X1>,<Y1> <X2>,<Y2> <Format> [[,]<Daten>])
```

Die einzelnen numerischen Parameter werden als Integerzahlen in der ASCII-Darstellung hinterlegt. Die in [ ] stehenden Parameter sind optional. Beim Extended Binary-Datensatz gilt folgende Syntax:

```
{...0x0002 <Col><Row><X1><Y1><X2><Y2>'<Format>'[<Daten>]}
```

Für die einzelnen Parameter gilt folgendes:

- Der Parameter *Col* legt die Zahl der Pixel pro Zeile fest. In *Row* findet sich die Zahl der Bildzeilen. Beim Binärdatensatz werden die Werte als vorzeichenlose 16-Bit-Zahlen hinterlegt.
- Die Parameter *X1,Y1* und *X2,Y2* beschreiben die logischen Koordinaten der linken unteren (X1,Y1) und der rechten oberen Ecke des auszugebenden Bildes. Die Parameter werden in der Binärdarstellung als vorzeichenbehaftete 32-Bit-Integerzahlen hinterlegt.
- Der Parameter *Format* enthält das Schlüsselwort, welches festlegt, wie die Bilddaten zu interpretieren sind.

An diese Parameter schließt sich die Sequenz mit den Bilddaten an. Diese werden entweder als ASCII-Sequenz (Integerzahlen) oder als Bytestream (Binärvariante) hinterlegt. Die Interpretation dieser Daten hängt anschließend vom Parameter *Format* ab:

URL	Der Datenbereich enthält eine URL, die auf die Bilddatei verweist.
RGB	Die Bilddaten liegen als zweidimensionales Feld vor. Die Farbe eines jeden Bildpunkts wird durch drei Integerwerte (ASCII-Variante) oder drei Bytes (Binärdaten) beschrieben. Jedes Byte ergibt eine der Farben ***Rot, Grün*** und ***Blau***, wobei die Intensitäten zwischen 0 und 255 liegen dürfen. Der Datenbereich muß dann 3 * Col * Row Einträge aufweisen.
RGBA	Diese Variante wird wie die RGB-Variante behandelt. Einziger Unterschied ist, daß jedem Bildpunkt ein zusätzlicher ***Alpha***-Wert (Byte) zugeordnet ist. Dieser Alpha-Wert bestimmt die Sichtbarkeit des jeweiligen Bildpunkts (0 = transparent, 255 = voll deckend). Der Datenbereich muß (Row * Col * 4) Einträge aufweisen.
Mapped	Die Daten werden als zweidimensionales Feld von Bildpunkten interpretiert. Jeder Bildpunkt wird dann durch einen Integerwert (ASCII-Variante) oder ein Byte (Binärvariante) definiert. Diese Werte stellen einen Index in die vorher definierte Palette (***Color Map***) dar. Diese Darstellung erlaubt 256 Farben (Werte zwischen 0 und 255). Der Datenbereich enthält (Row * Col) Einträge.
GIF	Der Datenbereich enthält die Daten im GIF-Format.
TIFF	Der Datenbereich enthält die Daten im TIFF-Format.
JPEG	Der Datenbereich enthält die Daten im JPG-Format.

Unabhängig vom verwendeten Format werden die Bildpunkte beginnend in der linken oberen Ecke zeilenweise von links nach rechts ausgegeben. Bei der Ausgabe ist das Bild gemäß den Angaben in dem durch die Punkte *X1,Y1* und *X2,Y2* beschriebenen Rechteck zu positionieren und ggf. zu skalieren.

Draw Line

Diese Funktion veranlaßt, daß eine einfache Linie zwischen zwei Punkten gezeichnet wird. Der Befehl liegt in vier Varianten vor. Der Single-Byte-Befehl mit Operanden in der ASCII-Darstellung besitzt die Syntax:

```
L <X1>,<Y1> <X2>,<Y2>
```

und zeichnet die Linie zwischen den angegebenen zwei Punkten. Die Parameter werden dabei als Integerwerte im ASCII-Format hinterlegt. Die binären Varianten geben relative Koordinaten an. Für die Ausgabe einer einfachen Linie mit relativen Koordinaten ist folgender Befehl definiert:

```
l<X1><Y1><X2><Y2>
```

Der Buchstabe *l* entspricht dem Code 6CH. Die Parameter mit den relativen logischen Koordinaten der Anfangs- und Endpunkte werden als vorzeichenbehaftete 32-Bit-Integerzahlen gespeichert. Alternativ gibt es die Möglichkeit, die Koordinaten für die beiden Punkte als16-Bit-Integerwerte zu kodieren. Hierbei gilt folgende Syntax:

```
0CH<X1><Y1><X2><Y2>
```

Der Opcode 0CH entspricht den Zeichen Ctrl-L und signalisiert, daß vier 16-Bit-Parameter folgen. Der Befehl mit dem Opcode 8CH erlaubt, mehrere Linien zwischen den angegebenen Punkten zu ziehen. Hierbei gilt folgende Syntax:

```
8CH<Count>[<X1><Y1><X2><Y2>]
```

Der Parameter *Count* legt fest, wie viele Liniensegmente folgen (Count muß zwischen 1 und 256 liegen). Die Punkte werden als Sequenz von Start- und Endkoordinaten der Form *X1,Y1* und *X2,Y2* definiert. Die Koordinatenwerte werden relativ angegeben, und die Parameter *X,Y* liegen als 16-Bit-Integer vor. Die Linie wird mit den Attributen *Line Weight*, *Line Cap*, *Line Pattern*, *Color* und *Visibility* gezeichnet. Ist eine Serie von Liniensegmenten zu zeichnen, wird die Funktion nur benutzt, falls die Liniensegmente nicht miteinander verbunden sind. Zum Zeichnen mehrerer verbundener Liniensegmente ist die Funktion *Polyline* zu verwenden.

Draw PolyBézier Curve

Mit dieser Funktion lassen sich Bézier-Kurven zeichnen. Es kann sich dabei um eine einzelne Kurve oder eine Serie verbundener Bézier-Kurven handeln. Für den (Extended ASCII-) Datensatz gilt folgende Syntax:

```
(Bezier <Count> <Xs>,<Ys>[ <XC1>,<YC1> <XC2>,<YC2> <Xe>,<Ye>] )
```

Damit wird die Bézier-Kurve vom Startpunkt *Xs,Ys* zum Endpunkt *Xe,Ye* gezeichnet. Die Koordinaten *XC1,YC1* und *XC2,YC2* definieren die Hilfspunkte zur Ausgabe der Bézier-Kurve (Bild 37.5). Die in [ ] stehenden Parameter können sich wiederholen, um die Ausgabe mehrerer Bézier-Kurven zu ermöglichen. Die Zahl der Bézier-Kurven wird in *Count* angegeben. Alle Koordinatenangaben erfolgen als Integerzahlen im ASCII-Format und definieren absolute Koordinaten.

n. gef. H:\Galileo\Born\375

Abbildung 37.5 Stützpunkte für eine Bézier-Kurve

Alternativ läßt sich die Kurve durch einen Single-Byte-Opcode mit binären Operanden angeben. Dieser Datensatz besitzt folgendes Format:

```
b<Count>[<Ecount>]<Xs><Ys>[<XC1><YC1><XC2><YC2><Xe><Ye>]
```

Der Datensatz besitzt den Opcode *b* (Code 62H) und wird von binären Operanden gefolgt. In der binären Variante sind alle Koordinatenangaben relativ und werden als vorzeichenbehaftete 32-Bit-Integer hinterlegt.

- Der Parameter *Count* legt die Zahl der verbundenen Vierpunkt-Bézier-Kurven fest. Im Binärformat legt der Wert 0 fest, daß ein erweitertes vorzeichenloses 16-Bit-Wort *Ecount* mit der Zahl der Vierpunkt-Bézier-Kurven folgt. Der Wert von *Ecount* ist um 256 zu erhöhen, d.h. es können zwischen 256 und 65791 Kurven aneinandergefügt werden.
- Die Parameter *Xs,Ys* definieren den Startpunkt der Bézier-Kurve. *Xc1,Yc1* geben die relativen Koordinaten des ersten Kontrollpunkts der Bézier-Kurve wieder. Der zweite Kontrollpunkt wird in den Parameter *Xc2,Yc2* vereinbart. Der Endpunkt der Bézier-Kurve wird mit den Parametern *Xe,Ye* definiert. Dieser Punkt ist ggf. der Startpunkt für die nächste Bézier-Kurve.

Das Aussehen der Bézier-Kurve wird durch eine mathematische Funktion beschrieben und durch die Lage der vier Punkte beeinflußt. Um einen runden Übergang zwischen mehreren Bézier-Kurven zu erreichen, müssen die Punkte *Xc2,Yc2* und *Xe,Ye* der vorhergehenden Bézier-Kurve und der Punkt *Xc1,Yc1* der folgenden Bézier-Kurve auf einer Linie liegen. Ist der Füllmodus nicht aktiv (Standard), zeichnet die *Draw PolyBézier*-Funktion nur eine Kurve, basierend auf den Attributen *Color*, *Visibility*, *Line Weight*, *Line Cap* und *Line Pattern*. Bei aktivem *Fill Mode*-Attribut wird das Innere der Fläche zwischen der Bézier-Kurve und der Verbindungslinie der Punkte *Xs,Yx* und *Xe,Ye* mit Farbe (*Color*, *Visibility*) gefüllt.

Draw Polyline/Polygon

Diese Funktion erlaubt Ihnen, ein Polygon (eine Folge verbundener Liniensegmente) zu zeichnen. Der Datensatz wird mit einem Opcodebyte kodiert, kann aber lesbare oder binäre Operanden aufweisen. Die Variante mit lesbaren Parametern besitzt folgende Syntax:

```
P <Count>[ <X>,<Y>]
```

wobei die Werte in [ ] sich mehrfach wiederholen können. Jeder Parametersatz *X,Y* definiert einen Stützpunkt des Polygons (ASCII-Wert im Integerformat) in absoluten Koordinatenangaben. Die Zahl der Stützpunkte findet sich im Parameter *Count*. Bei Datensätzen mit binären Parametern werden zwei Varianten unterschiedenen. Beide Varianten

geben die Koordinaten der Punkte in relativen Koordinaten an, wobei jedoch eine Variante die Binärwerte mit 16 Bit kodiert. Für die erste Variante gilt folgende Syntax:

```
p<Count>[<Ecount>][<Xi><Yi>] ]
```

Der Opcode *p* (Code 70H) leitet die Sequenz ein. Der Parameter *Count* ist als Byte definiert und legt die Zahl der folgenden Stützpunkte *Xi,Yi* fest. Ist *Count* in der binären Variante auf 0 gesetzt, folgt *Ecount* als vorzeichenlose 16-Bit-Zahl. *Ecount* gibt dann die Zahl der Stützpunkte im Bereich zwischen 256 bis 65791 an (der Wert ist um 256 zu erhöhen). Die Koordinaten der Stützpunkte *Xi,Yi* werden in dieser Variante als vorzeichenbehaftete 32-Bit-Integerwerte hinterlegt. Die zweite Variante besitzt die Syntax:

```
10H<Count>[<Ecount>][<Xi><Yi>] ]
```

Hier wird der Opcode 10H benutzt, was dem Zeichen Ctrl-p entspricht. Daran schließen sich die gleichen Parameter wie beim Opcode *p* an. Der einzige Unterschied besteht darin, daß die Koordinatenangaben als 16-Bit-Integer hinterlegt werden.

Die Funktion *Draw Polyline/Polygon* zeichnet eine Kette von Liniensegmenten durch die angegebenen Stützpunkte. Ist das *Fill Mode*-Attribut freigegeben, zieht die Funktion anschließend eine gerade Linie durch Start- und Endpunkt und füllt dann das geschlossene Polygon mit der Füllfarbe. Beim Zeichnen einer Kurve werden die Attribute *Line Weight*, *Line Cap*, *Line Pattern*, *Color* und *Visibility* verwendet. Gegenüber einer Serie von einfachen *Draw Line*-Befehlen gibt es dabei folgende Unterschiede:

- Ist ein Linienmuster (*Line Pattern*) definiert, wird dieses ohne Unterbrechung in den Kurven gezeichnet.
- Für die Linienenden kann ein *Line Join*-Stil definiert werden, der bei Liniendicken größer 1 Punkt sichtbar wird.

In *Count* steht die Zahl der Stützpunkte, d.h. dieser Wert ist um 1 höher als die Zahl der Liniensegmente. Die Füllfarbe wird bei aktivem *Fill Mode*-Attribut durch die Attribute *Color* und *Visibility* bestimmt. Der *Polyline*-Datensatz eignet sich besser zum Zeichnen verbundener Liniensegmente als eine Sequenz von *Draw Line*-Aufrufen.

Möchten Sie eine gefüllte Fläche zeichnen, in deren Innenbereich eine Fläche ausgespart wird? Dann zeichnen Sie zuerst die äußere Fläche und legen Sie ein weiteres Polygon darüber, welches quasi das Loch beschreibt. Enthält das Polygon mehr als 65791 Stützpunkte, verwenden Sie mehrere *Draw Polyline/Polygon*-Opcodes (oder Sie verwenden die ASCII-Variante des Befehls). Die folgende Sequenz zeichnet ein rotes Dreieck, in dessen Fläche ein schwarzes dreieckiges Loch auftritt:

```
(Color 255,0,0,255)(Comment Farbe Rot)
F(Comment Fill Mode ein)
P 3 0,0 10,0 5,5(Comment Zeichne Dreieck)
```

```
(Color 0,0,0,255)(Comment Farbe Schwarz)
f(Comment Fill Mode aus)
P 4 0,0 10,0 5,5 0,0 (Comment Zeichne Loch)
```

In obigem Beispiel muß für das zweite Dreieck ein Polygon mit vier Stützpunkten angegeben werden (Endpunkt ist gleich Startpunkt), damit eine geschlossene Fläche entsteht (der Füllmodus ist ja ausgeschaltet).

Enthält *Count* einen Wert kleiner 3, wird das *Fill Mode*-Attribut ignoriert (eine Linie läßt sich nicht füllen). Dann sollte eine *Draw Line*-Funktion anstelle von *Draw Polyline* benutzt werden.

Draw Polymarker

Diese Funktion zeichnet ein Symbol oder eine Folge von Symbolen aus der Symbolbibliothek (Marker Glyphs). Die Marker Glyphs müssen in der Symbolbibliothek definiert worden sein. Der Befehl kann mit lesbaren Operanden oder mit binär kodierten Operanden gespeichert werden. Für die Variante mit lesbaren Operanden gilt folgende Syntax:

```
M <Count>[ <Xi>,<Yi>]
```

Count legt die Zahl der zu zeichnenden Symbole fest. Die beiden Parameter *Xi,Yi* geben die absoluten Koordinaten für das jeweilige Symbol an. Für jedes Symbol ist ein solches Koordinatenpaar erforderlich. Die Werte werden als ASCII-Integerzahlen hinterlegt. Die binäre Variante benutzt relative Koordinatenangaben für die Punkte, an denen die Symbole auszugeben sind. Die erste Variante hinterlegt die Koordinaten als vorzeichenbehaftete 32-Bit-Integerzahlen und besitzt folgende Syntax:

```
m<Count>[Ecount][ <Xi>,<Yi>]
```

Dieser Datensatz beginnt mit dem Opcode *m* (Code 6DH), an den sich das Byte *Count* mit der Zahl der auszugebenden Symbole anschließt. Ist dieser Wert 0, folgt der 16-Bit-Wert *Ecount*, der die Symbolzahl zwischen 256 und 65791 festlegt. Für jedes Symbol ist eine relative Koordinantenangabe der Form *Xi,Yi* erforderlich. Die zweite Variante benutzt ebenfalls eine binäre Darstellung für die Parameter und besitzt die Syntax:

```
8DH<Count>[Ecount][ <Xi>,<Yi>]
```

Die Parameter sind wie beim Opcode 6DH definiert, lediglich die Koordinatenangaben werden in vorzeichenbehafteten 16-Bit-Integerwerten hinterlegt.

Die *Draw Polymarker*-Funktion erlaubt die Ausgabe eines oder mehrerer Symbole an definierbaren Positionen. Welches Symbol mit welchen Optionen ausgegeben wird, legen die Attribute *Marker Glyph*, *Marker Size*, *Visibility* und *Color* fest. Benötigen Sie mehr als 65791 Symbolpunkte, sind mehrere *Draw Polymarker*-Opcodes oder die ASCII-Variante zu verwenden.

Draw Polytriangle

Diese Funktion zeichnete eine Kette aneinandergrenzender Dreiecke (siehe auch Bild 37.4). Die Funktion gleicht der Funktion zum Zeichnen eines Polygons. Aber die Stützpunkte für die Dreiecke werden in besonderer Weise angeordnet, so daß sich die Innenflächen der Dreiecke schattieren lassen. Die Funktion läßt sich über einen Single-Byte-Datensatz mit binären Parametern gemäß folgender Syntax aufrufen:

```
74H<Count>[<Ecount>][ <Xi>,<Yi>] ...)
```

In Byte *Count* steht die Zahl der Stützpunkte im Poly-Dreieck. Ist der Wert 0, wird im folgenden 16-Bit-Wort *Ecount* die Zahl der Stützpunkte angegeben (zu diesem Wert ist 256 zu addieren, um die Zahl zu erhalten). Enthält *Count* einen Wert größer 0, entfällt *Ecount*. Daran schließt sich eine Sequenz mit den Koordinaten der Stützpunkte *Xi,Yi* an. Diese Werte werden als 32-Bit-Integer kodiert. Der Opcodes für die Extended ASCII-Variante und für die Binärvariante mit 16-Bit-Koordinatenwerten sind in DWF 1.0 noch nicht definiert.

Draw Text

Dieser (Extended ASCII-) Datensatz definiert einen auszugebenden String und besitzt die folgende Syntax:

```
(Text <X1>,<Y1> <X2>,<Y2> <String>)
```

In der DWF-Version 1.0 sind die betreffenden Parameter aber noch nicht in ihrer Bedeutung festgelegt.

Draw Textured Polytriangle

Die *Draw Textured Polytriangle*-Funktion zeichnet eine Textur, die aus einer Kette von aneinanderliegenden Dreiecken besteht (siehe auch Bild 37.4). In der DWF-Version 1.0 ist die Syntax der betreffenden Datensätze noch nicht definiert. Lediglich der Opcode für den Extended ASCII-Datensatz ist mit *(Texture ...)* festgelegt. Zusätzlich wurde der Opcode 77H für die binäre Variante des Single-Byte-Opcodes vereinbart.

Embed Source File

Dieser Datensatz erlaubt es, eine Quelldatei in die DWF-Datei einzubetten. In der DWF-Version 1.0 ist dieser Datensatztyp noch nicht komplett definiert. Bisher existieren zwei Extended ASCII-Datensatztypen, um die Datei anzugeben. Mit:

```
(EmbedRef (<Typ>/<Subtyp>;[Optionen]) (<Description>) (<Filename>) (<URL>))
```

wird eine Referenz auf die Quelldatei hinterlegt (z.B. EmbedRef (image/autocad-dxf;) () () (http://www.autodesk.com/haus.dxf)). Alternativ läßt sich die Quelldatei als Datensequenz einbetten:

```
(EmbedFile (<Typ>/<Subtyp>;[Optionen]) (<Description>) (<Filename>) (<Daten>))
```

Die Parameter dieser beiden Recordtypen besitzen dabei folgende Bedeutung:

- In *Typ* wird der MIME-Code (Multipurpose Internet Mail Extension) für die eingebetteten Daten oder die referenzierte Datei hinterlegt. Das Feld *Subtyp* legt den MIME-Subtyp für die betreffenden Daten fest. In *Optionen* lassen sich MIME-Optionen für die betreffenden Daten hinterlegen.
- Das Feld *Desc* dient zur Aufnahme einer allgemeinen Beschreibung der eingebetteten Daten (dieses Argument ist normalerweise zu ignorieren).
- Das Feld *URL* enthält den URL-Verweis auf die Quelldatei.
- Wird die Datei in der DWF-Datei eingebettet, enthält das Feld *Filename* den Dateinamen der Quelldatei, während die Daten als Sequenz im Feld *Data* stehen.

Dieser Datensatz soll es dem Benutzer erlauben, die Original-Quelldatei nach der Anzeige im Browser zu editieren. In der Version 1.0 der DWF-Spezifikation ist der Datensatz jedoch noch nicht endgültig definiert.

Set Clip

Diese Funktion legt einen rechteckigen *Clipping*-Bereich fest. Anschließend werden nur noch Objekte innerhalb des Bereichs gezeichnet. Der zugehörige Extended ASCII-Datensatz besitzt folgende Syntax:

```
(Clip [<X1>,<Y1>] <X2>,<Y2>[ ])
```

Die in [ ] stehenden beiden Koordinatenpaare *X1,Y1* und *X2,Y2* legen die linke untere und die rechte obere Ecke des Clipping-Bereichs fest. Beide Koordinatenangaben sind optional. Mit dem Befehl *(Clip)* wird dann die Clipping-Funktion wieder abgeschaltet. Durch Setzen mehrerer Clip-Befehle lassen sich auch Regionen definieren, die nicht rechteckig sind. Standardmäßig ist die Clipping-Funktion jedoch abgeschaltet.

Set Color

Diese Funktion ändert die Farbe, die bei der Ausgabe geometrischer Primitive von der Zeichenfunktion zu verwenden ist. Der (Extended ASCII-) Datensatztyp erlaubt es, die Farbe als RGBA gemäß folgender Syntax festzulegen:

```
(Color <R>,<G>,<B>,<A> )
```

Die Buchstaben R, G, B und A stehen dabei für die Farbanteile Rot, Grün, Blau und den Alpha-Kanal. Alternativ läßt sich die Farbe über einen Single-Byte-Opcode mit lesbaren Parametern in der Form:

```
C<Index>
```

darstellen. Nach dem Opcode 43H folgt eine Integerzahl *Index*, die den Index in die Farbpalette definiert. Alternativ gibt es die Möglichkeit, den Datensatz mit einem Opcodebyte und binären Operanden zu speichern. Hierbei sind zwei Varianten definiert:

```
63H<Indexbyte>
```

oder

```
03H<R><G><B><A>
```

In beiden Varianten werden die Parameter als Bytes kodiert. Die Variante mit dem Opcode 63H (Zeichen *c*) enthält ein Byte für den Index in die Farbtabelle. Die Variante mit dem Opcode 03H (Zeichen Ctrl-c) definiert dagegen einen Farbwert über die Anteile *Rot, Grün, Blau* und *Alpha*. Der Wertebereich für einen Farbanteil muß zwischen 0 und 255 liegen. Die Angabe von Indexwerten ist nur zulässig, falls eine Palette (Color Map) vereinbart wurde. Bei Verwendung eines Indexwerts können durchaus mehrere Farbpaletten definiert werden. Es gilt immer die zuletzt gelesene Farbpalette. Bei fehlerhaften Indexwerten kann der DWF-Reader festlegen, welche Farbe zu verwenden ist. Standardmäßig wird die Farbe Weiß (voll Opact) eingestellt, was dem Farbbefehl (Color 255,255,255,255) entspricht.

Set Color Map

Dieser Datensatz enthält die Daten zur Definition einer Farbtabelle. Diese läßt sich entweder mit einem Extended ASCII-Datensatz oder mit einem Extended Binary-Datensatz speichern. Für den Extended ASCII-Datensatz gilt folgende Syntax:

```
(ColorMap <Count> [ <Ri>,<Gi>,<Bi>,<Ai>] )
```

Der Parameter *Count* definiert die Zahl der Einträge der Farbpalette. Für jeden dieser Einträge ist ein Farbwert über die Parameter *Ri, Gi, Bi* und *Ai* festzulegen (entspricht den Farbanteilen *Rot, Grün, Blau* und *Alpha*-Kanal). Dies bedeutet, die in [ ] stehenden Parameter werden ggf. mehrfach wiederholt. Die Parameter können durch White-Space-Zeichen getrennt werden. Für die binäre Variante des Datensatzes gilt die Syntax:

```
{... 0001H<Count>[<Ri><Gi><Bi><Ai>]}
```

Alle Parameter werden als Bytewerte hinterlegt. Die in [ ] stehenden Parameter werden ggf. mehrfach wiederholt. Es gilt die Belegung wie bei der ASCII-Variante. Enthält der Parameter *Count* den Wert 0, bedeutet dies, daß die Palette 256 aufweist. Daher muß der Datensatz mindestens eine Farbpalette mit einem Eintrag definieren. Die Einträge der Farbpalette werden von den Befehlen *Draw Image* oder *Set Color* benutzt. Standardmäßig sollte pro DWF-Datei nur einmal der *Set Color Map*-Opcode benutzt werden (möglichst vor einem *Set Color*-Datensatz). Sie können aber durchaus mehrere Datensätze mit Pa-

letten in der DWF-Datei speichern. Die Standardvorgabe für eine Farbpalette mit 256 Farben ist folgendermaßen definiert:

```
(ColorMap 256 0,0,0,255 255,0,0,255 255,255,0,255 0,255,0,255 0,255,255,255
0,0,255,255 255,0,255,255 0,0,0,255 128,128,128,255 220,220,220,255
255,0,0,255 255,127,127,255 221,0,0,255 221,110,110,255 184,0,0,255
184,92,92,255 149,0,0,255 149,74,74,255 114,0,0,255 114,57,57,255
255,63,0,255 255,159,127,255 221,55,0,255 221,138,110,255 184,46,0,255
184,115,92,255 149,37,0,255 149,93,74,255 114,28,0,255 114,71,57,255
255,127,0,255 255,191,127,255 221,110,0,255 221,165,110,255 184,92,0,255
184,138,92,255 149,74,0,255 149,112,74,255 114,57,0,255 114,86,57,255
255,191,0,255 255,223,127,255 221,165,0,255 221,193,110,255 184,138,0,255
184,161,92,255 149,112,0,255 149,131,74,255 114,86,0,255 114,100,57,255
255,255,0,255 255,255,127,255 221,221,0,255 221,221,110,255 184,184,0,255
184,184,92,255 149,149,0,255 149,149,74,255 114,114,0,255 114,114,57,255
191,255,0,255 223,255,127,255 165,221,0,255 193,221,110,255 138,184,0,255
161,184,92,255 112,149,0,255 131,149,74,255 86,114,0,255 100,114,57,255
127,255,0,255 191,255,127,255 110,221,0,255 165,221,110,255 92,184,0,255
138,184,92,255 74,149,0,255 112,149,74,255 57,114,0,255 86,114,57,255
63,255,0,255 159,255,127,255 55,221,0,255 138,221,110,255 46,184,0,255
115,184,92,255 37,149,0,255 93,149,74,255 28,114,0,255 71,114,57,255
0,255,0,255 127,255,127,255 0,221,0,255 110,221,110,255 0,184,0,255
92,184,92,255 0,149,0,255 74,149,74,255 0,114,0,255 57,114,57,255
0,255,63,255 127,255,159,255 0,221,55,255 110,221,138,255 0,184,46,255
92,184,115,255 0,149,37,255 74,149,93,255 0,114,28,255 57,114,71,255
0,255,127,255 127,255,191,255 0,221,110,255 110,221,165,255 0,184,92,255
92,184,138,255 0,149,74,255 74,149,112,255 0,114,57,255 57,114,86,255
0,255,191,255 127,255,223,255 0,221,165,255 110,221,193,255 0,184,138,255
92,184,161,255 0,149,112,255 74,149,131,255 0,114,86,255 57,114,100,255
0,255,255,255 127,255,255,255 0,221,221,255 110,221,221,255 0,184,184,255
92,184,184,255 0,149,149,255 74,149,149,255 0,114,114,255 57,114,114,255
0,191,255,255 127,223,255,255 0,165,221,255 110,193,221,255 0,138,184,255
92,161,184,255 0,112,149,255 74,131,149,255 0,86,114,255 57,100,114,255
0,127,255,255 127,191,255,255 0,110,221,255 110,165,221,255 0,92,184,255
92,138,184,255 0,74,149,255 74,112,149,255 0,57,114,255 57,86,114,255
0,63,255,255 127,159,255,255 0,55,221,255 110,138,221,255 0,46,184,255
92,115,184,255 0,37,149,255 74,93,149,255 0,28,114,255 57,71,114,255
0,0,255,255 127,127,255,255 0,0,221,255 110,110,221,255 0,0,184,255
92,92,184,255 0,0,149,255 74,74,149,255 0,0,114,255 57,57,114,255
63,0,255,255 159,127,255,255 55,0,221,255 138,110,221,255 46,0,184,255
115,92,184,255 37,0,149,255 93,74,149,255 28,0,114,255 71,57,114,255
```

```
127,0,255,255 191,127,255,255 110,0,221,255 165,110,221,255 92,0,184,255
138,92,184,255 74,0,149,255 112,74,149,255 57,0,114,255 86,57,114,255
191,0,255,255 223,127,255,255 165,0,221,255 193,110,221,255 138,0,184,255
161,92,184,255 112,0,149,255 131,74,149,255 86,0,114,255 100,57,114,255
255,0,255,255 255,127,255,255 221,0,221,255 221,110,221,255 184,0,184,255
184,92,184,255 149,0,149,255 149,74,149,255 114,0,114,255 114,57,114,255
255,0,191,255 255,127,223,255 221,0,165,255 221,110,193,255 184,0,138,255
184,92,161,255 149,0,112,255 149,74,131,255 114,0,86,255 114,57,100,255
255,0,127,255 255,127,191,255 221,0,110,255 221,110,165,255 184,0,92,255
184,92,138,255 149,0,74,255 149,74,112,255 114,0,57,255 114,57,86,255
255,0,63,255 255,127,159,255 221,0,55,255 221,110,138,255 184,0,46,255
184,92,115,255 149,0,37,255 149,74,93,255 114,0,28,255 114,57,71,255
84,84,84,255 118,118,118,255 152,152,152,255 186,186,186,255 220,220,220,255
255,255,255,255)
```

Set Current Point

Dieser Datensatz wird mit einem Opcodebyte und einem binären Operand gespeichert und legt den Wert der Variablen *CurrentPoint* fest. Hierbei gilt folgende Syntax:

```
4FH<X><Y>
```

Die beiden Parameter *X,Y* werden als vorzeichenlose 32-Bit-Integerwerte hinterlegt und geben die absoluten Koordinaten des *CurrentPoint* an. Der *CurrentPoint* wird von allen Befehlen benötigt, die mit relativen Koordinaten arbeiten. Standardmäßig wird der Wert von *CurrentPoint* mit 0,0 definiert.

Set Fill Mode

Dieser Datensatz wird mit einem Opcodebyte und ohne Parameter gespeichert. Es gibt zwei Varianten, die das Attribut für den Füllmodus ein- oder ausschalten: Mit dem Zeichen *F* (Opcode 46H) wird der Füllmodus eingeschaltet, während das Zeichen *f* (Opcode 66H) den Füllmodus wieder abschaltet. Der Füllmodus wird von verschiedenen Zeichenfunktionen wie *Draw Polyline/Polygon*, *Draw Circle* und *Draw Ellipse*, die Flächen definieren, verwendet. Bei aktivem Füllmodus wird die Fläche in der aktuellen Farbe gezeichnet. Bei deaktiviertem Füllmodus zeichnen die Funktionen nur die Umrißlinie. Ein DWF-Reader sollte dabei immer die Stützpunkte der Umrißlinie in der Reihenfolge gerade/ungerade (1, 2, 3, 4 etc.) verbinden. Standardmäßig ist der Füllmodus abgeschaltet.

Set Layer

Dieser Datensatz wird verwendet, um eine Gruppe von Opcodes zu trennen. Diese Gruppe von Opcodes läßt sich zu einem späteren Zeitpunkt bearbeiten. Der Datensatz ist als Extended ASCII mit folgender Syntax definiert:

```
(Layer <LayerNummer> [<LayerName>])
```

In *LayerNummer* wird die Nummer für die jeweilige Gruppe (Ebene) vereinbart, während *LayerName* eine optionale Bezeichnung für die Ebene aufnimmt. Mit diesem Opcode können verschiedene Ebenen innerhalb einer Zeichnung vereinbart werden, die sich ggf. ein- oder ausblenden lassen (z. B. elektrische Leitungen in einem Gebäude). Das nachfolgende Beispiel zeigt die Anwendung:

```
L 0,0 15,15(Comment Default ist Layer 0 aktiv)
(Layer 1 Electrical)(Comment nächster Layer)
L 10,10 35,35(Comment zeichne Linie in Layer 1)
L 0,10 30,30(Comment noch eine Linie in Layer 1)
(Layer 0)(Comment Wähle Layer 0)
L 0,20 20,20(Comment Linie in Layer 0)
```

Ähnlich wie bei den anderen Attributen kann immer nur ein Layer aktiv sein. Der Layer 0 ist immer vorhanden und sichtbar. Sie können bis zu 32767 Layer verwenden. Eine Zeichenfunktion kann zu einer Zeit nur in einem Layer (dem aktuellen Layer) zeichnen. Die Layer sind standardmäßig vorhanden und auf den Status *sichtbar* gesetzt. Sie können einen Layer aber über *Set Layer State* abschalten (unsichtbar setzen). Die Layer beeinflussen nicht die Reihenfolge, in der die Zeichenoperationen ausgegeben werden. Sie können einem geometrischen Objekt eine URL zuweisen, die in einem unsichtbaren Layer liegt (dann kann der Benutzer nicht auf den URL-Verweis zugreifen).

Set Line Cap

Diese Funktion legt fest, wie das Ende einer Linie auszugeben ist. Dies wirkt sich aber nur bei Linien aus, die mehr als 1 Pixel dick sind. Der Datensatz besitzt folgende Syntax:

```
(LineCap <Cap>)
```

Der Parameter *Cap* enthält ein Schlüsselwort, welches die Form des Linienendes festlegt. Diese Form gilt für die Linienenden (Line Caps) und für sich in einem spitzen Winkel treffende Linienenden (Line Joins). Gültige Werte für *Cap* sind dabei:

butt	Die Linie wird gerade abgeschlossen. Bei sich treffenden Linienenden (Line Join) wird eine Spitze gezeichnet (Miter Join).
square	Die Linie wird mit einem Rechteck abgeschlossen. Dadurch verlängert sich die Linie um die Hälfte der Liniendicke. Bei sich treffenden Linienenden (Line Join) wird die Spitze abgeschnitten (Bevel Join).
round	Das Linienende wird mit einem Halbkreis abgerundet. Der Radius entspricht der Hälfte der Liniendicke. Bei sich treffenden Linienenden wird eine runde Spitze (Round Join) gezeichnet.

Wie die Linienenden genau dargestellt werden, ist systemabhängig. Standardmäßig wird die Vorgabe *(LineCap butt)* verwendet.

Set Line Join

Diese Funktion definiert, wie die Verbindung zweier sich treffender Linienenden zu zeichnen ist. Hierbei gilt folgende Syntax:

```
(LineJoin <Join>)
```

Der Parameter *Join* legt die Form der Verbindung fest und kann folgende Schlüsselworte enthalten:

Miter	Die Linien laufen in einer Spitze zusammen.
Bevel	Die Spitze wird abgeflacht dargestellt.
Round	Die Spitze wird abgerundet.

Diese Option wirkt sich jedoch nur aus, falls Linien mit einer Dicke von mehr als einem Punkt verwendet werden. Standardmäßig wird *(LineJoin miter)* verwendet, d.h. zwei an einem Punkt endende Linien besitzen eine Spitze bzw. Ecke.

Set Line Pattern

Dieser Datensatz definiert ein Linienmuster. Der Datensatz besitzt folgende Syntax:

```
(LinePattern <Pattern>)
```

Der Parameter *Pattern* enthält dabei einen ASCII-Textstring mit dem Linienmuster:

----	durchgezogene Linie
- -	gestrichelte Linie
....	gepunktete Linie
-.-.	strichpunktierte Linie
-..	Strich-Punkt-Punkt-Linie
-...	Linienmuster mit einem Strich und drei Punkten
-- --	lange gestrichelte Linie
center	Linienmuster mit abwechselnden langen und kurzen Strichen
phantom	Linienmuster mit alternierenden langen und sehr kurzen Strichen

Die Art, wie das Linienmuster anschließend ausgegeben wird, hängt von der Anwendung ab. Allgemein wird das Muster auf einzelne Bildpunkte des Screens abgebildet. Benötigen Sie sehr genaue Linienmuster, müssen Sie die Funktion *Draw Line* verwenden. Standardmäßig ist die Einstellung auf *(LinePattern ----)* gesetzt.

Set Line Weight

Dieser Datensatz definiert die Linienstärke und besitzt folgende Syntax:

```
(LineWeight <Dicke>)
```

Der Parameter *Dicke* definiert die Dicke aller nachfolgend gezeichneten Linien als Vielfaches einer logischen Koordinateneinheit. Das Attribut wird von verschiedenen Funktionen zum Zeichnen von Linien, Kreisen etc. benutzt. Häufig ist die logische Liniendicke geringer als ein Bildpunkt auf dem Schirm. Damit wird die Linie mit einer Dicke von mindestens einem Pixel gezeichnet. Die Dicke *0* ist daher immer mit einem Pixel zu zeichnen (unabhängig vom Zoomfaktor). Standardmäßig wird die Liniendicke auf *(LineWeight 0)* gesetzt.

Set Marker Glyph

Dieser Datensatz wählt ein Symbol (Marker Glyph) aus der Symboltabelle aus. In der DWF-Version 1.0 ist ein Single-Byte-Opcode mit einem lesbaren Parameter in der Form:

```
G <Index>
```

definiert. Der ASCII-Parameter *Index* gibt dann das Symbol in der Tabelle an, welches zukünftig als Marker Glyph zu verwenden ist. Alternativ können Sie die Form:

```
87H<Index>
```

verwenden. Dann wird der Parameter als 16-Bit-Wort kodiert. Wichtig ist, daß das betreffende Symbol in der Symboltabelle mit *Define Marker Glyph* definiert wurde. Standardmäßig ist ein Stern als Symbol (Marker Glyph) definiert. Dieses Symbol läßt sich mit G0 wählen.

Set Marker Size

Dieser Datensatz legt die Größe für das auszugebende Symbol (Marker) fest. Die Variante mit einem Opcodebyte und lesbaren Parametern besitzt folgende Syntax:

```
S <Size>
```

Hinter dem Zeichen *S* (Opcode 53H) folgen optionale White-Space-Zeichen und dann der Parameter *Size* mit der Größe in logischen Koordinateneinheiten. Die Variante mit binären Parametern besitzt die folgende Syntax:

```
s<Size>
```

Der Parameter *Size* wird dabei als 16-Bit-Wort festgelegt. Die *Set Marker Size*-Operation bewirkt, daß die Symbolgröße an den bei der *Set Marker Glyph* definierten Parameter *Unit* angepaßt wird. Standardmäßig ist die Markergröße auf 0 (logische Einheiten im Ko-

ordinatensystem) gesetzt, was einer Größe von 1 Pixel entspricht. Dies läßt sich durch den Befehl S0 darstellen.

Set Projection

Dieser Datensatz teilt der lesenden DWF-Anwendung mit, welcher Bereich des logischen Koordinatenraums anzuzeigen ist. Hierbei gilt folgende Syntax:

```
(Projection <Proj>)
```

Der Parameter *Proj* legt dabei die Projektion für den anzuzeigenden Quadranten fest:

- *normal:* Der angezeigte Ausschnitt wird symmetrisch skaliert, damit er in das Anzeigefenster paßt. Verhindert der *Apect Ratio*, daß das Bild in den Ausschnitt paßt, wird es entsprechend skaliert.
- *stretch:* Das Bild wird so gestreckt, daß es genau in das Anzeigefenster paßt. Die Strekkung bezieht sich auf beide Achsen, wodurch das Bild unter Umständen verzerrt wird.
- *chop:* Besitzt die gleiche Bedeutung wie *normal*, aber die Bereiche außerhalb des logischen Bildbereichs werden nicht gezeichnet. Dadurch bleiben in der Regel auf zwei Seiten des Bildfensters freie Bereiche, die keine Zeichenelemente enthalten.

Standardmäßig wird die Projektion auf *normal* gesetzt. In der DWF-Spezifikation 1.0 sind die Eigenschaften dieses Recordtyps noch nicht endgültig definiert.

Set URL Link

Dieser Datensatz verbindet eine geometrische Primitive mit einer URL zu einem anderen Dokument im WWW. Es gilt folgender Satzaufbau:

```
(URL <URL-Text>)
```

Als Parameter wird die URL-Adresse des Dokuments angegeben. Die URL bezieht sich auf eine in der DWF-Datei nachfolgende graphische Primitive (z. B. Linie, Marker etc.). Damit kann der Benutzer ggf. das Zeichenobjekt anklicken und zu einem Dokument im WWW verzweigen. Die Primitiven, auf die sich die URL bezieht, sind durch ein Paar von *Set URL Link*-Opcodes einzufassen:

```
(URL http://www.autodesk.com)(Comment setze URL)
L 10,10 35,35(Comment URL bezieht sich auf diese Linien)
L 0,10 30,30
(URL)(Comment Hier wird die URL wieder aufgehoben)
L 0,20 20,20(Comment Linie ist nicht mit URL verbunden)
```

Es wird immer die zuletzt definierte URL auf die folgenden Primitiven angewandt (jeder Primitive kann nur eine URL zugewiesen werden). Daher können Sie innerhalb einer

Gruppe mehrere URLs definieren und zum Schluß mit (URL) die Gruppe abschließen. Standardmäßig ist keine URL definiert, was (URL) entspricht.

Set Visibility

Dieser Datensatz bestimmt, ob die folgenden geometrischen Primitiven zu zeichnen sind. Es gibt zwei Opcodes zum Ein-/Ausschalten der Anzeige. Mit *V* (Opcode 56H) wird die Anzeige der nachfolgenden Zeichenoperationen eingeschaltet. Der Buchstabe *v* (Opcode 76H) schaltet dagegen die Sichtbarkeit der graphischen Primitiven aus.

Mit diesem Befehl können Sie zum Beispiel Primitiven unsichtbar machen, die mit einer URL verbunden sind. Alternativ können Sie den *Set Layer*-Opcode benutzen, um Primitiven zu Gruppen zusammenzufassen. Standardmäßig ist die Sichtbarkeit eingeschaltet (entspricht dem Opcode *V*).

Anmerkung: Damit möchte ich die Beschreibung des DWF-Formats abschließen. Die aktuelle Spezifikation in der Version 1.0 weist noch einige Lücken auf, und die Implementierung der WHIP!-Bibliothek umfaßt nur einen Teil der in der Version 1.0 definierten Opcodes. Weitere Informationen über das DWF-Format lassen sich über die Autodesk-Homepage *http://www.Autodesk.com* abrufen.

41 Das Amiga Animation-Format (ANI)

Auf dem Amiga wird ein erweitertes IFF-Format zur Darstellung von Animationen verwendet. Die Datei wird in einen Header und mehrere CHUNKs unterteilt (Bild 41.1).

In ANI-Dateien werden die Daten im Motorola-Format gespeichert.

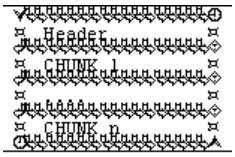


Abbildung 41.1 Struktur eines Amiga (ANI)-File

Der ANI-Header

Eine ANI-Datei enthält einen Header, der sich an die IFF-Konventionen anlehnt (Tabelle 41.1). Eine Beschreibung des IFF-Formats findet sich in Kapitel 55.

Offset	Bytes	Bemerkungen
00H	4	Signatur (46 4F 52 4D)
04H	4	Dateilänge in Byte
08H	4	IFF-Typ 'ANIM'

Tabelle 41.1 Struktur eines ANI-Headers

Das erste Feld enthält die Signatur für IFF-Dateien ('FORM'). Daran schließt sich ab Offset 04H die Dateilänge in Byte an. Die letzten vier Byte enthalten die Kennung für den Dateityp. Hier wird bei ANI-Dateien die Signatur 'ANIM' hinterlegt.

Die ANI-CHUNKs

Hinter dem Header schließen sich die CHUNKs der ANI-Datei an. Hierbei handelt es sich um gültige IFF-CHUNKs, die bereits im Abschnitt über das IFF-Format vorgestellt wurden.

Signatur	CHUNK-Typ
BMHD	Bitmap Header
CMAP	Color Map
BODY	BODY Datenblock
CAMG	Graphik Modus

Tabelle 41.2 IFF-CHUNKs in einer ANI-Datei

Über diese bereits beschriebenen CHUNK-Typen hinaus werden drei neue Typen eingeführt.

Der CPAN-CHUNK

Dieser CHUNK definiert die Länge der Animationssequenz und besitzt folgende Struktur:

Offset	Bytes	Bemerkungen
00H	4	Signatur (43 50 41 4E)
04H	4	Blocklänge in Byte
08H	2	Versionsnummer
0AH	2	Zahl der Bilder
0CH	4	Reserviert

Tabelle 41.3 Struktur eines CPAN-CHUNK

Die ersten vier Byte enthalten die Signatur 'CPAN'. Daran schließt sich die Blocklänge in Byte an. Ab Offset 08H folgt ein Wort, in dem das bearbeitende Programm seine Versionsnummer ablegt. Dieses Wort sollte von Fremdsoftware ignoriert werden. Das Wort ab Offset 0AH gibt die Zahl der Bilder (Frames) in der folgenden Sequenz an. Die letzten vier Byte sind reserviert.

Der ANHD-CHUNK

Dieser CHUNK bildet den Header der Animationssequenz und besitzt folgende Struktur:

Offset	Bytes	Bemerkungen
00H	4	Signatur (41 4E 48 44)
04H	4	Blocklänge in Byte
08H	1	Kompression
09H	1	Maske
0AH	2	Breite View Area
0CH	2	Höhe View Area
0EH	2	X-Position View Area
10H	2	Y-Position View Area
12H	4	Delay Time 1
16H	4	Delay Time 2
1AH	1	Frame Count
1BH	1	Füllbyte (0)
1CH	4	Kompressions-Flag

Tabelle 41.4 Struktur eines ANHD-CHUNK

Offset	Bytes	Bemerkungen
20H	16	Reserviert (0)

Tabelle 41.4 Struktur eines ANHD-CHUNK

Dieser CHUNK wird teilweise an Stelle des 'BMHD'-CHUNK verwendet. Als erstes findet sich die Signatur 'ANHD', gefolgt von der Blocklänge des CHUNK.

Das Byte ab Offset 08H spezifiziert die Komprimierungsmethode, wobei Werte im Intervall 1..6, 74 auftreten können. Das DWORD ab Offset 1CH enthält vermutlich Flags für die Komprimierungsmethoden 4 und 5. Die Bedeutung der Komprimierungsmethoden sind zur Zeit aber ungeklärt. Das Byte ab Offset 09H enthält eine Maske für die Bits der Bildebenen mit Änderungen, deren Bedeutung ungeklärt ist.

Die beiden Worte ab Offset 0AH definieren einen Bildausschnitt (View Area) als Breite * Höhe. Die Lage dieses Bildausschnitts wird in den beiden Worten ab Offset 0EH definiert.

Die Einzelbilder werden mit einer gewissen Verzögerungszeit ausgegeben. Das DWORD ab Offset 12H definiert die Verzögerung relativ zum 1. Frame in 1/50 Sekunden. Das DWORD ab Offset 16H definiert die Verzögerung zum vorhergehenden Frame. Ab Offset 1AH definiert ein Byte, für wie viele vorhergehende Bilder eine Änderung gilt. Mit dieser Technik läßt sich die Animationsgeschwindigkeit variieren.

Der DLTA-CHUNK

Das erste Bild einer Animation wird meist als komprimierte Bitmap mit einer Lauflängenkomprimierung abgelegt. Ist das erste Byte kleiner 80H werden n+1 folgende Bytes unkomprimiert als Bilddaten übernommen. Bei negativen Werten (> 80H) wird das Folgebyte -n+1 mal wiederholt. Bei einem Wert von 80H findet keine Operation statt.

Der DLTA-CHUNK bildet dann die Delta Frames der Animationssequenz ab. Es gilt folgende Struktur:

Offset	Bytes	Bemerkungen
00H	4	Signatur (44 4C 54 41)
04H	4	Blocklänge in Byte
08H	n	Graphik Datenbereich

Tabelle 41.5 Struktur eines DLTA-CHUNK

Dieser CHUNK wird mit der Signatur 'DLTA' eingeleitet. Daran schließt sich ein DWORD mit der Blocklänge an. Die folgenden Bilddaten werden nach verschiedenen Verfahren komprimiert. Der CHUNK scheint zur Zeit aber als privat deklariert zu sein. Informationen über den Aufbau des Datenbereichs liegen zur Zeit nicht vor.

48 Das Intel Digital Video-Format (DVI)

Mit DVI hat die Firma Intel eine Hardwarelösung zur Darstellung digitaler Videos auf dem PC geschaffen. Zusätzlich zur Hardware werden Softwaretreiber geliefert, die Dateiem im DVI-Format lesen können.

DVI-Dateien existieren in unterschiedlichen Varianten. Bilder ohne Audiodaten werden in Dateien mit unterschiedlichen Erweiterungen gespeichert:

Extension	Bemerkungen
.IMR	Red channel video data (8 Bit)
.IMG	Green channel video data (8 Bit)
.IMB	Blue channel video data (8 Bit)
.IMY	Y luminance channel video data (8 Bit)
.IMI	I color channel video data (8 Bit)
.IMQ	Q color channel video data (8 Bit)
.IMA	Alpha channel video data (8 Bit)
.IMM	Monochrome or gray scale video data (8 Bit)
.IMC	Color map video data (8 Bit)
.CMY	Compressed Y luminace channel video data (8 Bit)
.CMI	Compressed I color channel video data (8 Bit)
.CMQ	Compressed Q color channel video data (8 Bit)
.I8	Device dependant data (8 Bit)
.I16	Device dependant (16 Bit)
.C16	Compressed device dependant (16 Bit)

Tabelle 48.1 Intel Formate für DVI-HardwareIntel Formate für DVI-Hardware

Unkomprimierte Dateien mit 8-Bit-Daten werden je nach verwendetem Farbsystem in Einzeldateien gespeichert. Für das RGB-System gelten die Erweiterungen .IMR, .IMG, .IMB für die Daten des jeweiligen Farbkanals. Die Buchstaben R, G, B stehen dabei für rot, grün und blau und geben den Farbkanal an. Der erste Buchstabe I identifiziert unkomprimierte Daten. Bei komprimierten Daten wird der Buchstabe C vorangestellt.

Das AVSS-Format

Für Videosequenzen mit Audiodaten existiert das AVSS-Format (nachfolgend als DVI-Format bezeichnet). Dieses lehnt sich an die in einem der vorhergehenden Kapitel beschriebene AVI-Struktur an. Eine AVSS-Datei besitzt einen DVI-Header, gefolgt von weiteren Strukturen (Bild 48.1).

```
DVI-Header
 AVL-Header
   Stream Header
     Audio Substream Header
     Video Substream Header
   Frames
```

Abbildung 48.1 Struktur eines DVI-File

Neben dem DVI-Header schließt sich ein AVL-Header mit der Beschreibung der Video- und Audio-Streams an. Die Frames werden im Anschluß gespeichert.

Der DVI-Header

Eine DVI-Datei beginnt immer mit einem 12-Byte-Header mit folgender Struktur.

Offset	Bytes	Bemerkungen
00H	4	File ID 'VDVI'
04H	2	Header-Größe
06H	2	Version
08H	4	Zeiger auf Anmerkung (annotation)

Tabelle 48.2 Struktur eines DVI-Heder

Die ersten vier Byte enthalten die Signatur ('VDVI') für einen gültigen DVI-File mit Audio- und Videodaten. Fehlen die Audiodaten, ist die Signatur 'VIM ' einzutragen. Daran schließt sich ein Wort mit der Headerlänge in Byte an (0CH). In einigen älteren Versionen wird dieser Wert auf 1 gesetzt und sollte dann ignoriert werden.

Die Versionsnummer des Headers wird zur Zeit auf 1 gesetzt. Ab Offset 08H findet sich ein Zeiger auf einen Datenbereich, in dem Anmerkungen gespeichert sind. Der Datenbereich wird in der Regel am Dateiende angefügt. Ein Eintrag von 0 im Feld signalisiert, daß keine Anmerkungen existieren.

Der AVL-Header

An den DVI-Header schließt sich der AVL-Header mit einer Länge von 120 Byte an. Dieser leitet die Strukturbeschreibung der Streams ein und besitzt folgendes Format.

Offset	Bytes	Bemerkungen
00H	4	Header ID 'AVSS'
04H	2	Header-Größe
06H	2	Header-Version

Tabelle 48.3 Struktur des AVL-Headers

Offset	Bytes	Bemerkungen
08H	2	Zahl der Stream Groups
0AH	2	Stream Group-Größe
0CH	4	Lage 1. Stream Group
10H	2	Stream Group-Version
12H	2	Stream Header-Größe
14H	2	Stream Header-Version
16H	2	Zahl der Stream Header
18H	4	Offset Stream Structure Array
1CH	4	Header Pool Offset
20H	4	Labelzahl in der Datei
24H	4	Offset 1. Label
28H	2	Label-Größe
2AH	2	Label-Format (Version)
2CH	4	Offset Video-Sequenz-Header
30H	2	Größe Video-Sequenz-Header
32H	2	Frame Header-Version
34H	4	Zahl der Frame Header
38H	4	Größe Frame Header+Daten
3CH	4	Offset 1. Frame
42H	4	Offset letztes Frame Byte +1
46H	2	Größe Frame-Header
48H	2	Größe Frame Directory
4AH	4	Offset Frame Directory
4EH	2	Frame Directory Version
50H	2	Frames pro Sekunde
52H	4	Update Flag
56H	4	Unbenutzt (Free Block Offset)
5AH	32	Füllbytes

Tabelle 48.3 Struktur des AVL-Headers

Das erste Feld identifiziert den Header als AVL-File (AVSS). Das Feld ab Offset 04H definiert die Länge des Headers (120). Die Versionsnummer des Headers (Offset 06H) definiert die Struktur. Die vorliegende Struktur wird mit der Signatur 03H identifiziert.

Eine DVI-Datei kann in mehrere *Stream*-Gruppen aufgeteilt werden. Das Feld ab Offset 08H definiert die Anzahl der Stream Groups. Daran schließt sich die Angabe über die Länge einer solchen Stream Gruppe an. Der nächste Eintrag definiert einen Zeiger auf

den ersten Eintrag der Gruppe. Das Format dieser Gruppe wird durch die Versionsnummer festgelegt. Ist keine *Stream*-Gruppe vorhanden, werden diese Felder zu 0 gesetzt.

Ab Offset 12H folgen vier Felder mit der Beschreibung der *Stream*-Header. Hierbei handelt es sich um ein Array, welches in der DVI-Datei abgespeichert wird. Das erste Feld definiert die Größe jeder Stream-Struktur (44 Byte). Das nächste Feld gibt eine Versionsnummer für die Struktur an (aktuell 3). Der dritte Eintrag enthält die Zahl der Einträge im Array. Das letzte Feld definiert den Offset auf den ersten Eintrag im Array.

Der Eintrag ab Offset 1CH zeigt auf den ersten Sub-Stream Header. Steht hier der Wert 0, existieren keine Sub-Streams.

Enthält eine DVI-Datei Marken (Labels), definieren die Felder ab 20H die Zahl der Labels, den Offset zum ersten Label, die Größe eines Labels und die Versionsnummer der Labelstruktur. Ein Wert von 00H im Feld mit der Zahl der Labels signalisiert, daß keine Labels in der Datei auftreten.

Die beiden Felder ab Offset 2CH geben den Offset und die Länge des (optionalen) Videosequenz-Headers an.

Das Datenformat der Frame-Daten wird über die Versionsangabe im Feld ab Offset 32H definiert. Daran schließen sich Felder mit Informationen über die Zahl der Frames, die Framelänge (inclusive Header), den Offset auf den ersten Frame-Header an. Das folgende Feld definiert die Größe aller Frame-Header. Die Frame-Directory besitzt eine Größe von 4. Der Offset definiert die Lage der Frame-Directory.

Im Feld ab Offset 50H definieren die Zahl der Bilder pro Sekunde für die Wiedergabe (Playback Rate). Ein Wert 00H ab Offset 52H zeigt eine unmodifizierte Datei an. Die restlichen Bytes werden zum Füllen der Struktur bis zur 120-Byte-Grenze benutzt.

Der Stream-Header

Die DVI-Dateien können einen oder mehrere Daten-Streams enthalten. Jedem Stream wird ein eigenen Header vorangestellt. Dieser Header besitzt folgende Struktur:

Offset	Bytes	Bemerkungen
00H	4	Signatur 'STRM'
04H	2	Typ
06H	2	Subtyp
08H	2	Zahl der Substream-Header
0AH	2	ID Next Stream
0CH	2	Group ID Current Stream
0EH	2	Unbenutzt

Tabelle 48.4 Struktur eines Stream-Headers

Offset	Bytes	Bemerkungen
10H	4	Flags
14H	4	Frame Size
18H	4	Offset 1. Subframe-Header
1CH	16	Streamname

Tabelle 48.4 Struktur eines Stream-Headers

Das erste Feld enthält die Signatur ('STRM'). Daran schließen sich zwei Worte mit dem Typ und dem Subtyp des Streams an. Für den Typ gilt:

Typ	Bemerkungen
02H	Compressed Audio Stream
03H	Compressed Image Stream
05H	Associated per Frame Data
06H	Uncompressed Image Stream
07H	Pad Stream

Tabelle 48.5 Stream-Typen

Der Subtyp definiert Variationen innerhalb der Daten.

Subtyp	Bemerkungen
00H	No Subtype for Audio Data
01H	Y-channel data only
0BH	U-channel data only
0CH	V-channel data only
0DH	YVU data
0EH	YUV data

Tabelle 48.6 Stream-Subtypen

Bei Audiodaten gibt es keine Subtypen, d.h. der Wert wird immer auf 0 gesetzt. Bei Videodaten definiert der Subtyp die Kodierung der Kanäle.

Das Wort ab Offset 08H definiert die Zahl der Sub-Streams. Das folgende Feld ist unbelegt und wird auf FFFFH (-1) gesetzt. Daran schließt sich eine ID-Nummer für den aktuellen Stream an.

Die Flags ab Offset 10H zeigen mit dem Wert 04H an, daß sich die Größe der Einzelbilder ändert.

Das Feld Frame Size (Offset 14H) definiert die Größe eines Frames im Stream in Byte. Daran schließt sich der Offset auf den ersten Substream-Header an. Der Stream-Name wird als ASCIIZ-String abgelegt.

Der Audio Stream-Header

Auf jeden Stream-Header folgt ein Sub-Stream Header. Bei Audio-Daten enthält der Stream Header als Typ den Eintrag 02H (siehe oben). Gleichzeitig wird der Eintrag im Feld Subtyp auf 00H gesetzt. Der Header umfaßt 168 Byte mit folgender Struktur:

Offset	Bytes	Bemerkungen
00H	4	Signatur ('AUDI')
04H	2	Header-Größe
06H	2	Header-Version
08H	80	Filename
58H	4	ID Original Frame
5CH	2	ID Original Stream
5EH	2	Unbenutzt
60H	4	Frame Counter
64H	4	Offset Next Header
68H	16	Name Library
78H	16	Compression Algorithm
88H	4	Datenrate (Bits/sec)
8CH	2	Filter Cut Off Frequenz
8EH	2	Unbenutzt
90H	2	Volume linker Kanal
92H	2	Volume rechter Kanal
94H	4	Unbenutzt
98H	4	ID Start Frame
9CH	4	Flags (Mono, Stero)
A0H	2	Playback Rate
A2H	2	Unbenutzt
A4H	4	Compression facility ID

Tabelle 48.7 Struktur des Audio Stream-Headers

Die ersten vier Byte enthalten die Signatur 'AUDI', gefolgt von der Längenangabe (in Byte) für den Header. Das folgende Wort definiert die Versionsnummer (5) für die Headerstruktur. Daran schließt sich ein Dateiname von 80 Byte an. Der Name definiert die

Datei, aus der der Datenstrom übernommen wurde. Der String ist mit einem Nullbyte abzuschließen.

Die nächsten drei Felder werden nicht verwendet und zu 0 gesetzt. Das Feld Frame Counter (Offset 60H) definiert die Zahl der Audio-Frames in diesem Stream. Das Feld mit dem Offset auf den nächsten Sub-Stream wird immer auf 0 gesetzt. Das gleiche gilt für den Bibliotheksnamen. Der Name des Kompressionsalgorithmus wird als ASCIIZ-String definiert (z. B. 'apdcm4e' oder 'pcm8').

Ab Offset 88H folgt die Datenrate des Audio-Kanals in Bit pro Sekunde. Das nächste Wort definiert die maximale Frequenz innerhalb des Audio-Kanals. Höhere Frequenzen wurden bei der Aufzeichnung ausgefiltert.

Die Lautstärke (Volume) der Audio-Kanäle wird in Prozent angegeben. Der Start Frame wird auf 0 gesetzt und das Flag definiert Mono- (4000H) oder Stereobetrieb (8000H). Die restlichen Byte sind unbenutzt, bzw. werden auf 00 oder FFH gesetzt.

Der Video Stream-Header

Enthält ein Stream Videodaten, wird der Stream-Header von einem Video Substream-Header (136 Byte) gefolgt. Dieser besizt folgende Struktur:

Offset	Bytes	Bemerkungen
00H	4	Signatur ('CMIG')
04H	2	Header Größe
06H	2	Header Version
08H	80	Filename
58H	4	ID Orignal Frame
5CH	2	ID Original Stream
5EH	2	Unbenutzt
60H	4	Frame Counter
64H	4	Offset Next Substream Header
68H	2	X-Koordinate oben links
6AH	2	Y-Koordinate oben rechts
6CH	2	Bildbreite in Pixel
6EH	2	Bildhöhe in Pixel
70H	2	X Cropping Koordinate
72H	2	Y Cropping Koordinate
74H	4	Unbenutzt
78H	4	Still Periode

Tabelle 48.8 Struktur eines Video Substream-Headers

Offset	Bytes	Bemerkungen
7CH	2	Buffer Minumum
7EH	2	Buffer Maximum
80H	2	ID Decompressions Algorithmus
82H	2	Unbenutzt
84H	4	Compression Facility ID

Tabelle 48.8 Struktur eines Video Substream-Headers

Die ersten vier Byte enthalten die Signatur 'CMIG', gefolgt von der Längenangabe (in Byte) für den Header. Das folgende Wort definiert die Versionsnummer (4) für die Headerstruktur. Daran schließt sich ein Dateiname von 80 Byte an. Der Name definiert die Datei, aus der der Datenstrom übernommen wurde. Der String ist mit einem Nullbyte abzuschließen.

Die nächsten drei Felder werden nicht verwendet und zu 0 gesetzt. Das Feld Frame Counter (Offset 60H) definiert die Zahl der Video-Frames in diesem Stream. Das Feld mit dem Offset auf den nächsten Sub-Stream wird immer auf 0 gesetzt.

Ab Offset 68H folgen die Koordinatenangaben für die linke obere Ecke des Bildes, die Bildabmessungen und eventuell Informationen zum Bildausschnitt (crop).

Das Feld Still Period definiert das Intervall, in dem intraframe encoding erfolgt. Dieses Feld und die restlichen Felder werden auf 0 oder FFH gesetzt.

Die Frames-Struktur

Jeder Frame innerhalb des Datenbereiches (Audio und Video) enthält einen vorgeschalteten Header.

Offset	Bytes	Bemerkungen
00H	4	Sequenznummer Frame
04H	4	Offset vorhergehendes Frame
08H	4	Checksumme
0CH	n*4	Array mit allen Frame Größen

Tabelle 48.9 Struktur eines Frame-Headers

Der Header beginnt mit der Sequenznummer des Frames. Daran schließt sich ein Zeiger auf den vorhergehenden Frame an. Beim ersten Frame enthält der Zeiger den Wert 0.

Die Checksumme bezieht sich auf den Header. Das Feld ab Offset 0CH enthält n Einträge zu je 4 Byte mit der Bytezahl der Frames, die in diesem Stream gespeichert sind.

Die Position jedes Frames wird in einem Verzeichnis (directory) gespeichert. Pro Frame wird eine Frame-Directory-Struktur im Stream gespeichert. Diese enthält einen 4-Byte-Zeiger mit dem Offset auf den Frame.

49 Graphics Interchange Format (GIF87a)

1987 wurde von der Firma CompuServe ein Protokoll zum Austausch von Grafikdaten über Mailboxen definiert. Über das Format ist es möglich, mehrere Bilder in einer Datei zu speichern. Dekodierung und Komprimierung wurden so ausgelegt, daß eine schnelle Bildwiedergabe möglich ist. Das Grafikformat ist hardwareunabhängig und erlaubt es, Bilder mit bis zu 16.000 x 16.000 Punkten und aus bis zu 256 Farben zu kombinieren, die sich aus 16 Millionen Farbstufen zusammensetzen können.

Eine GIF-Datei besteht aus mehreren Blöcken zur Aufnahme der Datei. Dabei werden ähnlich wie beim TIFF-Format *Tag*-Felder benutzt. Tabelle 49.1 zeigt die prinzipielle Struktur der Dateien:

Offset	Bytes	Feld
00H	3	Signatur 'GIF'
03H	3	Version '87a' oder '89a'
06H	7	Logical Screen Descriptor Block
0DH	n	Global Color Map (optional)
...	n	Extension Block (optional)
...	n	Image Descriptor Block
...	n	Local Color Map (optional)
...	n	Raster Data Block
...	n	Extension Block (optional)
...	1	Terminator 3BH=;

Tabelle 49.1 Die Struktur einer GIF-Datei

In den ersten drei Bytes steht eine Signatur für den Header (meist 'GIF'). Daran schließen sich drei Byte mit der GIF-Version an (z. B. '87a' oder '89a'). Ab Offset 06H beginnt ein 7 Byte großer Block mit den Daten des logischen Bildschirms (*Logical Screen Descriptor Block*) mit folgendem Aufbau:

Bytes	Feld
2	Screen width
2	Screen height
1	Resolution flag
1	Background color
1	Pixel aspect ratio

Tabelle 49.2 Aufbau des Logical Screen Descriptor-BlocksAufbau des Logical Screen Descriptor-Blocks

Der erste Eintrag umfaßt einen 16-Bit-Wert und gibt die Breite des logischen Bildschirms in Pixeln an. Dabei wird die Intelnotation (*Low Byte first*) zur Speicherung benutzt. Das nächste Feld mit der Bildschirmhöhe enthält ebenfalls einen 16-Bit-Wert. Ab Offset 04H findet sich im *Logical Screen Descriptor*-Block (LSDB) ein Bitfeld (*Resolution Flag*) mit der Kodierung gemäß Bild 49.1.

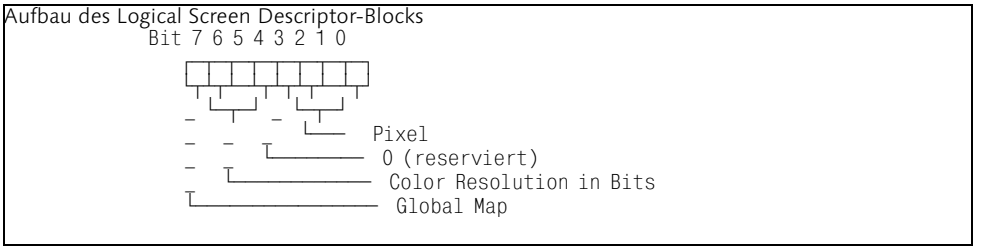


Abbildung 49.1 Kodierung des Resolution-Flags

Bit 3 des *Resolution*-Flags ist bisher nicht belegt. Das oberste Bit 7 markiert, ob eine *Global Color Map* existiert. Ist Bit 7 gleich 1 gesetzt, folgt auf den *Logical Screen Descriptor*-Block die globale Color Map. In den Bits 4 bis 6 wird festgehalten, wie viele Bits zur RGB-Darstellung einer Farbe in der Farbtabelle zur Verfügung stehen. Der Wert in den Bits ist jeweils um 1 zu erhöhen. Wert 7 besagt, daß pro Farbe 8 Bit in der entsprechenden Tabelle benutzt werden. Der Wert in den Bits 0 bis 2 muß ebenfalls um 1 erhöht werden und gibt die Anzahl der Bits pro Bildpunkt an.

Ab Offset 05H findet sich im *Logical Screen Descriptor*-Block ein Byte mit der Kodierung der Hintergrundfarbe. Diese Hintergrundfarbe wird aus den 256 möglichen Farben ausgewählt. Das Byte ab Offset 06H im LSDB ist gemäß Bild 49.2 kodiert, es enthält jedoch meistens den Wert 0.

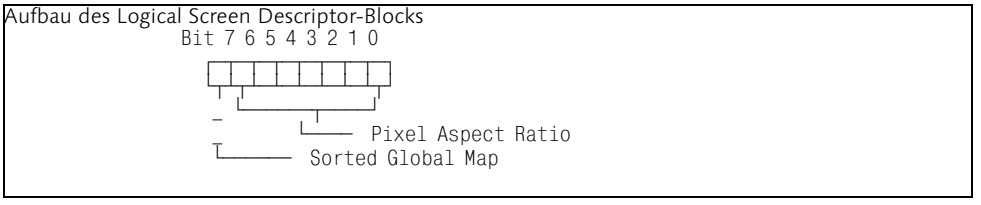


Abbildung 49.2 Kodierung des Pixel Aspect Ratio-Flags

Im Anschluß an den *Logical Screen Decriptor*-Block kann optional ein Block mit der Global-Color Map gespeichert sein. Dies ist immer dann der Fall, wenn im vorhergehenden *Logical Screen Descriptor*-Block das Bit 7 des *Resolution*-Flags gesetzt ist. Im *Color Map*-Block wird die globale Farbtabelle für die nachfolgenden Bilder spezifiziert; für jeden Bildpunkt sind 8 Bit vorgesehen, womit sich maximal 256 Farben oder Graustufen abbilden lassen. Um das Farbspektrum feiner abzustufen, kann man die 256 Farben aus 16

Millionen Farbschattierungen selektieren. Hierzu wird in der Farbtabelle für jede der 256 Farben ein Tripel (3 Byte) mit den Grundfarben Rot, Grün und Blau abgelegt (Bild 49.3).

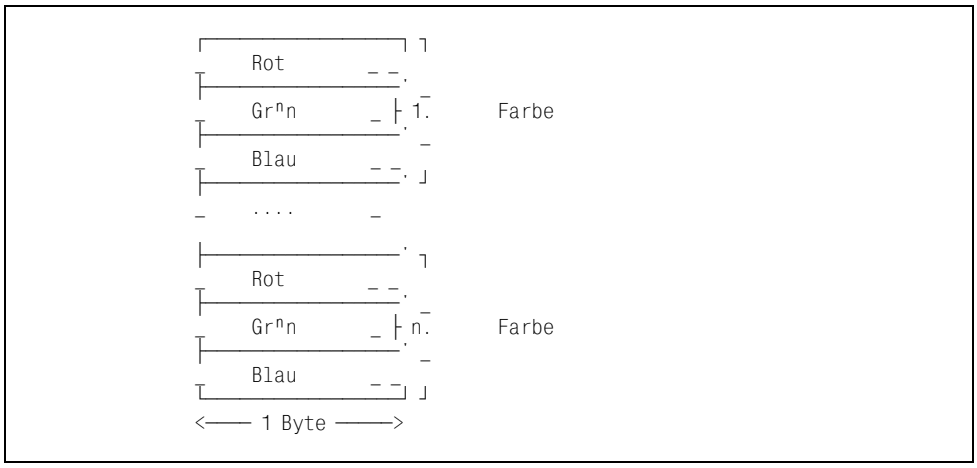


Abbildung 49.3 Der Aufbau der Global Color Map

Jeder dieser drei Grundfarben ist ein Byte zugeordnet. Der in dem Byte abgelegte Wert bestimmt den Farbanteil (*Intensität*) in der Mischfarbe. Mit drei Byte pro Farbe lassen sich 16 Millionen Farbstufen darstellen; die Farbtabelle enthält deshalb 256x3 Byte mit den Farbwerten. Die Farbkodierung eines Bildpunktes läßt sich dann in die Farbtabelle als 8-Bit-Offset kodieren, und ein Bild kann durch Variation der Farbtabelle in mehreren Farbnuancen dargestellt werden. Die Größe der Farbtabelle und die Zahl der Bits pro Farbe wird ebenfalls im Resolution Flag spezifiziert (3 Byte * 2 ** Bit pro Pixel).

Neben der globalen Farbtabelle lassen sich auch *vor* den Raster-Image-Blöcken lokale Farbtabellen definieren. Diese Definitionen haben für den betreffenden Block Vorrang, und die globale Tabelle entfällt oder wird für den Block nicht ausgewertet. Der Aufbau der lokalen Tabelle, von der gegebenenfalls auch mehrere angelegt werden können, stimmt mit dem der globalen Farbtabelle überein.

An den Block mit der Farbkodierung schließt sich ein optionaler Extension-Block (Erweiterungsblock) an. Über diese »Extension-Blocks« wurde ein Weg für zukünftige Erweiterungen des GIF-Formats geschaffen. Im GIF-Format dient der erste Erweiterungsblock zum Speichern von Informationen über das bilderzeugende Gerät, die benutzte Software, die Ausrüstung zur Bildabtastung etc. In einem zweiten Erweiterungsblock werden Daten zum Bildaufbau (Kontrolldaten) abgelegt. Die Blöcke haben den in Tabelle 49.3 beschriebenen Aufbau.

Das erste Byte des Erweiterungsblocks enthält das Zeichen »!« als Signatur. Darauf folgt ein Byte mit dem Funktionscode (Belegung nicht festgelegt), der die Art der nachfolgen-

den Daten definiert. Anschließend folgt der Datenbereich, der mehrere Records der gleichen Struktur enthalten kann. In jedem dieser Records findet sich im ersten Byte die Angabe über die Anzahl der nachfolgenden Datenbytes, womit ein Datenbereich die maximale Länge von 255 Byte besitzen kann.

Bytes	Feld
1	Extension Block Header (ASCII 21H = '!')
1	Funktionscode (0 bis 255)
1	Länge Datenblock 1 (in Byte)
n	Datenblock 1
1	Länge Datenblock 2
n	Datenblock 2
...	...
1	Länge Datenblock n
n	Datenblock n
1	00H als Terminator

Tabelle 49.3 Aufbau eines Extension-Blocks

Offset	Bytes	Feld
00H	1	Image Separator Header (ASCII 2CH = ',')
01H	2	Koordinate linker Bildrand
03H	2	Koordinate oberer Bildrand
05H	2	Bildbreite
07H	2	Bildhöhe
09H	1	Flags

Tabelle 49.4 Aufbau eines Image Descriptor-BlocksAufbau eines Image Descriptor-Blocks

Bei mehr Daten müssen mehrere Records abgelegt werden. Das Ende des Erweiterungsblocks wird durch ein Nullbyte (00H) markiert. Über die genaue Belegung der Datenbereiche liegen zur Zeit keine Informationen vor. Die drei Blöcke »Image Descriptor-Block«, »Local Color Map« und »Raster Data-Block« können durchaus mehrfach als Sequenz in einer GIF-Datei auftreten.

Das Bildformat wird in einem eigenen 10 Byte großen Block, dem *Image Descriptor*-Block abgelegt (Tabelle 49.4). Der Block enthält die wichtigsten Eckdaten eines Bildes wie Abmessungen, Koordinaten der linken oberen Ecke etc. Das erste Byte wird mit dem Separator (',') gefüllt. Die Felder für Bildkoordinaten und -abmessungen (in Pixeln) umfassen jeweils 16 Bit. Das letzte Byte dient zur Aufnahme verschiedener Flags, die folgendermaßen kodiert sind (Bild 49.4):

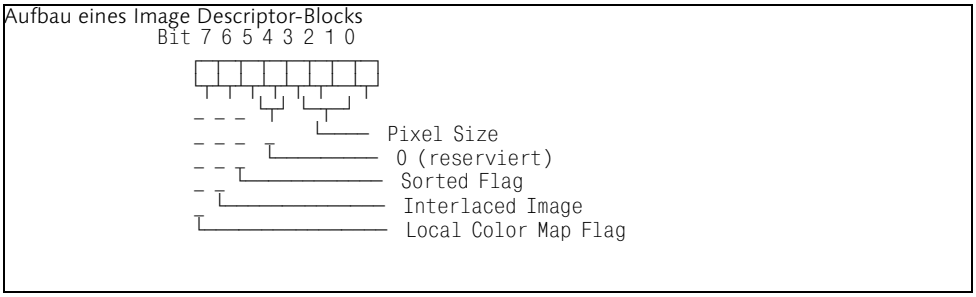


Abbildung 49.4 Kodierung des Flags im Image Descriptor-Block (IDB)

Das oberste Bit spezifiziert, ob sich eine lokale Color Map an den IDB anschließt. In diesem Fall ist das Bit gesetzt, und die Farben des Teilbildes beziehen sich auf diese Tabelle. In Bit 6 zeigt ein Flag an, ob das Grafikbild *sequentiell* (Bit 6 = 0) oder *interlaced* (Bit 6 = 1) auszugeben ist. Der sequentielle Modus gibt Zeile für Zeile des Bildes aus. Im Gegensatz dazu wurde der *Interlaced Mode* geschaffen, um bei einer Übertragung per Telefonleitung möglichst schnell über ein Rohbild zu verfügen. In diesem Modus wird bei jedem Durchlauf nur jede achte Zeile übertragen und ausgegeben, so daß nach dem ersten Durchlauf zunächst die 0., 8., 16. usw. Zeile vorliegt. Die fehlenden Zeilen werden dann in den folgenden Durchläufen in der Folge der Anfangsreihen 4, 2, 1, 3, 5, 7 ergänzt. Beim zweiten Durchlauf erhält man demnach die 4., 12., 20. usw. Zeile und beim dritten Durchlauf die 2., 10., 18. usw. Zeile. Bit 5 gibt an, wie die lokale Farbtabelle sortiert ist (Frequenz oder Farbe); die unteren Bits 0 bis 2 zeigen, wie viele Bits pro Bildpunkt gespeichert werden. Bit 5 wird aber nur in wenigen Anwendungen benutzt, da die offizielle GIF-Dokumentation dieses Bit nicht erwähnt und zu 0 setzt.

Ist im IDB-Flag (Bild 49.4) das Flag für die Local Color Map gesetzt, folgen die Daten der Farbtabelle. Der Aufbau entspricht dem der Global Color Map. Anschließend folgt der *Raster Data*-Block mit den eigentlichen Bilddaten (Bild 49.5). Er folgt entweder auf den *Image Descriptor*-Block oder, falls verwendet, auf die lokale Color Map. Die Daten sind dabei als Index in die Color Map zu interpretieren.

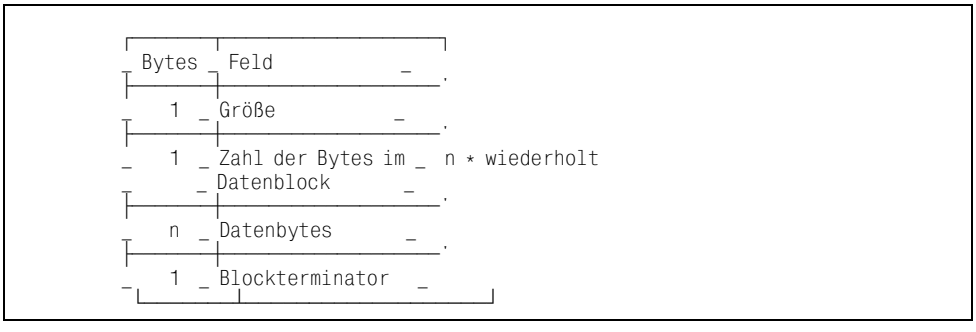


Abbildung 49.5 Aufbau eines Raster Data-Blocks

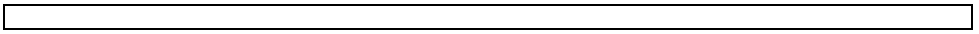


Abbildung 49.5 Aufbau eines Raster Data-Blocks

Im ersten Byte steht die Zahl der Bits pro Pixel. Darauf folgt ein Zähler, der die Zahl der Bytes im nachfolgenden Datenblock angibt. Den Abschluß bildet der Blockterminator. Dieser besitzt den Wert 00H. Die Daten im Datenblock liegen in komprimierter Form vor; zur Kodierung wird der LZW-Algorithmus von Lempel-Ziv & Welch in leicht modifizierter Form benutzt. Dieser Algorithmus baut eine Tabelle mit gelesenen Zeichen auf. Diese gelesenen Zeichen werden nach Möglichkeit zu Ketten zusammengefaßt und mit Mustern in der Tabelle verglichen. In der Ausgabedatei wird lediglich der *Index* zur Position des Musters in der Kodierungstabelle gespeichert, die durch neu auftretende Muster ständig ergänzt wird, bei GIF-Dateien dürfen die Codefolgen zwischen 3 und 12 Bit lang sein. Ist die Tabelle voll, wird ein Reset-Code abgespeichert, und der Aufbau der Tabelle beginnt erneut. Eine genauere Beschreibung des LZW-Verfahrens finden Sie in der im Anhang angegebenen Literatur unter /6, 13/.

Die Blöcke *Image Desciptor*, *Local Color Map* und *Raster Data* lassen sich bei Bedarf mehrfach in einer Datei anlegen. Dies erlaubt es, mehrere Bilder in einer Datei zu speichern. Im Anschluß an den letzten *Raster Data*-Block erscheint optional ein zusätzlicher Erweiterungsblock. Den Abschluß für die GIF-Datei übernimmt ein Semikolon (3BH) als Terminator.

Anmerkung: Die Beschreibung des GIF89a-Formats finden Sie im folgenden Kapitel.

53 Windows ICON-Format (ICO)

Die Icons in Windows werden als Bitmaps mit verschiedenen Auflösungen gespeichert. Die Dateien besitzen die Erweiterung ICO und sind in folgender Struktur gespeichert:

Offset	Bytes	Bedeutung
		Header
00H	2	reserviert (muß 0 sein)
02H	2	Resource Typ (1 für Icons)
04H	2	Zahl der Bilder in der Datei
		Resource Directory
06H	1 1 1 1 2 2 4 4	Feld mit *n* Einträgen gemäß folgender Struktur: Icon Breite in Pixeln (Width) Icon Höhe in Pixeln (Height) Zahl der Farben (ColorCount) reserviert Planes (Windows 3.1) Bits in Icon-Bitmap (Windows 3.1) Größe Pixelarray in Byte (icoDIBSize) Offset in Byte auf Bilddaten ... ggf. weitere Einträge
..	40	TBitMapInfoHeader
..	64	16*4-TRGBQuad-Farbtabelle Farben blau, grün, rot, Intensität
..	512	Bytesequenz mit Color Bitmap (XOR)
	128	Bytesequenz mit Monochrom Bitmap (AND)

Tabelle 53.1 Aufbau des Windows-Icon-Formats

Windows ermittelt die Auflösung eines Ausgabegerätes und rechnet dann die Bildpunkte des Icons entsprechend um.

Das erste Wort der Icon-Datei ist reserviert und muß 0 sein. Daran schließt sich ein Wort mit dem Typ der Ressource an. Für Icons ist hier der Wert 1 einzutragen (Cursor-Bilder haben den gleichen Header und belegen das Feld mit dem Wert 2).

Das Wort ab Offset 04H gibt die Zahl der Bilder in der Datei an. Meist ist jedoch nur ein Icon in einer ICO-Datei gespeichert.

An diesen Header schließt sich ein Feld mit *n* Einträgen an, welches das Icon beschreibt. Für jedes Bild enthält diese Tabelle die Abmessungen des Icons *Width* und *Height*. Dabei sind die Werte 16, 32 und 64 Pixel erlaubt. Das Feld *ColorCount* enthält die Zahl der ver-

wendeten Farben (2, 8 oder 16). Die nächsten 5 Byte sind in Windows 3.0 reserviert. In Windows 3.1 ist nur das erste Byte reserviert. Die folgenden 4 Bytes geben die Größe des Pixelarrays (in Byte) des betreffenden Icons an. Das folgende Doppelwort enthält den Offset vom Anfang der Datei auf das erste Byte der Bilddaten (DIB).

Die Struktur der folgenden 40 Byte ist unbekannt. Daran schließt sich die Definitionstabelle für die Farbpalette an. Jede Farbe besitzt dabei 4 Byte zur Definition. An die Farbtabelle schließt sich der Datenbereich an. Die Anfangsadresse wird im Header definiert. Die Bilddaten liegen unkomprimiert vor, wobei für jedes Icon zwei Bitmaps gespeichert werden. Die erste Bitmap enthält die XOR-Maske mit dem Farbbild für das Icon. Diese Bitmap umfaßt 512 Byte. Daran schließt sich die AND-Maske mit dem monochromen Bild an. Dieses Bild umfaßt 128 Byte und wird für den transparenten Teil des Icons genutzt.

Anmerkung: In Windows 95 wird ein erweitertes ICO-Format verwendet. Informationen hierzu finden sich auf den CD-ROMs des Windows 95-SDK, die im Rahmen des Developers Network Program (Level 2) erhältlich sind.

60 Die MPEG-Spezifikation

Mit MPEG wurde ein Standard von der Motion Picture Expert Group zur Übertragung von Bewegtbildern geschaffen. Dieser Standard ist in der (draft) ISO-Norm CD 11172 beschrieben. Zur Zeit ist kein Fileformat für MPEG-Daten definiert. Ein MPEG-Player liest den Datenstrom und decodiert die darin enthaltenen Bilder.

73 Das Soundblaster Instrument File-Format (SBI)

Die Firma Creative Labs hat weitere Dateiformate zur Speicherung von Musikdaten entwickelt, die das CMF-Format erweitern. Ein Problem liegt in der Anpassung der Instrumentdefinitionen an die Soundkarte. Liegen CMF-Dateien vor, sind Sie auf die Instrumentdefinitionen im Instrument Block angewiesen. Sie können diese Parameter variieren, um den Klang eines Instruments optimal einzustellen. Haben Sie dann die für Ihre Vorstellungen optimale Einstellung gefunden, stehen Sie vor einem Problem. Sie müssen alle CMF-Datei bearbeiten und die Instrumentdefinitionen im Header austauschen.

Um diese Schwierigkeiten zu umgehen, hat Creative Labs das Soundblaster Instrument File-Format (SBI) definiert. Hier werden nur die Daten eines Instruments gespeichert. Ein Programm kann daher diese Informationen vor einer CMF-Datei einlesen. Benutzt die CMF-Datei das Instrument, werden die Instrumentvorgaben aus der SBI-Datei verwendet.

Der Aufbau einer SBI-Datei ist recht einfach. Es umfaßt immer nur die Daten eines Instruments und ist auf eine feste Länge von 52 Byte begrenzt (Tabelle 73.1). Die Daten werden dabei im Intel-Format abgespeichert.

Offset	Bytes	Bemerkungen
00H	4	File ID (53 42 49 1A)
04H	30	Instrument Name
24H	1	Modulator Charakteristik
25H	1	Carrier Charakteristik
26H	1	Modulator Amplitude
27H	1	Carrier Amplitude
28H	1	Modulator Attack/Decay
29H	1	Carrier Attack/Decay
2AH	1	Modulator Sustain/Release
2BH	1	Carrier Sustain/Release
2CH	1	Modulator Wave Form
2DH	1	Carrier Wave Form
2EH	1	Feedback/Link
2FH	5	Reserviert

Tabelle 73.1 Struktur einer Instrument-Datei (SBI)

Die ersten vier Byte enthalten die Signatur 'SBI', gefolgt von dem Zeichen 1AH. Damit wird gleichzeitig eine Anzeige der Datei als DOS-TextDatei verhindert. An diese Signatur

schließt sich der Name des Instruments als ASCIIZ-String an. Sie müssen dabei nicht alle Bytes ausnützen, sollten aber einheitliche Namen zur Identifizierung verwenden.

Anschließend folgt die Charakteristik des Modulators. Das Byte wird dabei gemäß Tabelle 73.2 strukturiert.

Bit	Bemerkungen
0-3	Multiplikationsfaktor
4	Hüllkurve abfallend
5	Hüllkurventyp (Envelope)
6	Vibrato Effect
7	Tremolo Effect

Tabelle 73.2 Kodierung der Modulator-Charakteristik

Das folgende Byte beschreibt die Charakteristik des Trägersignals. Hierbei gilt die gleiche Kodierung wie beim Modulator (siehe Tabelle 73.2).

Das Byte ab Offset 26H definiert die Modulator-Amplitude. Dieses Byte enthält den Dämpfungsfaktor und den Parameter für die Amplitudendämpfung:

Bits	Bemerkungen
0-5	Dämpfungsfaktor
6-7	Amplitudendämpfung

Tabelle 73.3 Kodierung der Modulator-Amplitude

Der Dämpfungsfaktor kann Werte zwischen 0 und 63 annehmen, wobei 0 keine Dämpfung und 63 maximale Dämpfung mit minimaler Lautstärke darstellt.

Das vierte Byte eines Instrument Records beschreibt die Träger Amplitude. Die Kodierung entspricht der Modulator Amplitude (Tabelle 73.3).

Ab Offset 04H findet sich der Modulator Attack/Decay-Wert. Dieses Byte teilt sich ebenfalls bitweise auf:

Bits	Bemerkungen
0-3	Decay
4-7	Attack

Tabelle 73.4 Kodierung Attack/Decay-Byte

Für den Träger werden die gleichen Informationen ab Offset 05H abgelegt.

Im Byte ab Offset 06H findet sich der *Modulator Sustain/Release*-Wert. Das Byte beschreibt zwei Werte einer ADSR-Hüllkurve:

Bits	Bemerkungen
0-3	Release
4-7	Sustain

Tabelle 73.5 Kodierung Sustain/Release-Byte

Ab Offset 07H findet sich das Byte mit dem Sustain/Release-Wert für den Träger. Hierbei gilt ebenfalls die in Tabelle 73.5 gezeigte Kodierung.

Offset 08H und 09H enthalten je ein Byte mit der Wellenform des Modulators und des Trägers. Hierbei wird die Wellenform nur in den beiden unteren Bits 0,1 kodiert. Die Bits 2 bis 7 müssen gelöscht sein.

Die Informationen über die Rückkopplung/Verbindung wird im Byte ab Offset 0AH gespeichert. Hierbei gilt die Kodierung aus Tabelle 73.6.

Bits	Bemerkungen
0	Link
1-2	Feedback Modulator
4-7	Reserviert (muß 0 sein)

Tabelle 73.6 Kodierung Feedback/Link-Byte

Die restlichen 5 Byte wurden für spätere Erweiterungen freigelassen.

Haben Sie nun Anpassungen für ein Instrument auszuführen, müssen Sie lediglich die betreffende SBI-Datei korrigieren. Der Nachteil dieses Formats liegt darin, daß für jedes Instrument eine eigene Datei geladen werden muß.

76 Das Adlib Music-Format (ROL)

Die Firma Adlib hat ein eigenes Format zur Speicherung von Sounds für Adlib Karten definiert. Diese Datei gliedert sich in einen Header, gefolgt von Blöcken mit den Beschreibungen der Music Events an. Die Daten werden dabei im Intel-Format abgelegt.

Abbildung 76.1 Struktur einer ROL-Datei

Der Header enthält dabei die Kennung, sowie die Versionsnummer. Der folgende Datenbereich unterteilt sich in Events mit der Beschreibung der Sounddaten.

Der ROL-Header

Der Header einer ROL-Datei umfaßt immer C9H Byte und ist gemäß Tabelle 76.1 strukturiert.

Offset	Bytes	Bemerkungen
00H	4	Format Version
04H	40	Intern benutzt
2CH	2	Timer Ticks 1/4 Note
2EH	7	Für Visual Composer benutzt
35H	1	Play Modus, 0 = Rhythmus, 1 = Melody
36H	143	intern benutzt
C5H	4	Speed

Tabelle 76.1 Struktur eines ROL-Headers

Die meisten 20 Byte des Headers dienen internen Zwecken. Die Anzahl der Uhrtakte pro Viertelnote wird ab Offset 2CH definiert. Ab Offset 35H findet sich der Abspielmodus. Der letzte Wert (Offset C5H) enthält den Wert für das Grundtempo als 4-Byte-Gleitkommazahl.

Der ROL-Datenbereich

An den Header schließt sich der Datenbereich der ROL-Datei an. Dieser Datenbereich wird in verschiedene Subblöcke aufgeteilt. Diese Subblöcke besitzen folgende Struktur.

Der Tempo-Block

Als erstes steht ein Block mit den Tempodefinitionen im Datenbereich. Dieser Block besitzt eine variable Länge mit folgender Struktur:

Offset	Bytes	Bemerkungen
00H	2	Zahl der Tempo Events
02H	2	Länge Tempo Events
04H	4	Tempo Multiplikator
...	...	Wiederholung letzte 2 Felder

Tabelle 76.2 Struktur des Tempo-Blocks

Das erste Wort gibt die Zahl der Tempo Events in diesem Bereich an. Für jedes Tempo Event enthält der Bereich die beiden Felder mit der Längenangabe (Offset 02H) und das Tempo. Die Länge eines Events wird dabei in Uhrtakten (Timer Ticks) angegeben. Das Tempo wird als 4-Byte-Fließkommazahl definiert. Dieser Wert ist mit der Vorgabe für das Grundtempo zu multiplizieren und ergibt das Tempo für das zugehörige Event.

Der Note-Block

An den Tempo-Block schließt sich der *Note*-Block an. Dieser besitzt folgenden Aufbau:

Offset	Bytes	Bemerkungen
00H	15	Intern benutzt
0FH	2	Länge aller Noten
11H	2	Note
13H	2	Länge Note
...	...	Wiederholung letzte 2 Felder

Tabelle 76.3 Struktur eines Note-Blocks

Die ersten 15 Byte des Blocks sind für eine interne Verwendung reserviert. Das Byte ab Offset 0FH definiert die Länge aller Noten in Uhrtakten (Timer Ticks). Ab Offset 11H folgt ein Wort mit dem Wert der Note. Dieser Wert muß vor der Ausgabe an den Treiber um 60 verringert werden. Der Wert 0 steht für Stille. Das Wort ab Offset 13 gibt die Länge der Note in Uhrtakten (Timer Ticks) an. Die beiden Felder für die Note und die Länge wiederholen sich solange, bis die Gesamtlänge aller Noten erreicht ist.

Der Instrument-Block

An den Notenbereich schließt sich ein Block mit der Instrumentdefinition an. Dieser Bereich besitzt folgende Struktur:

Offset	Bytes	Bemerkungen
00H	15	Intern benutzt
0FH	2	Zahl der Instrument Events
11H	2	Länge der Instrument Event
13H	9	Instrument Name
1CH	3	Intern benutzt
...	...	Wiederholung letzte 3 Felder

Tabelle 76.4 Struktur einer Instrument-Area

Die ersten 15 Byte sind für die interne Benutzung reserviert. Daran schließt sich ein Wort an, welches die Zahl der folgenden Einträge (Zahl der Instrumente) angibt. Für jedes Instrument werden dann drei Felder gespeichert. Das erste Feld (Wort) definiert die Länge des Instrument Events in Uhrtakten. Die folgenden 9 Byte enthalten den Namen des Instruments als String, der mit einem Nullbyte abzuschließen ist. Die letzen drei Byte sind für interne Zwecke reserviert. Diese Struktur wiederholt sich *n* mal.

Der Volume Block

Dieser Bereich definiert die Lautstärke. Für den Block gilt folgende Struktur.

Offset	Bytes	Bemerkungen
00H	15	Intern benutzt
0FH	2	Zahl der Volume Events
11H	2	Länge der Volume Events
13H	4	Volume Shift
...	...	Wiederholung letzte 2 Felder

Tabelle 76.5 Struktur eines Volume-Blocks

Die ersten 15 Byte sind reserviert. Dann folgt ein Wort mit der Zahl der eingetragenen Events. Jedes Event besteht aus 6 Byte, wobei das erste Wort die Länge des Volume Events in Uhrtakten angibt. Die folgenden vier Byte enthalten eine Gleitkommazahl mit der Lautstärkeänderung. Diese Felder werden für jeden Event wiederholt.

Der Tonhöhe Block

Der letzte Block in der ROL-Datei definiert die Tonhöhe (Frequenz) und besitzt folgenden Aufbau:

Offset	Bytes	Bemerkungen
00H	15	Intern benutzt

Tabelle 76.6 Struktur eines Frequence-Blocks

Offset	Bytes	Bemerkungen
0FH	2	Zahl der Frequence Events
11H	2	Länge der Frequence Events
13H	4	Frequence Shift
...	...	Wiederhole letzte 2 Felder

Tabelle 76.6 Struktur eines Frequence-Blocks

Die ersten 15 Byte sind für interne Zwecke reserviert. Dann folgt ein Wort mit der Zahl der Tonhöhe Events (Frequenz Events). Die einzelnen Events umfassen jeweils 6 Byte mit der Länge des Tonhöhe Events in Uhrtakten und einer 4-Byte-Gleitkommazahl mit der Veränderung der Tonhöhe. Diese beiden Felder werden für jedes Event wiederholt.

77 Das Adlib Instrument Bank-Format (BNK)

Neben dem ROL-Format gibt es ein weiteres Format, welches zur Speicherung der Instrumentbeschreibung dient. Die BNK-Dateien nehmen ähnlich wie bei den Soundblaster-Dateien die Instrumentbeschreibung auf. Eine BNK-Datei besitzt folgendes Format.

Offset	Bytes	Bemerkungen
00H	2	Format Version
02H	6	Signature 'ADLIB'
08H	2	Zahl benutzte Instrumente
0AH	2	Zahl gespeicherte Instrumente
0CH	4	Startadresse Instrumentname
10H	4	Startadresse Instrumentdaten
14H	8	Reserviert
1AH	n*18	Instrumentnamen
..H	n*33	Instrumentdaten

Tabelle 77.1 Struktur einer BNK-Datei

Das erste Wort enthält die Versionsnummer, gefolgt von der Signatur 'ADLIB'. Ab Offset 08H steht die Zahl der benutzten Instrumente, gefolgt von einem Wort mit der Zahl der definierten Instrumente. Dieser Wert ist zur Berechnung der Längen der Instrumententabellen wichtig. Die folgenden zwei 4-Byte-Felder enthalten Zeiger auf den Anfang der Tabellen mit den Instrumentnamen und Instrumentdaten.

Die Instrumentnamen-Liste

Die Liste mit den Instrumentnamen umfaßt n Einträge a 18 Byte. Jeder Eintrag wird gemäß folgender Tabelle strukturiert.

Bytes	Bemerkungen
2	Instrument Index
1	Instrument Daten existieren (0 = Nein, 1 = Ja)
14+1	Instrumentname

Tabelle 77.2 Eintrag in der Liste der Instrumentnamen

Das erste Wort enthält den Index für das Instrument. Daraus läßt sich der Beginn des zugehörigen Datenbereiches in der Datentabelle berechnen:

```
Offset=Startadresse+28*Index
```

Das folgende Byte gibt an, ob die Datentabelle Werte für das Instrument enthält (1 = Yes). Der Instrumentenname ist als ASCIIZ-String abzulegen und darf bis zu 14 Zeichen umfassen.

Die Instrumentdaten-Liste

Die Liste mit den Instrumentdaten umfaßt n Einträge à 33 Byte. Jeder Eintrag wird gemäß folgender Tabelle strukturiert.

Bytes	Bemerkungen
1	Instrument Typ (0 Melody, 1 Rhythmus)
1	Instrument Nummer (bei Rhythmus)
29	Instrument Daten
1	Modulator Typ Sinuswelle
1	Carrier Typ Sinuswelle

Tabelle 77.3 Eintrag im Bereich der Instrumentdaten

Das erste Byte enthält den Instrumententyp (0 melodisch, 1 rhythmisch). Bei rhythmischen Instrumenten steht im nächsten Byte die Instrumentennummer. Der Bereich mit den Instrumentendaten umfaß 29 Byte, die Struktur ist mir aber nicht bekannt. Die letzten beiden Byte eines Eintrages definieren den Typ des Modulators und des Trägers.

79 Das AMIGA IFF-Format (IFF)

Dieses Format wurde ebenfalls für den Amiga Computer definiert. Eine Verbreitung für andere Plattformen ist selten.

Das IFF-Format dient auf diesen Rechnern zur Speicherung von Grafiken, Texten und Sound. Auf dem PC wurde im wesentlichen der Grafikteil übernommen.

Das IFF-Format besteht aus einem Header, gefolgt von einzelnen CHUNKs. Diese CHUNKs bestimmen den Inhalt der verschiedenen Daten. Zur Speicherung von Sounds werden die 8SVX-CHUNKs benutzt. Eine Beschreibung des IFF-Formats mit den betreffenden CHUNKS finden Sie in /1/ im Bereich der Grafikformate.

Das Audio IFF-Format (AIFF)

Hierbei handelt es sich um eine Variante des IFF-Formats, welches auf Apple-Computern verwendet wird. Die AIFC-Spezifikation (auch als AIFF-C bezeichnet) ist eine erweiterte Variante, die komprimierte Daten speichern kann. Eine Beschreibung der AIFF-CHUNKs finden Sie ebenfalls im Kapitel mit der Beschreibung des IFF-Formats.

84 Hewlett Packard Printer Communication Language (PCL)

Mit der Einführung des LaserJet II definierte die Firma Hewlett Packard eine neue Sprache zur Kommunikation mit zukünftigen Druckern. Diese PCL-Sprache setzte sich recht schnell durch und wird mittlerweile von vielen anderen Drucker emuliert. Der Befehlssatz der Sprache erlaubt – ähnlich wie PostScript – die Ausgabe von Grafik und verschiedenen Schriftarten. Beim LaserJet II wurde noch die Version PCL 4 benutzt, mit dem LaserJet III wurde dann PCL 5 eingeführt. Die nachfolgend beschriebenen Steuersequenzen beziehen sich auf diese neuere Version der Sprache.

PCL-Befehle werden in verschiedenen Gruppen zusammengefaßt, wobei jeder Befehl mit einem ESC-Zeichen (1BH) beginnt, dem mehrere Parameter folgen. Dieses Zeichen wird nachfolgend als Esc dargestellt. Die in eckigen Klammern [] stehenden Zahlen geben die gleiche Sequenz in Hexadezimalnotation wieder.

Befehle für einen Druckauftrag

Mit diesen Sequenzen wird ein kompletter Druckauftrag abgewickelt.

Befehl

Drucker Reset

Format

EscE [1B 45]

Funktion

setzt den Drucker auf die Standardeinstellung zurück.

Befehl

Anzahl Kopien

Format

Esc&l#X [1B 26 6C #...# 58]

Funktion

spezifiziert die Anzahl von Kopien, die auf einer Seite auszugeben sind. Das Zeichen »#« dient als Platzhalter und enthält die jeweilige Nummer der Kopien (1-99).

Befehl

Positionierung logische Seite in Querrichtung

Format

Esc&l#U [1B 26 6C #...# 55]

Funktion

dreht die Druckausgabe so, daß die X-Achse im Hochformat liegt. Der Platzhalter # enthält die Zahl der Punkte (1/720 Zoll), um die das Druckbild verschoben werden soll.

Befehl

Positionierung logische Seite in Längsrichtung

Format

Esc&l#Z [1B 26 6C #...# 5A]

Funktion

dreht die Druckausgabe so, daß die Y-Achse im Hochformat liegt. Der Platzhalter # enthält die Zahl der Punkte (1/720 Zoll), um die das Druckbild verschoben werden soll.

Seitenbeschreibungsbefehle

Diese Gruppe umfaßt Befehle zur Definition der Ausgabe einer Seite.

Befehl

Druck-(Seiten-)Format

Format

Esc&l#A [1B 26 6C #...# 41]

Funktion

gibt das Seitenformat und damit die Größe des Papiers an. Der Platzhalter # nimmt den Code für das Seitenformat auf. Für die definierten Codes muß jeweils die entsprechende Papierkassette verwendet werden:

Code	Seitenformat
Papierformen	
1	Executive (7 1/4 x 10 1/2 Zoll)
2	Letter (8 1/2 x 11 Zoll)
3	Legal (8 1/2 x 14 Zoll)
26	DIN A4 (210 x 297 mm)
Umschlagsformen	

Tabelle 84.1 Seitenformate

Code	Seitenformat
80	Monarch (3 7/8 x 7 1/2 Zoll)
81	COM-10 (4 1/8 x 9 1/2 Zoll)
90	International DL (110 mm x 220mm)
90	International C5 (162 mm x 229mm)

Tabelle 84.1 Seitenformate

Befehl

Papierquelle

Format

Esc&l#H [1B 26 6C #...# 48]

Funktion

gibt die Papierquelle für den Einzug an. Der Parameter # steht dabei für den Code der Quelle. Folgende Codes sind definiert:

Code	Quelle
0	aktuelle Seite ausdrucken
1	Papier aus Mehrzweckkassette einziehen
2	manuell zugeführtes Papier einziehen
3	Umschlag einziehen

Tabelle 84.2 Papierzufuhr

Befehl

Seitenlänge

Format

Esc&l#P [1B 26 6C #...# 50]

Funktion

gibt die Länge einer Seite in Zeilen an. Der Parameter # nimmt dabei die Zeilenzahl auf.

Befehl

Ausrichtung Druck

Format

Esc&l#O [1B 26 6C #...# 4F]

Funktion

gibt über den Parameter # die Ausrichtung des Drucks an. Die definierten Codes finden Sie in der nachfolgenden Tabelle.

Code	Druckrichtung
0	Hochformat
1	Querformat
2	umgekehrtes Hochformat
3	umgekehrtes Querformat

Tabelle 84.3 Ausrichtung beim Drucken

Befehl

Druckrichtung

Format

```
Esc&a#P      [1B 26 61 #...# 50]
```

Funktion

gibt die Ausrichtung des Drucks bezogen auf die X-Achse an. Der Parameter # spezifiziert dabei den Winkel in 90-Grad-Schritten; damit lassen sich senkrechte Schriften ausgeben.

Befehl

Oberer Rand

Format

```
Esc&l#Esc      [1B 26 6C #...# 45]
```

Funktion

legt über den Parameter # den oberen Rand fest; # definiert die Anzahl der freibleibenden Zeilen.

Befehl

Textlänge

Format

```
Esc&l#F      [1B 26 6C #...# 46]
```

Funktion

legt über den Parameter # die Länge des zu druckenden Textes in Zeilen fest.

Befehl

Linker Rand

Format

Esc&a#L [1B 26 61 #...# 4C]

Funktion

legt über den Parameter # den linken Rand fest – in # steht die Anzahl der Spalten, die am linken Rand frei bleiben sollen.

Befehl

Rechter Rand

Format

Esc&a#M [1B 26 61 #...# 4D]

Funktion

legt über den Parameter # den rechten Rand fest. In # steht die Spaltennummer, an der der Text abgeschnitten wird.

Befehl

Seitliche Ränder löschen

Format

Esc9 [1B 39]

Funktion

setzt die Randeinstellung auf die Standardwerte zurück.

Befehl

Perforation überspringen

Format

Esc&l#L [1B 26 6C #...# 4C]

Funktion

legt über den Parameter # fest, ob der Ausdruck am Seitenende unterbrochen oder auf der folgenden Seite fortgesetzt wird. Bei # gleich 0 ist der Modus aktiv, und der Ausdruck erfolgt über Blattgrenzen. Mit dem Wert 1 wird der Modus abgeschaltet.

Befehl

Horizontaler Spaltenabstand

Format

Esc&k#H [1B 26 6B #...# 4B]

Funktion

legt über den Parameter # den horizontalen Abstand zwischen zwei Spalten fest. # gibt den Abstand in 1/120 Zoll an, wobei durchaus vier Stellen erlaubt sind.

Befehl

Vertikaler Zeilenabstand

Format

Esc&l#C [1B 26 6C #...# 43]

Funktion

legt über den Parameter # den vertikalen Abstand zwischen zwei Zeilen fest. Der Parameter gibt den Abstand in 1/48 Zoll an, wobei durchaus vier Stellen erlaubt sind.

Befehl

Zeilen pro Zoll

Format

Esc&l#D [1B 26 6C #...# 44]

Funktion

legt über den Parameter # die Zahl der Zeilen pro Zoll fest. Für den Parameter sind die Werte 1, 2, 3, 4, 6, 8, 12, 16, 24 und 48 Zeilen/Zoll zulässig.

Cursorsteuerung

Die Gruppe enthält Befehle zur Positionierung des Ausgabecursors.

Befehl

Vertikal

Format

Esc&a#R [1B 26 61 #...# 52]

Funktion

verschiebt den Cursor auf Zeile #. Soll die Verschiebung in Punkten (1/300 Zoll) angegeben werden, gilt folgende Sequenz:

Esc*p#Y [1B 2A 70 #...# 59]

Der Parameter # gibt dabei die Zahl der Punkte an. Mit der Sequenz

Esc&a#V [1B 26 61 #...# 56]

wird die Verschiebung in Dezimalpunkten (1/720 Zoll) angegeben.

Befehl

Horizontal

Format

Esc&a#C [1B 26 61 #...# 43]

Funktion

verschiebt den Cursor auf Spalte #. Soll die Verschiebung in Punkten (1/300 Zoll) angegeben werden, gilt folgende Sequenz:

Esc*p#X [1B 2A 70 #...# 58]

wobei der Parameter # die Zahl der Punkte angibt. Mit der Sequenz:

Esc&a#H [1B 26 61 #...# 48]

wird die Verschiebung in Dezimalpunkten (1/720 Zoll) angegeben.

Befehl

Halber Zeilenvorschub

Format

Esc= [1B 3D]

Funktion

führt einen Vorschub um eine halbe Zeile aus. Die Steuerzeichen CR, LF, Space, Tab, BS, FF verursachen ebenfalls eine Cursorbewegung um Zeilen oder Spalten.

Befehl

Cursorposition

Format

Esc&f#S [1B 26 66 # 53]

Funktion

legt über den Parameter # fest, ob der Cursor gelesen *(# = 1)* oder gesichert wird (# = *0*).

Schriftauswahl

In dieser Befehlsgruppe lassen sich einzelne Schriftarten selektieren.

Befehl

Schriftart

Format

Esc(## [1B 28 #...#]

Funktion

legt über den Parameter ## die Schriftart fest, wobei ## für zwei ASCII-Zeichen steht. Der PCL-Modus definiert zur Zeit folgende Schriftarten:

Code	Schrift
0D	ISO 60: Norwegen 1
1D	ISO 61: Norwegen 2 (*)
1E	ISO 4: Großbritannien
0F	ISO 25: Frankreich (*)
1F	ISO 69: Frankreich
0G	HP Deutsch (*)
1G	ISO 21: Deutsch
0I	ISO 15: Italien
0K	ISO 14: JIS ASCII (*)
2K	ISO 57: China (*)
0N	ECMA 94: Latin
0S	ISO 11: Schweden
1S	HP Spanien (*)
2S	ISO 17: Spanien
3S	ISO 10: Schweden (*)
4S	ISO 16: Portugal (*)
5S	ISO 84: Portugal (*)

Tabelle 84.4 Schriftarten

Code	Schrift
6S	ISO 85: Spanien (*)
0U	ISO 6: ASCII
2U	ISO 2: IRV (*)
8U	HP Roman8
10U	PC 8
11U	PC 8 (D/N)
12U	PC 850

Tabelle 84.4 Schriftarten

Die mit Sternchen (*) markierten Schriften sollten nicht weiter verwendet werden, da ihre Bedeutung abnimmt.

Befehl

Primärabstand

Format

```
Esc(s#P          [1B 28 73 #..# 50]
```

Funktion

legt über den Parameter # den Abstand der Zeichen fest:

```
# = 0  fixed
# = 1  proportional
```

Befehl

Primäre Zeichendichte

Format

```
Esc(s#H          [1B 28 73 #...# 48]
```

Funktion

legt über den Parameter # die Anzahl der Zeichen pro Zoll fest.

Befehl

Zeichendichte einstellen

Format

```
Esc&k#S          [1B 26 6B #...# 53]
```

Funktion

stellt die Zeichendichte über den Parameter # ein. Folgende Werte sind zulässig:

Code	Dichte
0	10.0
2	komprimiert (16,5)
4	Elite (12,0)

Tabelle 84.5 Einstellung der Zeichendichte

Befehl

Primäre Zeichengröße

Format

```
Esc(s#V          [1B 28 73 #...# 56]
```

Funktion

legt über den Parameter # die Größe eines Zeichens in *Pica*-Punkt fest.

Befehl

Schriftlage

Format

```
Esc(s#S          [1B 28 73 #...# 53]
```

Funktion

legt über den Parameter # die Lage der Schrift fest. Es gelten die Belegungen:

```
# = 0  geradestehend
# = 1  Kursivschrift
```

Befehl

Strichstärke Primärschrift

Format

```
Esc(s#B          [1B 28 73 #...# 42]
```

Funktion

Der Befehl legt über den Parameter # die Strichstärke fest. Für # sind die oben aufgelisteten Codes zulässig. Das Minuszeichen für die dünnen Schriften ist ebenfalls als Code anzugeben.

Code	Strichstärke
-7	Ultrafein
-6	Extrafein
-5	Fein
-4	Extramager
-3	Mager
-1	Dreiviertelmager
-2	Halbmager
0	Normal
1	Halbfett
2	Dreiviertelfett
3	Fett
4	Extrafett
5	Schwarz
6	Extraschwarz
7	Ultraschwarz

Tabelle 84.6 Strichstärken der Primärschrift

Befehl

Schrifttyp

Format

```
Esc(s#T          [1B 28 73 #...# 54]
```

Funktion

legt über den Parameter # den Schrifttyp fest. Für # sind folgende Codes zulässig:

Code	Schrifttyp
0	Lineprinter
1	Pica
2	Elite

Tabelle 84.7 Schrifttypen

Code	Schrifttyp
3	Courier
4	Helvetica
5	Times Roman
6	Gothic
7	Script
8	Prestige

Tabelle 84.7 Schrifttypen

Befehl

Standard Schrift

Format

```
Esc(s#@          [1B 28 33 40]
```

Funktion

In diesem Format legt der Befehl die primäre Schrift fest; die sekundäre Schrift wird über das folgende Format bestimmt:

```
Esc)s#@          [1B 29 33 40]
```

Befehl

Transparente Druckdaten

Format

```
Esc&p#X[Daten]    [1B 26 70 #...# 58 ...]
```

Funktion

legt die vom Drucker zu interpretierenden Zeichen fest. Der Parameter # gibt die Zahl der im Feld *[Daten]* folgenden Zeichen an.

Befehl

Unterstreichen Ein/Aus

Format

```
Esc&d0D          [1B 26 64 30 44]
```

Funktion

Das Format setzt den Modus *Unterstreichen EIN* fest. Für *Unterstreichen EIN angepaßt* ist folgendes Format vorgesehen:

Esc&d3D [1B 26 64 33 44]

In diesem Modus werden Leerzeichen nicht unterstrichen. Mit

Esc&d@ [1B 26 64 40]

wird der Modus *Unterstreichen* ausgeschaltet.

Schriftverwaltung

In dieser Gruppe finden sich verschiedene Befehle zum Laden und Löschen von Schriften.

Befehl

Schriftkennung zuweisen

Format

Esc*c#D [1B 2A 63 #...# 44]

Funktion

Der Parameter # steht für die Kennummer, die zwischen 0 und 32767 liegen darf. Alle nachfolgenden Schriftverwaltungsbefehle beziehen sich auf diese Nummer.

Befehl

Steuerung von Schriftzeichen

Format

Esc*c#F [1B 2A 63 #...# 46]

Funktion

Der Parameter # steht für den jeweiligen Code mit einer der aufgelisteten Bedeutungen:

Code	Bedeutung
0	alle Schriften löschen
1	alle temporären Schriften löschen
2	Schrift mit zuletzt angegebener Kennung löschen
3	zuletzt angegebenes Zeichen löschen
4	Schrift temporär setzen
5	Schrift permanent setzen
6	aktuelle Schrift als temporär zuweisen/kopieren

Tabelle 84.8 Steuerung von Schriftzeichen

Befehl

Schrift auswählen

Format

Esc(#X [1B 28 #...# 58]

Funktion

Das Format macht die Schrift mit der Kennummer # zur primären Schrift. Zur sekundären Schrift wird sie mit diesem Format:

Esc)#X [1B 29 #...# 58]

Erstellung ladbarer Schriften

Mit dieser Befehlsgruppe lassen sich eigene Schriften laden.

Befehl

Schrift Descriptor

Format

Esc)s#W[Daten] [1B 29 73 #...# 57 Daten]

Funktion

legt über den Parameter # die Länge des nachfolgenden Schriftdescriptors fest. An den Befehl schließen sich *n* Byte mit der Schriftbeschreibung an.

Befehl

Zeichen Code

Format

Esc*c#Esc [1B 2A 63 #...# 45]

Funktion

legt über den Parameter # den Code des ASCII-Zeichens fest, dem das nächste Zeichen zugeordnet werden soll.

Befehl

Zeichen laden

Format

Esc(s#W[Daten] [1B 28 73 #...# 57 Daten]

Funktion

legt über den Parameter # die Länge des nachfolgenden Zeichendescriptors fest. An den Befehl schließen sich *n* Byte mit der Zeichenbeschreibung an.

Grafikbefehle

In dieser Gruppe finden sich die Befehle für die Ausgabe von Grafiken im Raster- und Vektorformat; der HP-GL/2-Zeichensatz wird dabei unterstützt.

Befehl

HP-GL/2 Modus

Format

Esc%0B [1B 25 30 42]

Funktion

legt fest, daß die letzte Stiftposition des HP-GL/2-Modus benutzt wird. Im folgenden Format bestimmt er, daß die Grafikausgabe an der aktuellen PCL-Ausgabeposition beginnen soll:

Esc%1B [1B 25 31 42]

Befehl

HP-GL/2 Plotbreite

Format

Esc*c#K [1B 2A 63 #...# 4B]

Funktion

stellt über den Parameter # die Strichbreite in Zoll ein.

Befehl

HP-GL/2 Plotlänge

Format

Esc*c#L [1B 2A 63 #...# 4C]

Funktion

stellt über den Parameter # die Strichlänge in Zoll ein.

Befehl

Referenzpunkt im Grafikbereich

Format

Esc*c0T [1B 2A 63 30 54]

Funktion

setzt die aktuelle Position als Referenzposition fest.

Befehl

Breite des Grafikbereichs

Format

Esc*c#X [1B 2A 63 #...# 58]

Funktion

stellt über den Parameter # die Breite des Grafikbereichs in Dezimalpunkten ein.

Befehl

Höhe des Grafikbereichs

Format

Esc*c#Y [1B 2A 63 #...# 59]

Funktion

stellt über den Parameter # die Höhe des Grafikbereichs in Dezimalpunkten ein.

Befehl

Auflösung Rastergrafik

Format

Esc*t###R [1B 2A 74 #...# 52]

Funktion

legt die Auflösung der Rastergrafik fest. Für die Platzhalter stehen je nach Auflösung folgende Codes:

Codes (H)	Auflösung
37 35	75 Punkte pro Zoll

Tabelle 84.9 Auflösungen für Rastergrafik

Codes (H)	Auflösung
31 30 30	100 Punkte pro Zoll
31 35 30	150 Punkte pro Zoll
33 30 30	300 Punkte pro Zoll

Tabelle 84.9 Auflösungen für Rastergrafik

Die Werte sind dabei in Hexnotation angegeben und werden jeweils in die Sequenz eingefügt (z.B.: 75 Zoll = Esc*t75R).

Befehl

Darstellung der Rastergrafik

Format

```
Esc*r0F      [1B 2A 72 30 46]
```

Funktion

in diesem Format dreht sich das Bild bei der Ausgabe, während es mit der folgenden Sequenz im Querformat des Druckers ausgegeben wird:

```
Esc*r3F      [1B 2A 72 30 46]
```

Befehl

Beginn Rastergrafik

Format

```
Esc*r0A      [1B 2A 72 30 41]
```

Funktion

In diesem Format wird der linke Rand an der Spalte 0 festgelegt, während diese Sequenz den linken Rand an der aktuellen Position fixiert:

```
Esc*r1A      [1B 2A 72 31 41]
```

Befehl

Datenkomprimierung

Format

```
Esc*b#M      [1B 2A 62 #...# 4D]
```

Funktion

komprimiert die über den Parameter # definierten Daten. Je nach eingestelltem Wert für # variiert die Komprimierungsart:

Code	Komprimierung
0	unkodiert
1	Lauflängenkodierung
2	TIFF-Kodierung
3	Delta Row

Tabelle 84.10 Komprimierungsarten

Befehl

Übertragung

Format

```
Esc*b#W[Daten] [1B 2A 62 #...# 57 Daten]
```

Funktion

überträgt die Daten der Rastergrafik. Der Parameter # gibt die Zahl der Daten an, die sich an den Befehl anschließen.

Befehl

Ende Rastergrafik

Format

```
Esc*rB      [1B 2A 72 42]
```

Funktion

markiert das Ende der Rastergrafik.

Befehl

Rasterhöhe

Format

```
Esc*r#T     [1B 2A 72 #...# 54]
```

Funktion

gibt über den Parameter # die Zahl der Rasterzeilen an.

Befehl

Rasterbreite

Format

Esc*r#S [1B 2A 72 #...# 53]

Funktion

gibt über den Parameter # die Zahl der Rasterpixel pro Zeile an.

Druckmodell

In dieser Befehlsgruppe finden sich einige Anweisungen, die das jeweilige Druckmodell (Schwärzung, Transparenz) einstellen.

Befehl

Muster auswählen

Format

Esc*v#T [1B 2A 76 #...# 54]

Funktion

legt über den Parameter # die Druckfarbe fest. Dabei gilt folgende Zuordnung:

Code	Zuordnung
0	Vollton Schwarz
1	Vollton Weiß
2	Grautönungsmuster
3	Kreuzschraffierung

Tabelle 84.11 Musterauswahl

Befehl

Quelle auswählen

Format

Esc*v#N [1B 2A 76 #...# 4E]

Funktion

legt über den Parameter # den Deckmodus fest. Dabei gilt folgende Zuordnung:

Code	Zuordnung
0	transparent
1	deckend

Tabelle 84.12 Deckmodus

Befehl

Muster auswählen

Format

Esc*v#O [1B 2A 76 #...# 4F]

Funktion

legt über den Parameter # den Deckmodus des Musters fest. Dabei gilt folgende Zuordnung:

Code	Zuordnung
0	transparent
1	deckend

Tabelle 84.13 Deckmodus des Musters

Befehl

Breite Rechteckfläche

Format

Esc*c#A [1B 2A 63 #...# 41]

Funktion

gibt die Breite einer zu füllenden Rechteckfläche in Punkten an. Mit folgendem Befehl wird die Breite in Dezimalpunkten angegeben.

Esc*c#H [1B 2A 63 #...# 48]

Befehl

Höhe Rechteckfläche

Format

Esc*c#B [1B 2A 63 #...# 42]

Funktion

gibt die Höhe einer zu füllenden Rechteckfläche in Punkten an. In diesem Format wird die Höhe in Dezimalpunkten angegeben:

Esc*c#V [1B 2A 63 #...# 56]

Befehl

Rechteckfläche ausfüllen

Format

Esc*c#P [1B 2A 63 #...# 50]

Funktion

gibt im Parameter # das Füllmuster für die Rechteckfläche an:

Code	Füllmuster
0	Vollton Schwarz
1	Vollton Weiß
2	Grautönung
3	Schraffierung
4	Benutzermuster
5	aktuelles Muster

Tabelle 84.14 Füllmuster für Rechteckflächen

Befehl

Kennummer der Musters

Format

Esc*c#G [1B 2A 63 #...# 47]

Funktion

weist einem Muster über den Parameter # einen Mustertyp oder die Schwärzung in Prozent zu. Für Graustufen gilt folgende Kodierung des Parameters:

Code	Grautönung
2	2 %
10	10 %
15	15 %
30	30 %

Tabelle 84.15 Graustufenkodierung

Code	Grautönung
45	45 %
70	70 %
90	90 %
100	100 %

Tabelle 84.15 Graustufenkodierung

Die Werte sind als Zahl (z.B. *100 % = 1B 2A 63 31 30 30 47*) in die Sequenz einzusetzen. Für das Muster gilt folgende Kodierung:

Code	Muster
1	horizontale Linien
2	vertikale Linien
3	diagonale Linien
4	Gitter
5	diagonales Gitter

Tabelle 84.16 Musterkodierung

Durch die Art des Codes kann demnach das Muster bestimmt werden.

Makros

In dieser Gruppe sind die Befehle zur Makrodefinition zusammengefaßt.

Befehl

Makrokennung

Format

Esc&f#Y [1B 26 66 #...# 59]

Funktion

definiert eine Kennummer für Makros; # gibt dabei die Nummern zwischen 0 und 32767 an.

Befehl

Makrosteuerung

Format

Esc&f#X [1B 26 66 #...# 58]

Funktion

definiert die Steuerung von Makros, wobei der Parameter # einen der folgenden Modi für die Steuerung angibt:

Code	Modus
0	Beginn Makrodefinition
1	Ende Makrodefinition
2	Makro ausführen
3	Makro aufrufen
4	Überlagern aktivieren
5	Überlagern deaktivieren
6	Makro löschen
7	alle temporären Makros löschen
8	Makro mit zuletzt angegebener Kennung löschen
9	Makro temporär setzen
10	Makro permanent setzen

Tabelle 84.17 Makrosteuerung

Die einzelnen Ziffern der Codes müssen für # als Hexwerte eingesetzt werden (z.B. *Code 10 = 31 30*).

Programmierhinweise

Hier sind weitere Befehle zur Steuerung der Druckeranzeige und des Zeilenumbruchs abgelegt. Die Befehle können zur Fehlersuche benutzt werden.

Befehl

Anzeigefunktion

Format

```
EscY        [1B 59]
EscZ        [1B 5A]
```

Funktion

Die erste Befehlssequenz schaltet die Anzeige aus, die zweite schaltet sie wieder ein.

Befehl

Automatischer Zeilenumbruch

Format

Esc&s0C [1B 26 73 30 43]
Esc&s1C [1B 26 73 31 43]

Funktion

Mit den beiden Sequenzen wird der automatische Zeilenumbruch ein- und wieder abgeschaltet.

PCL-Zugriffserweiterung

Mit den Befehlen dieser Gruppe lassen sich im HP-GL/2-Modus PCL-Befehle aktivieren. Diese Befehle sind im Abschnitt Dual Context Extension im Kapitel über die HP-GL/2-Befehle beschrieben. Der HP-GL/2-Modus läßt sich über die Sequenz Esc%0B aktivieren.

Diverse Windows-Dateiformate

Windows 3.x Kalender-Format (CAL)

Das Windows Cardfile-Format (WINDOWS 3.x)

Clipboard Format (CLP)

Die Windows 3.x Gruppendateien (GRP)

86 Windows 3.x Kalender-Format (CAL)

Das Windows Kalender-Programm besitzt ein eigenes Format zur Speicherung der Daten für Termine und Notizen. Die folgende Beschreibung befaßt sich mit der Version für Windows 3.x.

Der Header

Der Header enthält die Signatur, die die Datei als gültigen Kalenderfile ausweist, sowie Informationen über die Zahl der Einträge. Die folgende Tabelle 86.1 gibt die Struktur des Headers wieder.

Offset	Byte	Bedeutung
00H	8	Signatur (B5 A2 B0 B3 B3 B0 A2 B5H)
08H	2	Datumseinträge
0AH	2	Alarm (in Minuten)
0CH	2	Sound (Boolean)
0EH	2	Intervall zwischen Terminen
10H	2	Intervall in Minuten
12H	2	24-Std-Anzeige (Boolean)
14H	2	Startzeit Tagesanzeige
16H	64	reserviert

Tabelle 86.1 Der CAL-Header

Die ersten 8 Bytes enthalten die Signatur, die eine gültige Kalender-Datei identifiziert. Diese Sequenz setzt sich aus der Addition der Buchstabencodes für folgende Zeichensequenz zusammen:

'C' + 'r' = b5
'A' + 'a' = a2
'L' + 'd' = b0
'E' + 'n' = b3
'N' + 'e' = b3
'D' + 'l' = b0
'A' + 'a' = a2
'R' + 'c' = b5

Das nächste Wort enthält einen Zähler mit der Zahl der Datumseinträge in der Datei. Die folgenden 12 Byte werden in sechs Felder aufgeteilt und enthalten globale Einträge für die Datei. Das erste Wort definiert die Alarmzeit (early ring) in Minuten. Daran schließt sich eine logische Variable (fSound) an, die definiert, ob der Alarm hörbar sein soll. Das Wort ab Offset 10H definiert die Abstufung des Terminkalenders:

0 = 15 Minuten
1 = 30 Minuten
2 = 60 Minuten

Das folgende Wort gibt das Zeitintervall in Minuten an. Die Variable (f24HourFormat) definiert bei der Einstellung »true«, daß die Uhrzeit im 24-Stunden-Format angezeigt wird. Steht in der Zelle eine 0, wird eine 12-Stunden-Anzeige benutzt. Das letzte belegte Wort definiert die Startzeit in der Tagesanzeige, die in Minuten nach Mitternacht in der ersten Anzeige erscheint. Die restlichen 64 Byte sind reserviert.

Der Datenbereich

An den Kopf schließt sich der Datenbereich mit den Termineinträgen an. Dieser Datenbereich besteht aus einem Feld, wobei jeder Eintrag einen kompletten Tag umfaßt. Die Zahl der Einträge in diesem Array wird durch das Feld »cDateDescriptors« im Header beschrieben. Jedes Element in dem Array besteht aus 12 Bytes. Deren Bedeutung sehen Sie in Tabelle 86.2.

Offset	Byte	Bedeutung
00H	2	Datum (in Tagen seit 1.1.1980)
02H	2	Marke für Datum 128 : Rechteck 256 : Klammern 512 : Kreis 1024 : Kreuz 2048 : unterstrichen
04H	2	Zahl der Alarme des Tages
06H	2	Offset in 64 Byte Block mit den Tagesinformationen
08H	2	reserviert (FFFFH)
0AH	2	reserviert (FFFFH)

Tabelle 86.2 Aufbau des Datenbereichs

Das erste Wort enthält die Tageszahl als Integerwert, wobei dieser die Zahl der Tage seit dem 1.1.1980 angibt. Daran schließt sich ein Flag an, welches spezifiziert, wie das Datum angezeigt werden soll. Ab Offset 04H steht die Zahl der Einträge pro Tag, an denen ein Alarm auszulösen ist. Das letzte Wort ab Offset 06H bezeichnet ein Wort, welches einen Zeiger in einen 64-Byte-Block mit den Informationen über den Tag enthält.

In diesem Wort werden nur die unteren 15 Bits genutzt, daß oberste Bit ist immer 0. Findet sich in dem Wort der Wert 6, errechnet sich der Offset in den Block mit den Tagesinformationen zu 6*64 (Byte).

Die restlichen beiden Worte sind reserviert und mit FFFFH belegt.

Bereich mit den Tagesinformationen

Alle Informationen über einen Tag werden nach dem Array mit den Datumsbeschreibungen an 64-Byte-Grenzen gespeichert. Die Informationen besitzen dabei folgendes Format:

Offset	Byte	Bedeutung
00H	2	reserviert (muß 0 sein)
02H	2	Tage seit 1.1.1990
04H	2	reserviert (muß 1 sein)
06H	2	Zahl der Bytes in Notiz
08H	2	Zahl der Termine
0AH	x	Textinformation (Notiz)
..H	n*x	Feld mit Termineinträgen

Tabelle 86.3 Aufbau des Bereichs mit den Tagesinformationen

Die Zahl ab Offset 06H gibt die Länge der Textinformationen an. Dahinter schließt sich ein Feld mit mehreren Einträgen für die Termine an. Die Zahl der Einträge steht ab Offset 08H. Das Array mit den Einträgen für die einzelnen Termine besitzt folgenden Aufbau:

Offset	Byte	Bedeutung
00H	1	Zahl der Bytes im Eintrag
01H	1	Flags Bit 0 = 1 Alarm, Bit 1 = 1 spezielle Zeit
02H	2	Zeit (in Minuten seit 0:00 Uhr)
04H	x	ASCIIZ-String mit Termineintrag

Tabelle 86.4 Belegung des Terminbereichs

Der erste Wert gibt die Zahl der Folgebytes an. Damit läßt sich jederzeit die Lage des folgenden Eintrages bestimmen. Im Textfeld findet sich dann die Bemerkung des Benutzers zum jeweiligen Termin.

87 Das Windows Cardfile-Format (WINDOWS 3.x)

Das Cardfile-Format von Windows 3.x ist recht einfach aufgebaut. Der Header umfaßt 5 Byte und besitzt folgende Struktur:

Offset	Byte	Bedeutung
00H	3	Signatur (»MGC«)
03H	2	Zahl der Karteneinträge

Tabelle 87.1 CRD-File-Header

An diesen Header schließt sich ein Textbereich (Indexline) an, der die Kopftexte der einzelnen Karten enthält. Der erste Eintrag beginnt ab Offset 05H. Die folgenden Einträge liegen dann immer 34 Byte vom Beginn des vorhergehenden Eintrages entfernt. Jede Zeile im Indexbereich mit den Kopftexten besitzt folgendes Format:

Offset	Byte	Bedeutung
00H	6	Nullbytes (00 00 00 00 00 00)
06H	4	Offset Datenbereich Karte
0AH	1	Flagbyte (00)
0BH	22	Kopftext der Karte
2DH	1	Endebyte (00)

Tabelle 87.2 Indexbereich (CRD-File)

Der Zeiger gibt den Offset vom Beginn der Datei zum ersten Byte des Datenbereichs der Karte an. Dieser Datenbereich schließt sich an den Indexbereich mit den Kopftexten der Karten an.

Der Datenbereich der Karten kann Text und Grafik enthalten. Es werden daher folgende Unterscheidungen getroffen:

- Text
- Text und Grafik
- Grafik
- leere Karte

Abhängig von diesen Unterscheidungen gestaltet sich die Datenstruktur für die Karte.

Grafik & Text	Text	Grafik	Bemerkungen
0 – 1	0-1#	0 – 1	Länge der Grafik (Bitmap#)

Tabelle 87.3 Aufbau des Cardfile-Datenbereichs

Grafik & Text	Text	Grafik	Bemerkungen
2 – 3	*	2 – 3	Breite der Grafik (*)
4 – 5	*	4 – 5	Höhe der Grafik (*)
6 – 7	*	6 – 7	X_Koordinate der Grafik (*)
8 – 9	*	8 – 9	Y-Koordinate der Grafik (*)
A – x	*	A – x	Bitmap der Grafik (*)
x+1/x+2	2-3	x+1/x+2	Länge Texteintrag (#)
x+3-y	4-z	*	Text (*)

Tabelle 87.3 Aufbau des Cardfile-Datenbereichs

Die mit dem Zeichen (#) markierten Bytes sind 00H, falls keine Grafik oder kein Text auftritt. Der Wert X bestimmt sich aus der Länge der Bitmapgrafik + 9 Byte. Der Wert für y bestimmt sich aus x+2+Länge des Texteintrages. Die mit einem (*) markierten Bytes existieren nicht, falls kein Text oder keine Bitmap gespeichert ist. Der Wert für z ergibt sich aus 3 + Länge des Texteintrages.

Das erste Byte einer Karte wird im Header in der Indextabelle adressiert. Das Ende einer Karte wird nicht durch Nullbytes markiert.

Texte in der Karte werden im ASCII-Format gespeichert. Dabei wird jedes CR-Zeichen durch ein LF-Zeichen ergänzt. Das Cardfile-Format wurde in Windows 3.1 etwas erweitert, um OLE-Infos aufzunehmen. Die Formatbeschreibung ist mir zur Zeit jedoch nicht bekannt.

88 Clipboard Format (CLP)

Die Windows-Zwischenablage (Clipboard) speichert den Inhalt von kompletten Bildschirmen oder von Fenstern. Der Inhalt der Zwischenablage läßt sich durch andere Windows-Programme übernehmen oder in einer CPL-Datei sichern. Leider ist mir kein Fremdprogramm bekannt, das direkt die CLP-Dateien lesen kann. Deshalb möchte ich nachfolgend die Informationen zum CLP-Format vorstellen, soweit sie mir bekannt sind.

Die Datei beginnt mit einem 4-Byte-Header, der die Datei als Clipboard identifiziert.

Offset	Byte	Bedeutung
00H	2	File ID-Signatur (50 C3H)
02H	2	Formatcount
04H	2	Format ID
06H	4	Länge Datenbereich (Clipboard)
0AH	4	Offset (in Byte) zum Datenblock
0EH	79	Formatname als Text (privat)
...	..	ggf. Wiederholungen der Datenstruktur ab Offset 04H bis 0EH
...	..	Datenbereich Clipboard

Tabelle 88.1 Clipboard-FormatClipboard-Format

Das erste Wort enthält die Signatur, die ein gültige CLP-Datei markiert. Daran schließt sich ein Wort an, welches die Zahl der Datenblöcke im Clipboard definiert. In einer Datei können durchaus mehrere Datenblöcke mit unterschiedlichen Formaten gespeichert sein. An diesen Header schließen sich *n* Blöcke (n = Zahl in Formatcount) mit Angaben zu den Datenbereichen an. Das erste Wort eines solchen Blocks enthält einen Code, der das Format des Datenblocks beschreibt. Bisher sind die folgenden Datenformate definiert:

```
CF_TEXT1
CF_BITMAP2
CF_METAFILEPICT3
CF_SYLK4
CF_DIF5
CF_TIFF6
CF_OEMTEXT7
CF_DIB8
CF_PALETTE9
CF_OWNERDISPLAY80H
CF_DSPTEXT 81H
CF_DSPBITMAP82H
CF_DSPMETAFILEPICT83H
```

Ein Teil dieser Formate ist in den vorhergehenden Kapiteln des Buches beschrieben. Das nächste Feld enthält die Länge des zugehörigen Datenbereichs in Byte. Daran schließt sich ein Feld mit einem 4-Bit-Zeiger mit dem Offset vom Dateianfang auf das erste Byte des zugehörigen Datenbereichs an. Der letzte Eintrag in dem Block enthält einen 79 Byte langen Bereich für die Namen von privaten Clipboard-Formaten. Das folgende Bild 88.1 enthält einen Auszug aus einer CLP-Datei.

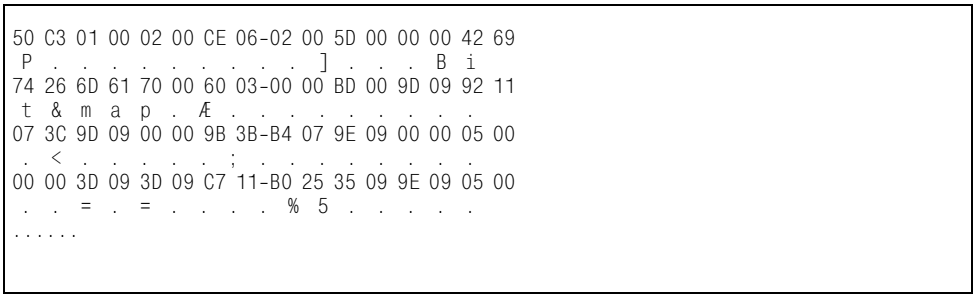


Abbildung 88.1 Auszug aus einer CLP-Datei

Dieses Bild enthält eine Bitmap, was an dem Eintrag im Namensfeld zu erkennen ist. An diesen Bereich schließen sich die Datenbereiche der Zwischenablage an. Deren Kodierung ist mir (bis auf die bereits beschriebenen Formate) nicht bekannt.

89 Die Windows 3.x Gruppendateien (GRP)

Der Programmanager von Windows 3.x verwaltet die Informationen, welche Programme zu einer Gruppe gehören, in eigenen Dateien mit der Extension GRP. Nachfolgend möchte ich den Aufbau der GRP-Dateien (soweit bekannt) beschreiben. Beachten Sie aber, daß alle Informationen durch Reverse Engineering gewonnen wurden und sich dadurch einige Ungenauigkeiten ergeben können.

Der Header

Offset	Byte	Bemerkungen
00H	4	File ID-Signatur 'PMCC'
04H	2	Checksumme
06H	2	Dateigröße GRP-File in Byte
08H	2	CmdShow-Flag
0AH	8	Koordinaten Group Window
12H	4	Koordinaten Group Window
16H	2	Pointer auf Group Name
18H	2	Horizontal Display Resolution
1AH	2	Vertikal Display Resolution
1CH	2	Bits per Pixel
1EH	2	Planes
20H	2	Einträge (Items)
22H	n*2	Array mit Item Struktur

Tabelle 89.1 Header einer GRP-Datei

Der Header einer Gruppendatei beginnt mit einer 4-Byte-Signatur ('PMCC'). Daran schließt sich eine Prüfsumme mit 2 Bytes an. Diese Prüfsumme errechnet sich aus der Differenz aller Worte (2 Byte) innerhalb der Datei. Ausnahme bildet das Wort mit der Prüfsumme (Offset 04H), welches nicht berücksichtigt werden darf. Weiterhin wird das letzte Byte der GRP-Datei ignoriert, falls die Länge ungerade ist (dann kann mit dem letzten Byte kein Wort gebildet werden). Die Berechung läßt sich leicht durch folgende Sequenz angeben:

```
         csum = 0
csum = csum û 1.Wort
csum = csum û 2.Wort
; Achtung 3. Wort ¬bergehen
csum = csum û 4.Wort
```

. . . . .

Ab Offset 06H findet sich die Größe der Group-Datei in Byte. Das folgende Word (CmdShow) definiert, wie der Programmanager die Gruppe anzeigen soll:

Value	Flag
00H	Hide
01H	Show normal
02H	Show minimized
03H	Show maximized
04H	Show noactivate
05H	Show
06H	Minimize
07H	Show Min & noactivate
08H	Show no activate
09H	Restore

Tabelle 89.2 CmdShow-Werte

Ab Offset 0AH finden sich die Koordinaten des Gruppenfensters (RECT-Struktur: X1,Y1,X2,Y2). Das folgende Feld enthält den Koordinatenpunkt (POINT-Struktur) der linken unteren Ecke des Gruppenfensters relativ zum *Parent-Window*.

Offset 16H enthält einen Zeiger auf den nullterminierten String mit dem Namen der Gruppe. Die folgenden beiden Felder definieren die Bildauflösung, auf die das Gruppen-Icon abgestimmt wurde. Ab Offset 1CH wird die Zahl der Bit pro Pixel für die Bitmap des Icons abgelegt. Daran schließt sich die Zahl der Farbebenen des Icons an.

Ab Offset 20H findet sich ein Wort mit der Zahl der Einträge in der folgenden Datenstruktur. Diese Datenstruktur beginnt ab Offset 22H und enthält die Einträge für die Programme der Gruppe. Für jeden Eintrag wird folgende Datenstruktur definiert:

Bytes	Bemerkungen
4	Koordinate (x,y) unten links des Icons in der Gruppe
2	Index für das Icon
2	Bytes in der Icon-Ressource
2	AND-Maske Bytes für das Icon
2	XOR-Maske Bytes für das Icon
2	Header Offset
2	AND-Pointer

Tabelle 89.3 Item Datenstruktur

Bytes	Bemerkungen
2	XOR-Pointer
2	Name Offset
2	Command Offset
2	Icon Pfad Offset

Tabelle 89.3 Item Datenstruktur

Der *Index* definiert die Nummer des Icons in einer EXE-Datei. Daran schließt sich die Zahl der Bytes des Icons in der Resourcendatei an. Die Felder mit der AND- und XOR-Maske enthalten die Zahl der Bytes in den betreffenden Masken. Der Header offset definiert den Abstand vom Beginn der Gruppendatei zu einem String mit dem Namen des Eintrages. Daran schließen sich wieder zwei Worte mit Zeigern auf die AND- und XOR-Masken an. Das nächste Wort definiert den Offset vom Beginn der Gruppe zu dem String mit dem Namen des Eintrag s(Programmname). Im folgenden Wort findet sich ein Zeiger vom Beginn der Programmgruppe auf den Namen der ausführbaren Programmdatei. Der Pfad zur ausführbaren Datei wird durch den Offset im folgenden Wort adressiert.

Die GRP-Datei besitzt für jeden Eintrag in der Programmgruppe einen solchen Datenbereich.

Index

C

D

E

F

G

H

I

P

Q

R

S

T

V

W

Z