

Inhalt

Vorwort 15

Teil 1: Tabellenkalkulation

<hr/> 1	Lotus 1-2-3-Format (WKS/WK1) 17
	Der Aufbau der WKS/WK1-Dateien 17
	Recordtypen in Lotus 1-2-3 (Versionen 1.1 bis 2.01) 18
<hr/> 2	LOTUS 1-2-3-Format (WK3) 53
	Die LOTUS 1-2-3-Recordtypen der Version 3.x 53
<hr/> 3	Binary Interchange Format (BIFF) 87
	Der BIFF-Recordaufbau der Versionen 2.0–8.0 87
	Indizierung in der Tabelle 91
	Die Recordtypen im BIFF2- bis BIFF8-Format 91
<hr/> 4	Quattro Pro 3.0-Dateiformat (WQ1) 219
	Der Recordaufbau 219
	Die Quattro Pro-Recordtypen 219
<hr/> 5	Symbolic Link-Format (SYLK) 279
	Die Basis-SYLK-Records 279
	SYLK-Formaterweiterungen für CHART 294
<hr/> 6	Data Interchange Format (DIF) 311
	Der Aufbau des DIF-Headers 311
	Die DIF-Datensatzstruktur 317
<hr/> 7	Super Data Interchange-Format (SDI) 323
	Der Header einer SDI-Datei 323
	Der Datenteil einer SDI-Datei 327

Teil 2: Textverarbeitung

<hr/>	8	Word 97-Dateiformat (DOC) 331
		Der Aufbau einer Word DOC-Datei 331
		Aufbau des Textbereichs 340
		Zeichen- und Absatzformatierung 343
<hr/>	9	WordPerfect-Format 345
		Der WordPerfect-Header (Version 5.0) 345
		Der WordPerfect-Datenbereich (5.0) 350
		Die 1-Byte-Steuercodes von 00H bis BFH 351
		Die Fixed Length Multibyte-Steuercodes von C0H bis CFH 354
		Variable Length Multibyte-Steuercodes von D0H bis FFH 357
		Subfunktionen zur Fontauswahl (Code D1H) 363
		Subfunktionen zur Gruppeneffinition (Code D2H) 364
		Set Group-Subfunktionen (Code D3H) 370
		Die Format-Group-Subfunktionen (Code D4H) 376
		Die Header/Footer-Group-Subfunktionen (Code D5H) 378
		Footnote/Endnote-Group-Subfunktionen (Code D6H) 380
		Generate Group-Subfunktionen (Code D7H) 381
		Display Group-Subfunktionen 387
		Miscellaneous Group 389
		Box Group 391
		Style Group 395
		WordPerfect-Header (Version 5.1) 397
		Der WordPerfect 5.1-Textbereich 400
		Die Fixed Length Multibyte-Steuercodes (Version 5.1) 401
		Die Variable Length Multibyte-Steuercodes (Version 5.1) 403
		Subfunktionen zur Fontauswahl (Code D1H) 407
		Subfunktionen zur Gruppeneffinition (Code D2H) 409
		Set Group-Subfunktionen (Code D3H, Version 5.1) 413
		Format-Group-Subfunktionen (Code D4H) (Version 5.1) 416
		Header/Footer-Group-Subfunktionen (Code D5H) 420
		Footnote/Endnote-Group-Subfunktionen (Code D6H) 420
		Generate Group-Subfunktionen (Code D7H) 420

Display Group-Subfunktionen (Code D8H) 421
Miscellaneous Group (Code D9H) 421
Box Group (Code DAH) 423
Table End of Line Codes Group 429
Table End of Pages Codes Group (ODDH) 431
Enhanced Merge Functions (Code DEH) 431

10 Rich Text Format (RTF Version 1.6) 437

Der Aufbau der RTF-Datei 438
Destination Control Words 440
Der Dokumentbereich 454
Verschiedene Kontrollwörter 512

Teil 3: Grafikformate

11 Das Adobe Illustrator File-Format (AI) 515

Die AI-Header-Comments 515

12 Das Adobe Photoshop-Format (PSD) 533

Der Photoshop-Header 533
Der Mode Data-Block 534
Der Resource Data-Block 535
Der Bilddatenbereich 535

13 AutoCAD Drawing Exchange Format (DXF) 537

Aufbau einer DXF-Datei 537
Der DXF-Header 548
DXF Table-Section 554
BLOCK-Abschnitt einer DXF-Datei 564
DXF Entities Section 566
AutoCAD Binary DXF 580
Das Drawing Exchange Binary-Format (DXB) 581

<hr/> 14	Das Autodesk Animator-Format (FLI) 583
	Der FLI-Header 583
	Die FLI-Frames 584
	Das Animator CEL- und PIC-Format 589
<hr/> 15	Das Autodesk 3D Studio-Format (FLC) 591
	Der FLC-Header 591
	Die FLC-Frames 593
<hr/> 16	Das Audio/Video Interleaved-Format (AVI) 599
	Die Resource Interchange File Format (RIFF)-Spezifikation 599
	Der AVI-Header-CHUNK (hdrl) 601
	Andere Daten-CHUNKs 609
<hr/> 17	Windows Bitmap-Format (BMP) 611
	Erweiterter Header in Windows 3.x 611
	Erweiterter Header in Windows NT 614
	Erweiterter BMP4-Header (Windows 95) 615
	Der Datenbereich 616
	Das Windows RLE-Format (RLE) 619
<hr/> 18	OS/2 Bitmap-Format (BMP) 621
	OS/2 Bitmap-Format (Version 1.1-1.3) 621
	OS/2 Bitmap-Format (Version 2.0) 623
<hr/> 19	Das CAS Fax-Format (DCX) 631
	Der DCX-Header 631
<hr/> 20	Computer Graphic Metafile-Format (CGM) 633
	Binäre CGM-Kodierung 633
	Kodierung als ASCII-Text 639
	Die Character-Kodierung mit ISO-Zeichen 642
	Metafile-Anweisungen 645
<hr/> 21	Das Dr. Halo-Format (PIC, CUT, PAL) 655

<hr/> <u>22</u>	Graphics Interchange Format (GIF) 661
	Der GIF-Header 662
	Die GIF-Blocks 662
	Subblocks mit Raster-Daten 674
<hr/> <u>23</u>	GEM Image File-Format (IMG) 681
	Der IMG-Kopfsatz 682
	Speichern von IMG-Daten 685
	Bildkomprimierung bei IMG-Dateien 686
	Pixelkodierung 686
	Das Solid Run-Format 687
	Das Bit String-Format 687
	Das Pattern Run-Format 688
	Das Vertical Replication Count-Format 689
<hr/> <u>24</u>	GEM-Metafile-Format (GEM) 693
	Der Aufbau des GEM-Metafile-Headers 693
	Das Format der Metafile-Objekte 695
	Generalized Drawing Primitives (GDP Opcode 0BH) 699
	Erweiterungen in GEM/3 716
<hr/> <u>25</u>	Initial Graphics Exchange Language (IGES) 719
	Die Start-Section 720
	Die Global-Section 720
	Die Directory-Entry-Section 722
	Die Parameter Data-Section 724
	Die Terminate-Section 724
	Die Elemente einer IGES-Datei 724
<hr/> <u>26</u>	Interchange File Format (IFF) 733
	Der IFF-Header 734
	IFF-Blockstruktur (CHUNK) 736
	Die CHUNKs eines ILBM-FORMs 738
	Die CHUNKs eines 8SVX-FORMs 744
	Die CHUNKs des AIFF-FORM 747

	Die CHUNKs des SMUS-FORM	748	
	Die CHUNKs des FTXT-FORM	749	
	Allgemeine CHUNKs	754	
<hr/>	27	Das JPEG/JFIF-Format (JPG)	757
		Die Marker einer JFIF-Datei	758
		Das SPIFF-Format	767
<hr/>	28	Das MAC-Paint-Format (MAC)	771
		Der MAC-Header	772
		Der MAC-Datenbereich	774
<hr/>	29	Das MAC-Picture-Format (PICT)	777
		Der PICT-Header	778
		Der PICT-Datenbereich	778
<hr/>	30	Micrografx-Formate (PIC, DRW, GRF)	789
		Recordtypen in den Grafikdateien	792
		Recordaufbau der Version 1	803
		Recordaufbau der Version 2	805
		Recordaufbau der Version 3	807
		Recordaufbau der Version 4	809
		Recordaufbau der Version 5	811
<hr/>	31	WINDOWS 2.0 PAINT File-Format (MSP)	819
		Der MSP-Header	819
		Die Indextabelle	820
		Der Datenbereich	820
<hr/>	32	Die PBM-Formate (PBM, PGM, PPM)	823
		Der PBM-Header	823
		Der PGM-Header	823
		Der PPM-Header	824
		Bilddaten	824

<hr/> 33	ZSoft Paintbrush File Format (PCX) 825 Der Aufbau des PCX-Headers 827 CGA-Farbpaletteninformationen 829 EGA/VGA-Palette mit 16 Farben 830 VGA-Palette mit 256 Farben 830 Die Kodierung der PCX-Daten 831 Das Format der PC Paintbrush-Bitmap-Zeichen 834 Das CAPTURE File-Format (SCR) 835
<hr/> 34	Das PCPAINT/Pictor-Format (PIC) 837 Der PCPAINT/Pictor-Header 837 Der PIC-Datenbereich 839
<hr/> 35	Das Portable Network Graphics-Format (PNG) 843 Der Aufbau der PNG-Datei 843 Kritische CHUNKs 845 Untergeordnete (ancillary) CHUNKs 847 Der Bilddatenbereich 851
<hr/> 36	Das Apple QuickTime-Format (QTM) 855 Das Movie Directory-Atom 856 Das Movie Header-Atom 857 Das Track Directory-Atom 858
<hr/> 37	Das SUN Rasterformat (RAS) 861 Der RAS-Header 861 Der Palettendatenbereich 863 Der RAS-Datenbereich 863
<hr/> 38	Das TARGA-Format (TGA) 865 Der TARGA-Header 865 Struktur des Headers 867 Palettendaten 869

Der Bilddatenbereich 870
Erweiterungen des TARGA-Formats 2.0 873

39 Tag Image File Format (TIFF 6.0) 877

Der TIFF-Header 877
Der Aufbau des Image File Directory (IFD) 878
Der Aufbau eines Tags 879
Beschreibung der Tag-Typen 881
TIFF-Komprimierungsverfahren 917

40 Das Windows Metafile-Format (WMF) 925

Der Metafile-Header 925
Die Metafile-Records 926
Das Enhanced Metafile-Format (EMF) 946

41 WordPerfect Graphic File Format (WPG) 951

WPG-Header 951
WPG-Records 951

Teil 4: Soundformate

42 Das Creative Music File-Format (CMF) 965

Der CMF-Header 965
Der Instrument-Block 967
Der Music-Block 969
Die Steuerbefehle 971

43 Das Creative Voice-Format (VOC) 975

Der VOC-Header 975
Der VOC-Datenbereich 976

44 Das AMIGA MOD-Format 981

Der MOD-Header 981
Der Noten-Block 982
Der Instrumentdatenbereich 983

45	Das MIDI-File-Format (MID)	985
	Der MIDI Header-CHUNK	985
	Der Track-CHUNK	987
	Die MIDI-Events	989
	Die Steuerbefehle	993
	Die MIDI-Betriebsartenbefehle	995
	Local Control	995
	All Notes off	995
	Die MIDI-Programm-Befehle	996
	Die MIDI-Timing-Befehle	997
	Die MIDI-System-Common-Befehle	997
	Die Meta-Events	1000

46	Das Windows WAV-Format	1005
	Der WAV-Header	1005
	Der FMT-CHUNK	1006
	Der DATA-CHUNK	1006

Teil 5: Beschreibungssprachen

47	Hewlett Packard Graphic Language (HP-GL/2)	1009
	HP-GL/2-Gruppen	1010
	Configuration and Status Group	1012
	Vector Group	1014
	Polygon Group	1017
	Line and Fill Attributes Group	1019
	Character Group	1021
	Technical Graphics Extension	1024
	Palette Extension	1027
	Dual Context Extension	1028
	Digitizing Extensions	1028

48	PostScript und Encapsulated PostScript (EPS)	1031
	Die EPS-Strukturkonventionen	1034
	PostScript-Anweisungen (Level 1)	1036
	PostScript Level 2	1047

<hr/> 49	Das Portable Document Format (PDF) 1067
	Hinweise zu PDF-Befehlen 1068
<hr/> 50	Standard Generalised Markup Language (SGML) 1071
	Struktur einer SGML-Datei 1071
	Struktur eines Dokuments 1072
<hr/> 51	Extended Markup Language (XML) 1081
	XML-Dateistruktur 1081
	XML-Elemente 1083
	Hinweise zur Extended Backus-Naur Form (EBNF) 1086
<hr/> 52	Hypertext Markup Language (HTML) 1089
	HTML-Varianten 1089
	Die Grundstruktur einer HTML-Datei 1090
	Sonderzeichen in HTML 1091
	Gestaltung von Überschriften 1093
	Horizontale Linien im Dokument 1096
	Dokumente mit Aufzählungen gestalten 1097
	Numerierungen in Dokumenten verwenden 1097
	Grafiken in HTML-Dokumenten 1098
	Hintergrundbilder und Hintergrundfarben 1099
	Verweise in HTML-Dokumenten 1099
	Tabellen in HTML-Dokumenten 1101
	HTML-Referenz 1102
	Index 1107

22 Graphics Interchange Format (GIF)

Das 1987 von der Firma CompuServe definierte GIF89a-Format wurde 1989 um einige Zusätze unter dem Begriff GIF89a erweitert. In diesen GIF-Dateien lassen sich mehrere Bilder (wird für animierte GIF-Bilder benutzt) und transparente Bilder hinterlegen. Populär wurde das GIF-Format durch das World Wide Web, da die HTML-Spezifikation Bilder in diesem Format unterstützt. Nachfolgend finden Sie die Beschreibung des GIF89a-Formats, beim GIF87a-Format fehlen einige dieser Bildelemente.

Eine GIF89a-Datei besteht (wie die GIF98a-Datei) aus mehreren Blöcken zur Aufnahme der Grafiken und zusätzlich benötigter Daten. Dabei werden ähnlich wie beim TIFF-Format Tag-Felder benutzt. Die Blöcke lassen sich in drei Gruppen zusammenfassen:

- ▶ Control-Blocks
- ▶ Graphic Rendering-Blocks
- ▶ Special Purpose-Blocks

Die Control-Blocks (z.B. *Header*, *Logical Screen Descriptor*, *Graphics Control Extension*, *Trailer*) enthalten Informationen zur Steuerung der Bildwiedergabe. In den *Graphic Rendering-Blocks* (z.B. *Image Descriptor*, *Plain Text Extension* etc.) finden sich die eigentlichen Daten zur Ausgabe der Grafiken. Die *Special Purpose-Blocks* (z.B. *Comment Extension*, *Application Extension*) dienen zur Aufnahme herstellerspezifischer Informationen und sollten vom GIF-Decoder überlesen werden.

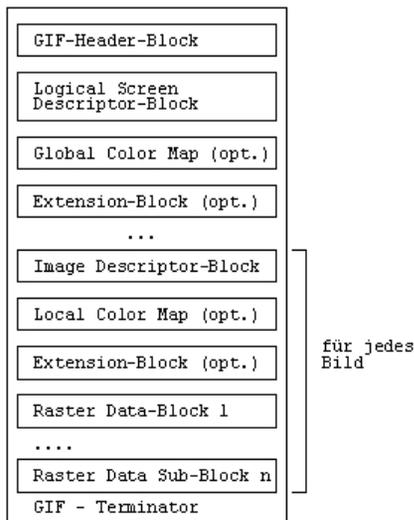


Abbildung 22.1 Struktur einer GIF-Datei

Die Blockgröße ist mit Ausnahme der Subblocks zur Aufnahme von Daten fest definiert. Enthält der Block ein *Block Size*-Feld, gibt dieser Wert die Zahl der folgenden Bytes im Block an. Diese Größe umfaßt nicht einen eventuell folgenden Terminator. Abbildung 22.1 gibt den strukturellen Aufbau einer GIF-Datei wieder.

Für jedes Bild der GIF-Datei werden ein Image Descriptor und ein oder mehrere *Raster Data*-Blocks abgelegt. Ein *Local Color Map*-Block kann optional auftreten. Innerhalb der Blockstruktur dürfen sogenannte Subblocks auftreten (z. B. Blöcke mit Bilddaten). Diese Subblocks bestehen aus einem Längenbyte, welches die Zahl der Folgebytes im Block (0–255) definiert. Daran schließt sich der Datenbereich mit 0 bis 255 Byte an.

Der GIF-Header

Eine GIF-Datei muß immer mit einem Header beginnen, der in beiden GIF-Versionen gleich ist. Dieser Header umfaßt 6 Byte und besitzt die in Tabelle 22.1 definierte Struktur (siehe auch vorhergehendes Kapitel).

Offset	Bytes	Feld
00H	3	Signatur 'GIF'
03H	3	Version '87a' oder '89a'

Tabelle 22.1 GIF87a- bzw. GIF89a-Header

In den ersten drei Bytes steht eine Signatur für den Header ('GIF'). Daran schließen sich drei Bytes mit der GIF-Version (GIF87a oder GIF89a) an. Der Header darf nur einmal innerhalb der GIF-Datei auftreten, und es muß der erste Block in der Datei sein. Die folgende Blockstruktur ist für GIF87a und GIF89a gleich, wobei in GIF89a zusätzliche Blocktypen definiert wurden. Weiterhin wurden einige Flags in der GIF89a-Version etwas modifiziert. An den betreffenden Stellen der Beschreibung finden Sie Hinweise auf die Änderungen. Auch ältere GIF87a-Decoder können eine GIF89a-Datei bearbeiten, wobei aber die erweiterten Blocktypen zu überlesen sind und eventuell Informationen verlorengehen.

Die GIF-Blocks

An den Header schließen sich verschiedene Blöcke mit Informationen an. Die verschiedenen Blocktypen werden nachfolgend beschrieben.

Der Logical Screen Descriptor-Block

An den Header muß sich (ab Offset 06H) der 7 Byte große *Logical Screen Descriptor*-Block mit den Daten des logischen Bildschirms anschließen. Auch dieser Block wird in GIF87a und GIF89a verwendet. Die Daten des Blocks gelten für die komplette GIF-Datei und sind in folgender Struktur abgelegt:

Bytes	Feld
2	Logical Screen Width
2	Logical Screen Height
1	Resolution-Flag
1	Background Color Index
1	Pixel Aspect Ratio

Tabelle 22.2 Struktur des Logical Screen Descriptor-Blocks

Der erste Eintrag umfaßt einen 16-Bit-Wert und gibt die Breite des logischen Bildschirms in Pixel an. Dabei wird die Intel-Notation (Low-Byte first) zur Speicherung benutzt. Das nächste Feld mit der Bildschirmhöhe enthält ebenfalls einen 16-Bit-Wert. Die beiden Werte für die Bildabmessungen beziehen sich auf einen virtuellen Bildschirm, dessen Nullpunkt sich in der oberen linken Ecke befindet.

Ab Offset 04H findet sich im *Logical Screen Descriptor*-Block ein Bitfeld (Resolution-Flag) mit der Kodierung gemäß Abbildung 22.2.

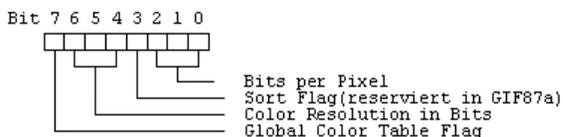


Abbildung 22.2 Kodierung des Resolution-Flag

Das oberste Bit 7 markiert, ob eine *Global Color Map* (Definition der Farbpalette) existiert. Ist Bit 7 auf 1 gesetzt, folgt auf den *Logical Screen Descriptor*-Block die *Global Color Map*. Ist dieses Bit auf 0 gesetzt, fehlt die *Global Color Map*, und die Bits für den *Background Color Index* (Hintergrundfarbe) besitzen keine Bedeutung.

In den Bits 4 bis 6 wird festgehalten, wie viele Bits zur RGB-Darstellung einer primären Farbe (Color Resolution) in der Farbtabelle zur Verfügung stehen. Der Wert in den Bits ist jeweils um 1 zu erhöhen. Der Wert 3 besagt, daß pro primäre Farbe 4 Bit in der entsprechenden Palette benutzt werden.

Bit 3 ist in der GIF87a-Spezifikation noch als reserviert markiert und muß auf 0 gesetzt werden. In der GIF89a-Spezifikation enthält dieses Bit das Sort-Flag. Der Wert 1 signalisiert, daß die *Global Color Table* sortiert vorliegt. Die Sortierung erfolgt dabei nach den Farben mit absteigender Bedeutung, d.h., die häufiger verwendeten Farben befinden sich am Beginn der Palette. Dies ist für Decoder, die nur wenige Farben unterstützen, hilfreich. Mit dem Wert 0 liegt die *Global Color Map* unsortiert vor.

In den Bits 0 bis 2 wird in GIF87a die Zahl der Bit pro Pixel angegeben. Der Wert ist dabei um 1 zu erhöhen. Der maximale Wert 7 bedeutet, daß 8 Bit pro Pixel verfügbar sind. Damit lassen sich 256 unterschiedliche Farben in einem Bild verwenden. In der GIF89a-

Spezifikation wird angegeben, daß bei gesetztem *Color Table*-Flag die drei Bits die Größe der globalen Farbpalette (global color table size) in Byte definieren. Letztlich handelt es sich aber um den gleichen Wert (Bit pro Pixel), da sich die Größe der Farbpalette zu

$$\text{Color Table Size} := 2^{(\text{value} + 1)}$$

berechnen läßt. Der Wert sollte auch dann gesetzt werden, wenn die GIF-Datei keine *Global Color Map* enthält. Dies ermöglicht dem Decoder, den entsprechenden Graphikmodus einzustellen.

Ab Offset 05H findet sich im *Logical Screen Descriptor*-Block ein Byte mit der Kodierung der Hintergrundfarbe (Background Color Index). Diese Hintergrundfarbe wird aus den 256 möglichen Farben ausgewählt. Die Hintergrundfarbe wird für die Teile des Bildschirms benutzt, die nicht durch die Bitmap belegt werden. Dieser Wert wird zum Beispiel benutzt, um transparente GIF-Grafiken für HTML-Dokumente zu erstellen. Falls das *Global Color-Flag* gelöscht (0) ist, sollte das Byte auf 0 gesetzt und vom Decoder überlesen werden.

Das Byte ab Offset 06H im *Logical Screen Descriptor*-Block definiert das *Pixel Aspect Ratio*. Die Belegung wird jedoch in GIF87a und GIF89a unterschiedlich gehandhabt. In GIF87a gilt die Kodierung gemäß Abbildung 22.3.

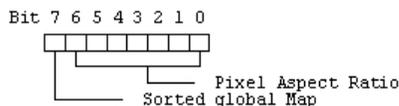


Abbildung 22.3 Kodierung des Pixel Aspect Ratio-Flags (GIF87a)

In GIF87a wird Bit 7 als *Sorted Global Color Map*-Flag verwendet. Die restlichen Bits geben das *Pixel Aspect Ratio* des ursprünglichen Bildes an.

In GIF89a ist das *Sorted Global Color Map*-Flag im *Resolution*-Flag integriert (siehe Abbildung 22.2). Daher werden alle Bits des *Pixel Aspect Ratio*-Feldes benutzt. Ist der Wert des Feldes ungleich 0, läßt sich das Verhältnis der Bildabmessungen zu:

$$\text{Aspect Ratio} = (\text{Pixel Aspect Ratio} + 15) / 64$$

berechnen. Das + ist als Quotient der Bildbreite zur Bildhöhe definiert. Die Spezifikation erlaubt einen Bereich zwischen 4:1 bis 1:4 in 1/64 Schritten.

Der Global Color Map-Block

Im Anschluß an den *Logical Screen Descriptor*-Block kann optional ein Block mit der *Global Color Map* (globale Farbpalette) gespeichert sein. Dies ist immer dann der Fall, wenn im vorhergehenden *Logical Screen Descriptor*-Block das Bit 7 des *Resolution*-Flag gesetzt ist.

Im *Color Map*-Block wird die globale Farbtabelle für die nachfolgenden Bilder spezifiziert. Diese globale Farbpalette wird immer dann verwendet, wenn ein Bild keine lokale Palette besitzt.

Für jeden Bildpunkt sind maximal 8 Bit vorgesehen, womit sich lediglich 256 Farben oder Graustufen abbilden lassen. Eine True-Color-Darstellung (Echtfarbenanzeige) ist nicht vorgesehen. Die *Global Color Map* enthält für jede der 256 Farben ein Tripel (3 Byte) mit den Grundfarben Rot, Grün und Blau (Abbildung 22.4).

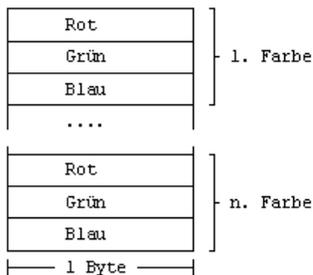


Abbildung 22.4 Struktur der Global Color Map

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 Farben darstellen, wobei jedoch nur 256 Farben über die Palette im Bild genutzt werden können. Der Wert eines Bildpunktes wird als Offset in die Farbtabelle interpretiert, und die Grafikkarte generiert dann den zugehörigen Farbwert. Die Größe der Farbtabelle und die Zahl der Bits pro Farbe werden ebenfalls im *Resolution*-Flag spezifiziert ($Palette\ Size = 3\ Byte * 2^{**} Bits\ pro\ Pixel$).

Der Image Descriptor-Block

Jedes Bild innerhalb der GIF-Datei muß durch einen 10 Byte großen *Image Descriptor*-Block eingeleitet werden. Der Block wird in beiden GIF-Varianten verwendet. Die Struktur dieses Blocks ist Tabelle 22.3 zu entnehmen.

Bytes	Feld
1	Image Separator Header (ASCII 2CH = ',')
2	Koordinate linker Rand
2	Koordinate oberer Rand
2	Bildbreite
2	Bildhöhe
1	Flags

Tabelle 22.3 Struktur eines Image Descriptor-Blocks

Der Block enthält die wichtigsten Daten eines Bildes wie Abmessungen, Koordinaten der linken oberen Ecke etc. Das erste Byte wird mit dem Separator ('', 2CH) gefüllt. Die beiden folgenden Felder geben die Bildkoordinaten für die obere linke Ecke des Bildes in Pixel an und umfassen jeweils 16 Bit (Unsigned Word). Diese Daten beziehen sich auf den logischen Bildschirm.

Ab dem dritten Feld (Offset 05H im Block) wird die Breite des Bildes (Image Width) in Pixel angegeben. Das folgende Wort definiert die Bildhöhe in Pixel. Beide Werte werden als unsigned Word definiert. Das letzte Byte dient zur Aufnahme verschiedener Flags, die gemäß Abbildung 22.5 kodiert sind.

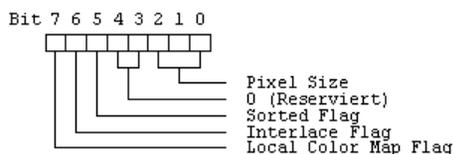


Abbildung 22.5 Kodierung des Flags im Image Descriptor-Block (IDB)

Das oberste Bit spezifiziert, ob sich eine lokale Palette (Local Color Map) an den *Image Descriptor*-Block anschließt. Ist Bit 7 = 1, folgt dem *Image Descriptor*-Block eine *Local Color Map*. In diesem Fall werden diese Daten für die Farbpalette des folgenden Teilbildes verwendet. Ein Decoder muß dann die *Global Color Map* sichern und die neuen Daten benutzen. Nach Bearbeitung des Bildes sind die Daten der *Global Color Map* zu restaurieren.

Bit 6 definiert das *Interlaced*-Flag, d. h., das Grafikbild kann sequentiell (Bit 6 = 0) oder interlaced (Bit 6 = 1) gespeichert sein. 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., 18. usw. Zeile und beim dritten Durchlauf die 2., 6., 10. usw. Zeile.

Bit 5 gibt in GIF89a an, ob die lokale Farbtabelle nach Farben sortiert ist. Bei gesetztem Bit werden die wichtigsten Farben zuerst in der Palette abgelegt. Die am häufigsten verwendete Farbe sollte zuerst gespeichert werden. Beim Flag = 0 werden die Farben der Palette unsortiert abgelegt. Bit 5 wird aber nur in wenigen Anwendungen benutzt, da die offizielle GIF87a-Dokumentation dieses Bit nicht erwähnt und zu 0 setzt.

Anmerkung: Die Bits 3 bis 5 sind in GIF87a als reserviert markiert und müssen auf 0 gesetzt werden.

Die unteren Bits 0 bis 2 definierten die Zahl der Bits pro Bildpunkt (GIF87a und GIF89a). Der Wert der Bits ist um 1 zu erhöhen. Mit einem Eintrag von 7 werden 8 Bit pro Pixel verwendet. Aus diesem Wert läßt sich die Zahl der Einträge in der Farbpalette (2^{**n}) und damit deren Größe (Einträge * 3 Byte) berechnen. Der Wert dieser drei Bits sollte auf 0 gesetzt werden, falls keine *Local Color Map* folgt.

Der Local Color Map-Block

Neben der globalen Farbtabelle lassen sich auch vor den *Raster Image*-Blöcken eines Teilbildes lokale Farbpaletten definieren. Ist das *Local Color Table*-Flag im *Image Descriptor*-Block gesetzt, folgt auf diesen Block der *Local Color Map*-Block. Dann muß der Decoder die Daten der *Global Color Map* sichern. Erst nach Bearbeitung der Bilddaten ist die *Global Color Map* zu restaurieren. Der Aufbau der *Local Color Map* stimmt mit dem der *Global Color Map* überein (siehe Abbildung 22.4). Eine GIF-Datei kann mehrere *Local Color Map*-Blocks enthalten. Die *Local Color Map*-Blocks sind in beiden GIF-Spezifikationen definiert.

Der Extension-Block

An den Block mit der Local Color Map kann sich ein optionaler *Extension*-Block (Erweiterungsblock) anschließen. Über diese *Extension*-Blocks wurde in GIF89a ein Weg für zukünftige Erweiterungen des Formats geschaffen. In GIF87a waren diese *Extension*-Blocks zum Speichern von Informationen über das bilderzeugende Gerät, die benutzte Software, die Ausrüstung zur Bildabtastung (Scanner) etc. vorgesehen. Die *Extension*-Blocks haben den in Tabelle 22.4 beschriebenen Aufbau.

Bytes	Feld
1	Extension-Block Header (ASCII 21H = '!')
1	Funktionscode (0 .. 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	OOH als Terminator

Tabelle 22.4 Struktur eines Extension-Blocks (GIF89a)

Das erste Byte des Erweiterungsblocks enthält das Zeichen ! als Signatur. Darauf folgt ein Byte mit dem Funktionscode, der die Art der nachfolgenden 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.

Bei längeren Datensequenzen sind mehrere Subblocks zu speichern. Das Ende des *Extension*-Blocks wird durch ein Nullbyte (00H) markiert.

Die Belegung der Funktionscodes ist in GIF87a nicht definiert. Auch der interne Aufbau wurde dem jeweiligen Entwickler des Encoders überlassen. In GIF89a ändert sich die Situation, denn die Spezifikation beschreibt verschiedene *Extension*-Blocks. Am Ende dieses Kapitels wird der Aufbau dieser *Extension*-Blocks für GIF89a vorgestellt.

Der Raster Data-Block

Die eigentlichen Bilddaten (Image Data) werden in einem oder mehreren *Raster Data*-Blocks abgelegt. Ist im *Image Descriptor*-Flag (Abbildung 22.5) das Flag für die *Local Color Map* gesetzt, folgen die Bilddaten im Anschluß an die Farbtabelle. Fehlt die *Local Color Map*, schließen sich die Daten an den *Image Descriptor*-Block oder an einen *Extension*-Block an. Der erste *Raster Data*-Block besitzt den Aufbau gemäß Tabelle 22.5.

Bytes	Feld
1	Code Size
1	Bytes im Datenblock
n	Datenbytes

Tabelle 22.5 Struktur eines Raster Data-Blocks

Im ersten Byte des ersten Blocks steht ein Byte, welches als *Code Size* bezeichnet wird. Dieser Wert definiert die minimale Codelänge, die für die Darstellung der Pixel bei der LZW-Komprimierung benötigt wird. Dieses Byte wird zur Initialisierung des Decoders benutzt. In der Regel stimmt der Wert mit der Zahl der Farbbits pro Bildpunkt überein. Nur bei Schwarzweißbildern (Bit pro Pixel = 1) muß *Code Size* = 2 gewählt werden.

Das zweite Byte definiert die Zahl der Bytes im nachfolgenden Datenblock. Der Wert kann dabei zwischen 0 und 255 liegen.

Ab dem dritten Byte folgen die komprimierten Bilddaten. Umfassen die Bilddaten mehr als 255 Byte, folgen weitere Subblocks mit den restlichen Daten (siehe Abschnitt Subblocks mit Rasterdaten). Nach der Dekodierung ist das Bild, beginnend in der linken oberen Ecke, von links nach rechts und von oben nach unten aufzubauen. Zur Komprimierung wird der LZW-Algorithmus von Lempel Ziv und Welch in leicht modifizierter Form benutzt (siehe folgenden Abschnitt).

Die LZW-Komprimierung

In Programmen zur Bearbeitung von GIF-Grafikdateien wird die Technik des LZW-Verfahrens (Lempel-Ziv-Welch) benutzt. Der Algorithmus versucht den zu komprimierenden Zeichenstrom in Teilketten zu zerlegen und diese in einer Tabelle zu speichern. Anschließend werden nur die Indizes in die betreffende Tabelle als Ausgabecodes gespeichert. Anhand dieser Ausgabecodes läßt sich dann der ursprüngliche Zeichenstrom wieder generieren. Der prinzipielle Aufbau ist in Abbildung 22.6 wiedergegeben.

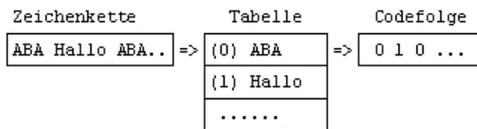


Abbildung 22.6 LZW-Encoding

Die Zeichenfolge aus Abbildung 22.6 läßt sich in Teilstrings (ABA, Hallo etc.) aufteilen. Diese Zeichenfolgen tauchen auch in der Kodierungstabelle auf. An Stelle der Teilstrings wird dann jeweils der zugehörige Tabellenindex als Ausgangscode übertragen. Aus der Ursprungszeichenkette mit 11 Zeichen (ABAHalloABA) wird dann eine Codefolge mit 3 Zeichen (010). Mit diesem recht einfachen Prinzip lassen sich beliebige Zeichenfolgen komprimieren.

Allerdings sind noch zwei Probleme zu beheben. Einmal wird die Codetabelle sich nur in den seltensten Fällen a priori bestimmen lassen. Sie wird aber auch nicht mit dem Code gespeichert und steht bei der Dekomprimierung nicht mehr zur Verfügung. Deshalb muß der Algorithmus bei der Komprimierung und Dekomprimierung die Tabelle jeweils selbst aufbauen. Die zweite Schwierigkeit betrifft die Größe der Tabelle. Theoretisch muß diese Tabelle unendlich groß sein, um alle Zeichenkombinationen aufzunehmen. Aus praktischen Gründen wird jedoch die Größe der Tabelle limitiert. Mit der Implementierung des LZW-Algorithmus für GIF-Dateien werden diese Probleme behoben.

Zur Diskussion des Verfahrens möchte ich vorher noch einige Begriffe erläutern.

- <new>: Speicherzelle mit dem letzten gelesenen Zeichen
- <old> : Speicherzelle mit dem vorletzten gelesenen Zeichen
- [..] : Zeichenpuffer zum Aufbau der Tabelle
- [..]K : Zeichenpuffer mit angefügtem Zeichen K

Für den Aufbau der Tabelle ist dabei der Puffer [..] von Bedeutung. Dieser Puffer kann einzelne Zeichen oder komplette Zeichenketten aufnehmen. Ziel des Algorithmus ist es, komplette Zeichenketten zu bilden, die noch nicht in der Tabelle abgespeichert sind. Tritt dieser Fall auf, wird der betreffende Teilstring in die Tabelle angefügt und ein Code für die

letzte vorhandene Teilkette ausgegeben. So baut sich die Kodierungstabelle selbst auf und kann anhand des Ausgabecodes jederzeit bei der Dekomprimierung rekonstruiert werden.

Weiterhin stellt sich die Frage nach der Initialisierung der Tabelle zu Beginn der Komprimierung. Die Größe muß willkürlich festgelegt werden. Mit 12 Bit lassen sich zum Beispiel 4096 Einträge kodieren. Jeder Eintrag speichert später eine Zeichenfolge. Als Ausgabecode werden dagegen nur die 12-Bit-Indizes der Tabelle gespeichert. Dies führt zu der gewünschten Komprimierung. Mit etwas Kenntnis über die möglichen Zeichen im Eingabecode läßt sich die Tabelle teilweise mit Anfangswerten initialisieren. Nehmen wir an, die Eingangszeichen stammen aus dem Alphabet der Großbuchstaben. Dann können die ersten 26 Einträge der Tabelle mit den Codes der Zeichen (0 = A, 1 = B, 2 = C etc.) belegt werden.

Der LZW-Algorithmus zur Komprimierung läßt sich dann mit folgenden Pseudocodeanweisungen beschreiben.

```
initialize table
Clear Buffer [...]
WHILE Not EOF DO
  Read Code in K
  IF [...]K in Tabelle?
    [...] <- [...]K
  ELSE
    Add [...]K in Tabelle
    Write Tabellenindex von [...]
    [...] <- K
  ENDIF
WEND
Write Tabellenindex von [...]
```

Zuerst ist die Tabelle zu initialisieren und der Puffer [...] zu leeren. Dann wird die Folge von Eingabecodes zeichenweise gelesen. Das gerade gelesene Zeichen der Eingabefolge wird rechts vom Puffer als Postfix beigestellt ([...]K). Nun ist zu prüfen, ob der so gebildete Teilstring [...]K bereits in der Tabelle vorkommt. In diesem Fall bearbeitet der Algorithmus das nächste Zeichen. Kommt der String noch nicht in der Tabelle vor, beginnt der nächste Schritt. Als erstes wird der neue String [...]K in der Tabelle an der ersten freien Position angehängt. Dann wird der Tabellenindex des Teilstrings [...] (not [...]K) als Ausgabecode gespeichert. Abschließend ist der alte Pufferinhalt durch das zuletzt gelesene Zeichen K zu ersetzen. Dann wird das nächste Zeichen gelesen, und die Bearbeitung beginnt erneut. Erst wenn alle Zeichen gelesen wurden, ist der Code des Puffers mit dem Reststring aus der Tabelle zu ermitteln und auszugeben.

Dieser Sachverhalt soll nochmals an einem kleinen Beispiel verdeutlicht werden. Aus der Menge der Buchstaben *A, B, C, D* soll die Zeichenkette *ABACABA* gebildet werden. Die Tabelle läßt sich dann in den ersten vier Einträgen mit den Buchstaben *A, B, C, D* initialisieren. Die folgende Abbildung gibt die einzelnen Schritte bei der Komprimierung wieder:

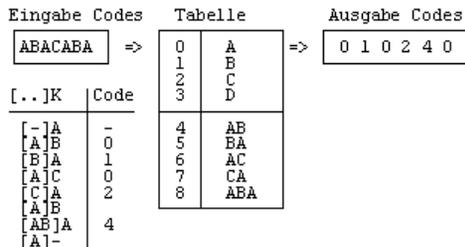


Abbildung 22.7 LZW-Kompression

Aus der Zeichenfolge *ABACABA* wird dann die Codefolge *010240*. Diese Folge umfaßt nur noch 6 Codes, während der ursprüngliche String 7 Zeichen enthält. Bei längeren Zeichenketten ergeben sich wesentlich bessere Komprimierungsverhältnisse.

Bei der Dekomprimierung muß nun die Codefolge wieder in den Ausgabestring überführt werden. Hierzu wird ein Algorithmus verwendet, der sich mit folgenden Pseudocodeanweisungen beschreiben läßt:

```

Initialize Stringtabelle
<code> := 1. Code Byte
Tabelle[<code>] -> Ausgabestring
<old> := <code>
WHILE NOT EOF DO
  <code> := Next Code Byte
  IF Tabelle[<code>] belegt?
    Tabelle[<code>] -> Ausgabestring
    [..] <- Tabelle[<old>]
    K <- 1. Zeichen (Tabelle[<code>])
    Write [..]K in Tabelle
    <old> := <code>
  ELSE
    [..] <- Tabelle[<old>]
    K <- 1. Zeichen (..)
    [..]K -> in Ausgabestring
    [..]K -> Tabelle
    <old> := <code>
  END IF
WEND

```

Bezüglich der verwendeten Nomenklatur gilt: `<old>` und `<code>` sind zwei Variablen, in denen das vorletzte und das letzte gelesene Zeichen stehen. Mit `[..]` wird ein Zeichenpuffer definiert, der Strings aus der Stringtabelle enthält. Die Stringtabelle wird mit `Tabelle[..]` bezeichnet. Mit `-> Ausgabestring` werden Strings in die Ausgabereinheit geschrieben.

Der Algorithmus initialisiert zuerst die Stringtabelle mit Basiswerten. Dann wird das erste Element der Codefolge gelesen. Der Wert dient als Index in die Stringtabelle. Der betreffende String (der Index zeigt auf den initialisierten Teil der Tabelle) wird dann in den Ausgabestrom geschrieben. Nun wird der gelesene Code in der Variablen `<old>` gespeichert. Die WHILE-Schleife sorgt dafür, daß alle Elemente der Codefolge bearbeitet werden. Sobald ein Codeelement gelesen wurde, prüft der Algorithmus, ob der Index auf einen bereits belegten Tabelleneintrag zeigt. Falls dies der Fall ist, wird der String in den Ausgabestrom angehängt. Anschließend wird der String aus `Tabelle[old]` in einen Zeichenpuffer übernommen und das erste Zeichen des Eintrags von `Tabelle[code]` angehängt (`[..]K`). Der neue Teilstring `[..]K` wird in der Tabelle an der ersten freien Position eingetragen. Dann muß nur noch das zuletzt gelesene Zeichen von `<code>` nach `<old>` kopiert werden.

Ist der Tabellenplatz für `<code>` noch unbelegt, muß der zugehörige String generiert werden. Hierzu wird der bereits existierende String aus `Tabelle[old]` in den Puffer `[..]` kopiert. Dann wird das erste Zeichen des Puffers `[..]` in die Variable `K` kopiert und an den Puffer angehängt `[..]K`. Dieser neue String wird sowohl in die Tabelle eingetragen als auch in den Ausgabestrom kopiert. Schließlich wird noch der Inhalt von `<code>` in `<old>` gesichert. Der Vorgang wiederholt sich so lange, bis alle Eingangscodes gelesen sind. Anschließend findet sich die ursprüngliche Zeichenfolge wieder im Ausgabestrom. Die folgende kleine Abbildung gibt die einzelnen Schritte der Dekomprimierung wieder.

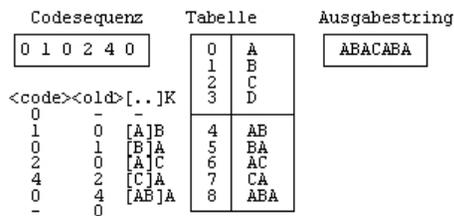


Abbildung 22.8 LZW-Dekodierung

Aus der Codefolge 010240 wurde wieder die ursprüngliche Zeichenfolge ABACABA generiert. Zu Beginn der Dekodierung wird die Stringtabelle mit den Zeichen A, B, C, D initialisiert. Dies waren die ursprünglichen Initialisierungszeichen der Komprimierungstabelle. Während der Bearbeitung der Eingabecodes wird die komplette Zeichentabelle wieder aufgebaut. Zum Abschluß liegt dann die komplette Tabelle mit allen Einträgen der Komprimierungstabelle vor (vergleichen Sie die beiden Beispiele).

Achtung: So elegant das LZW-Komprimierungsverfahren im GIF-Format funktioniert, gibt es einen gravierenden Nachteil, die Firma Unisys hält ein Patent auf dieses Verfahren. Dies führt dazu, dass Softwareentwickler Lizenzgebühren für Programme zahlen müssen, die das GIF-Format unterstützen. Selbst Webseitenanbieter werden mittlerweile von Unisys zur Kasse gebeten, da die GIF-Dateien häufig nicht mit lizenzierter Software erstellt wurden. Daher ist das PNG-Format wesentlich besser für Webgrafiken geeignet.

Modifiziertes LZW-Verfahren für GIF-Dateien

Bei der Komprimierung von GIF-Dateien gelangt das oben beschriebene Verfahren mit leichten Modifikationen zum Einsatz. Da es sich bei den Codes um Bitfolgen handelt, muß die Zahl der Bits pro Lesezugriff definiert werden. Es ist nicht sinnvoll, jeweils nur ein Bit (Pixel) zu lesen. Auch die Länge von 8 Bit pro Lesezugriff ist problematisch. Bei der GIF-Komprimierung wird vielmehr eine variable Codelänge verwendet, die zwischen 3 und 12 Bit lang sein darf. Im *Raster Data*-Block enthält das erste Byte die Größe Code Size. Dieser Wert entspricht zwar der Zahl der Bits pro Pixel. Der Wert wird aber als Startwert für die Codelänge interpretiert. Bei vier Bit pro Pixel wird in dem betreffenden Feld die Zahl $N = 3$ gespeichert. Dies bedeutet, daß bei jedem Zugriff auf die Eingangsdaten immer $N + 1 = 4$ Bit zu lesen sind. Während der weiteren Bearbeitung kann sich die Codelänge pro Zugriff auf 12 Bit erweitern. Sobald diese Länge erreicht wird, ist die Tabelle mit den Ausgabemustern voll, d. h., diese muß zurückgesetzt werden.

Der GIF-Komprimierer muß also eine Tabelle mit 4096 Einträgen anlegen. Zu Beginn wird diese Tabelle mit einigen Codes initialisiert. Die Größe des Initialisierungsbereiches wird dabei durch die Größe Code Size definiert. Bei 1 Bit pro Pixel muß $N = 2$ gesetzt werden. Dann werden die Einträge #0 und #1 der Tabelle initialisiert. Weiterhin werden an der Position $2^{*}N$ und $2^{*}N+1$ zwei Spezialcodes abgelegt. Bei $N = 2$ sind dies die Positionen #4 und #5. Der erste Eintrag an der Position #4 wird als *clear code* <CC> bezeichnet. Wird dieser Eintrag bei der Dekomprimierung erkannt, muß die Tabelle neu initialisiert werden. Der Komprimierer wird diesen Code immer dann ausgegeben, wenn die Tabelle voll ist. Weiterhin kann der Code als erstes Zeichen des Ausgangsstroms auftreten, um im Dekomprimierer einen Reset auszulösen. Der zweite Eintrag wird als *end of information* <EOI> bezeichnet. Er signalisiert dem Dekomprimierer, daß das Ende des Codestromes erreicht ist und keine weiteren Daten folgen.

Bei der Komprimierung/Dekomprimierung sollten neue Strings ab der Position <CC>+2 gespeichert werden. Der Code <CC> wird zu Beginn der Komprimierung und bei jedem Tabellenüberlauf in den Ausgabestrom geschrieben. Dann muß der Leser die Tabelle jeweils neu initialisieren. Die variable Codelänge der zu lesenden Daten sollte kein größeres Problem sein. Bei der Komprimierung wird mit der in *code size + 1* angegebenen Größe begonnen. Immer wenn ein Code aus der Tabellenposition $(2^{*}(Codelänge) - 1)$ ausgegeben wird, ist die Codelänge um 1 zu erhöhen. Dies wird so lange weitergeführt,

bis die Codelänge von 12 Bit erreicht ist. Dann muß die Tabelle neu initialisiert werden. Bei der Dekomprimierung beginnt man ebenfalls mit der in *code size + 1* vereinbarten Codelänge. Die Codelänge ist immer um 1 zu erhöhen, sobald der Tabelleneintrag ($2^{**}(\text{Codelänge}) - 1$) in die Ausgabe geschrieben wird. Beachten Sie auch, daß die Bits in der Codefolge mit den Bits der Stringcodes der Tabelle übereinstimmen.

Subblocks mit Raster-Daten

Da ein Bild in der Regel mehr als 255 Datenbytes umfaßt, werden die komprimierten Daten in einen *Raster Data*-Block und mehrere (Raster Data-)Subblocks unterteilt. Die Subblocks besitzen einen Aufbau ähnlich Tabelle 22.5, wobei das erste Byte (Code Size) fehlt. Ein Subblock beginnt mit einem Längenbyte, welches die Zahl der Folgebytes im Block definiert. Es können dann zwischen 0 und 255 Datenbytes folgen. Der Subblock kann damit maximal 256 Byte umfassen.

Terminator-Block

Das Ende des Bilddatenbereichs wird durch einen *Terminator*-Block (00H) angezeigt. Dies ist nichts anderes als ein Subblock, bei dem das Byte mit der Längenangabe auf 0 gesetzt wird. Dieser Block umfaßt dann nur das Längenbyte.

Die Blöcke *Image Descriptor*, *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.

Der Graphic Control Extension-Block (GIF89a)

Dieser Block wurde in GIF89a neu definiert und enthält zusätzliche Parameter für ein Bild. Der Block ist optional und muß nach dem *Image Descriptor*-Block, aber vor den eigentlichen *Raster Data*-Blöcken stehen. Die Daten gelten nur für das folgende (Teil-)Bild. Der Block besitzt den in Tabelle 22.6 gezeigten Aufbau.

Bytes	Bemerkungen
1	Extension-Block-Signatur (21H)
1	Graphic Control Label (F9H)
1	Block Size (4)
1	Flags
2	Delay Time
1	Transparent Color Index
1	Block-Terminator (00H)

Tabelle 22.6 Struktur eines Graphic Control Extension-Blocks (GIF89a)

Das erste Byte enthält die Signatur für den *Extension*-Block. Das Byte wird fest auf den Wert 21H (entspricht !) gesetzt.

Byte 2 enthält das *Graphic Control Label*. Dies ist nichts anderes als die Signatur für einen *Graphic Control Extension*-Block und wird fest auf den Wert F9H gesetzt. Das Feld *Block Size* gibt die Zahl der folgenden Datenbytes im Block an. Der Block-Terminator wird jedoch nicht eingerechnet. Das Feld enthält bei einem *Graphic Control Extension*-Block immer den Wert 4. Das folgende Flagbyte (Offset 03H) besitzt einen Aufbau gemäß Abbildung 22.9.

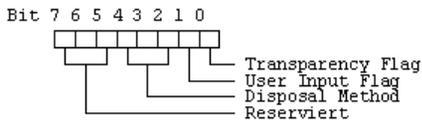


Abbildung 22.9 Kodierung des Flag im Graphics Control Extension-Block

Das *Transparency*-Flag signalisiert, ob das *Transparency Index*-Feld belegt ist. In diesem Fall wird das Flag auf 1 gesetzt. Ist das Flag auf 0 gesetzt, enthält das *Transparency Index*-Feld keinen gültigen Wert. (Dies wird in HTML genutzt, um transparente Bilder darzustellen.)

Das *User Input*-Flag definiert, ob auf eine Benutzereingabe nach der Ausgabe des Bildes gewartet wird. Ist das Flag auf 1 gesetzt, wird die Bearbeitung erst nach einer Benutzereingabe fortgesetzt. Die Art der Benutzereingabe (Return, Mausklick etc.) wird durch die Anwendung definiert. Ist eine Verzögerungszeit (Delay Time) definiert, wird die Bearbeitung nach dieser Wartezeit auch ohne Benutzereingabe fortgesetzt.

Die *Disposal*-Methode definiert, was mit der Grafik nach der Ausgabe passiert. Hierbei gibt es folgende Möglichkeiten:

- ▶ 0 No disposal specified
- ▶ 1 Do not dispose
- ▶ 2 Restore to background color
- ▶ 3 Restore to previous

Bei der Option 1 bleibt die Grafik erhalten. Beim Code 2 ist der Bereich der Grafik auf die Hintergrundfarbe zu setzen. Beim Code 3 ist die vorherige Einstellung zu restaurieren. Alle anderen Werte sind undefiniert.

Das Feld *Delay Time* ist als vorzeichenlose Zahl (unsigned Word) definiert. Ist das *User*-Flag gesetzt, definiert dieses Feld die Wartezeit in 1/100 Sekunden, ab der die Weiterbearbeitung der Datenfolge wieder aufgenommen wird. Diese Wartezeit kann durch eine Benutzereingabe abgebrochen werden. Die Wartezeit beginnt sofort nach der Ausgabe der Grafik.

Im Feld *Transparency Index* findet sich ein Bytewert. Tritt dieser Bytewert im Datenstrom für die Grafik auf, ist der betreffende Bildpunkt nicht auszugeben. Der Decoder kann zum

nächsten Bildpunkt weitergehen. Hierdurch bleibt der Bildschirmhintergrund erhalten. Dieser Index ist nur dann gültig, falls das *Transparency*-Flag gesetzt ist.

Das letzte Byte dient als Block-Terminator und besitzt den Wert 00H. Dieses Byte wird in der Längenangabe des *Extension*-Blocks nicht berücksichtigt. Im Grunde genommen handelt es sich um einen leeren Block, der generell das Ende mehrerer Subblocks definiert.

Anmerkung: Dieser Block wird für animierte GIF-Dateien benutzt, die von Browsern wie Internet Explorer oder Netscape Navigator unterstützt werden.

Der Comment Extension-Block (GIF89a)

Dieser optionale Block ist ebenfalls erst ab GIF89a definiert und kann Kommentare zur GIF-Datei (z. B. Autor, Credits etc.) enthalten. Der Block darf an jeder Stelle in der GIF-Datei auftreten, an der ein Block beginnen kann. Er sollte jedoch nicht zwischen Subblocks eingefügt werden. Empfohlen wird, den Block zu Beginn oder am Ende der GIF-Datei einzufügen. Der Inhalt des Blocks besitzt keinen Einfluß auf die GIF-Bilder. Tabelle 22.7 gibt den Aufbau des Blocks wieder.

Bytes	Bemerkungen
1	Extension-Block-Signatur (21H)
1	Comment Label (FEH)
1	Block Size
n	Kommentar als Subblocks mit 1 Länge Subblock n Datenbereich Subblock
1	Block-Terminator (00H)

Tabelle 22.7 Struktur eines Comment Extension-Blocks (GIF89a)

Das erste Byte enthält die Signatur für den *Extension*-Block. Das Byte wird fest auf den Wert 21H (entspricht !) gesetzt.

Das zweite Byte enthält die Signatur für den *Comment Extension*-Block und wird immer mit dem Code FEH belegt.

An diese Signatur schließt sich eine Sequenz von Subblocks mit dem eigentlichen Kommentartext an. Jeder Subblock enthält im ersten Byte die Zahl der folgenden Datenbytes. An dieses Byte schließen sich zwischen 0 und 255 Datenbytes an. Ist der Kommentarstring länger als ein Block, wird er auf mehrere Subblocks aufgeteilt. Das Ende des *Comment Extension*-Blocks wird durch einen Terminator markiert. Hierbei handelt es sich um einen Subblock mit einem Byte Länge, der den Wert 00H enthält.

Der Kommentarstring sollte mit 7-Bit-ASCII-Zeichen erstellt werden. Eine Darstellung multilingualer Zeichen (ä, ö, ü etc.) ist nicht vorgesehen. Der Inhalt des Blocks sollte nicht zur Speicherung von Dekodierinformationen benutzt werden, d.h., der Decoder kann den Block übergehen.

Der Plain Text Extension-Block (GIF89a)

Dieser optionale Block ist ebenfalls erst ab GIF89a definiert und kann Texte und Parameter zur grafischen Darstellung dieser Texte enthalten. Tabelle 22.8 gibt den Aufbau des Blocks wieder.

Bytes	Bemerkungen
1	Extension-Block-Signatur (21H)
1	Plain Text (01H)
1	Block Size
2	Text Grid Left Position
2	Text Grid Top Position
2	Text Grid Width
2	Text Grid Height
1	Character Cell Width
1	Character Cell Height
1	Text Foreground Color
1	Text Background Color
n	Sequenz von Text-Subblocks mit 1 Byte Länge Subblock n Byte Subblock mit Text
1	Block-Terminator (00H)

Tabelle 22.8 Struktur eines Plain Text Extension-Blocks (GIF89a)

Das erste Byte enthält die Signatur für den *Extension*-Block. Das Byte wird fest auf den Wert 21H (entspricht !) gesetzt. Das zweite Byte enthält die Signatur für den *Plain Text Extension*-Block und wird immer mit dem Code 01H belegt. Das dritte Byte enthält die Längenangabe für den folgenden Datenbereich. Beim *Plain Text Extension*-Block wird dieser Wert fest auf 12 (0CH) gesetzt. Der Text wird dann in Subblocks mit eigenen Längenangaben gespeichert.

Ab Offset 03H vom Blockanfang beginnt eine Sequenz von Feldern, die als unsigned Word interpretiert werden. Das erste Feld definiert den linken Rand (column number) für das Gitter zur Textausgabe. Die Position wird dabei in Pixel vom linken Fensterrand definiert. Das folgende Feld legt die obere Gitterposition (row number) für die Textausgabe in Pixel fest und bezieht sich ebenfalls auf den logischen Bildschirm.

Das Feld *Character Cell Width* definiert die Breite einer Gitterzelle in Pixel. Diese Gitterzelle dient zur Aufnahme eines Zeichens. In *Character Cell Height* wird die Höhe einer Gitterzelle definiert. Auch dieser Wert wird als unsigned Word angegeben. Der Decoder muß diese Werte auf die Abmessungen des virtuellen Bildschirms umrechnen, wobei das Ergebnis nur ganzzahlig sein darf.

Die letzten beiden Felder belegen jeweils nur ein Byte und geben den Farbwert (Index in die Farbpalette) für die Vordergrund- und die Hintergrundfarbe des Textes an. Der eigentliche Text wird in einer Sequenz von Subblocks gespeichert. Diese Subblocks schließen sich direkt an die obige Struktur an. Jeder Subblock enthält im ersten Byte die Zahl der folgenden Datenbytes. An dieses Byte schließen sich zwischen 0 und 255 Datenbytes mit dem auszugebenden Text an. Ist der Text länger als ein Block, wird er auf mehrere Subblocks aufgeteilt. Das Ende des *Plain Text Extension*-Blocks wird durch einen Terminator markiert. Hierbei handelt es sich um einen Subblock mit einem Byte Länge, der den Wert 00H enthält.

Zur Ausgabe des Textes wird ein Gittermuster (grid of character cells) definiert, wobei jede Gitterzelle ein einzelnes Zeichen aufnimmt. Die Parameter für das Gittermuster finden sich im *Plain Text Extension*-Block. Der Decoder muß die Parameter so umsetzen, daß die Gitterabmessungen Ganzzahlen (Integer) ergeben. Ein auftretender Nachkommanteil ist abzuschneiden. Aus Kompatibilitätsgründen sollten die Zellabmessungen mit 8 x 8 oder 8 x 16 Punkten (Breite x Höhe) gewählt werden.

Die einzelnen Zeichen sind sequentiell zu lesen und beginnend in der oberen linken Ecke des Gitters zeilenweise in die einzelnen Zellen einzutragen. Zur Anzeige ist der bestmögliche Monospace-Font mit der passenden Größe vom Decoder zu wählen. Der auszugebende Text muß mit 7-Bit-ASCII-Zeichen kodiert werden. Eine Darstellung multilingualer Zeichen (ä, ö, ü etc.) ist nicht vorgesehen. Treten Zeichencodes unterhalb 20H und oberhalb von 7FH auf, muß der Decoder ein Leerzeichen (Space, 20H) ausgeben.

Der Application Extension-Block (GIF89a)

Dieser optionale Block wurde erst in GIF89a definiert. Der Block dient zur Aufnahme anwendungsspezifischer Informationen. Tabelle 22.9 gibt den Aufbau des Blocks wieder.

Bytes	Bemerkungen
1	Extension-Block-Signatur (21H)
1	Application Extension (FFH)
1	Block Size 11
8	Application Identifier
3	Application Authentication Code

Bytes	Bemerkungen
n	Sequenz von Subblocks mit 1 Byte Länge Subblock n Byte Daten Subblock
1	Block-Terminator (00H)

Tabelle 22.9 Struktur eines Application Extension-Blocks (GIF89a)

Das erste Byte enthält die Signatur für den *Extension*-Block. Das Byte wird fest auf den Wert 21H (entspricht !) gesetzt. Das zweite Byte enthält die Signatur für den *Application Extension*-Block und wird immer mit dem Code FFH belegt.

Das dritte Byte enthält die Längenangabe für den folgenden Datenbereich. Beim *Application Extension*-Block wird dieser Wert fest auf 11 (0BH) gesetzt. Die eigentlichen Parameter werden dann in Subblocks mit eigenen Längenangaben gespeichert. Ab Offset 03H folgen 8 Byte mit dem *Application Identifier*. Dieser Identifier muß aus druckbaren ASCII-Zeichen bestehen und dient zur Bezeichnung der Anwendung, die die Daten erzeugt hat.

Anschließend folgen drei Byte für den *Application Authentication Code*. Hier kann ein Binärcode gespeichert werden, der durch die Anwendung berechnet wird. Damit ist eine eindeutige Identifizierung der erzeugenden Anwendung möglich.

An dieses Feld schließen sich die Subblocks mit den anwendungsspezifischen Daten an. Jeder Subblock beginnt mit einem Längenbyte, gefolgt von bis zu 255 Datenbytes. Der letzte Block enthält nur das Längenbyte mit dem Wert 00H und dient als Terminator.

Der GIF-Terminator

Der Abschluß einer GIF-Datei wird durch einen *Terminator*-Block markiert. Hierbei handelt es sich wieder um einen 1-Byte-Block. Dieses Byte enthält ein Semikolon (Code 3BH) als Terminator.

48 PostScript und Encapsulated PostScript (EPS)

Mit PostScript hat der Hersteller Adobe eine druckerunabhängige Beschreibungssprache geschaffen, die sich als Standard durchsetzt. Viele Softwareprodukte unterstützen deshalb den Import von PostScript-kodierten Bildern und Texten. Um einen Austausch zwischen verschiedenen Systemen zu erleichtern, verlangt Adobe bestimmte Strukturkonventionen (EPS), die von PostScript-Programmen einzuhalten sind. Abbildung 48.1 enthält eine solche Datei, die mit dem Lotus-Produkt *Freelance* erstellt wurde.

```
%!PS-Adobe-2.0 EPSF-1.2
%%Title: born1.EPS
%%Creator: Freelance Plus 3.01
%%CreationDate: 8/1/1990
%%Pages: 1
%%DocumentFonts: (atend)
%%BoundingBox: 0 0 648 468
%%EndComments
%%BeginProcSet: Freelance Plus
/FreeLance_Plus dup 100 dict def load begin
[ ] {bind} stopped { (patching the bind operator...) = flush
/*bind /bind load def /bind { dup xcheck { *bind } if } *bind def }
if pop /bdf {bind def} bind def /ldf {load def} bdf
/mt /moveto ldf /rt /rmoveto ldf /l2 /lineto ldf /c2 /curveto ldf
/sg /setgray ldf /gs /gsave ldf /ef /eofill ldf /rl2 /rlineto ldf
/st /stroke ldf /gr /grestore ldf /np /newpath ldf
/sv /save ldf /su /statusdict ldf /rs /restore ldf
/sw /setlinewidth ldf /sd /setdash ldf /cp /closepath ldf /ed
{exch def } bdf /cfnt {findfont exch makefont setfont} bdf
/itr {transform round exch round exch itransform} bdf
/fres 72 0 matrix currentmatrix dtransform
exch dup mul exch dup mul add sqrt def
/res fres def /mcm matrix currentmatrix bdf
%%EndProcSet
end
%%EndProlog
%%BeginSetup
FreeLance_Plus begin
```

```

save newpath
.1 .1 scale
/ecm matrix currentmatrix bdf
/sem {ecm setmatrix} bdf
-720 -720 translate
0 setlinecap
0 setlinejoin
1.42 setmiterlimit%%EndSetup
/Ich 256 array def StandardEncoding Ich copy pop /bullet
/paragraph/section/dieresis/tilde/ring
/circumflex/grave/acute/quotedblleft/quotesingle
/ellipsis/endash/emdash/guilsinglleft/guilsinglright
/quotedblbase/quotesinglbase/quotedblright/OE/oe
/Ydieresis/fi/fl/dagger/daggerdbl/Ccedilla/udieresis
/eacute/acircumflex/adieresis/agrave/aring/ccedilla
/ecircumflex/edieresis/egrave/idieresis/icircumflex
/igrave/Adieresis/Aring/Eacute/ae/AE/ocircumflex
/odieresis/ograve/ucircumflex/ugrave/ydieresis
/Odieresis/Udieresis/oslash/sterling/Oslash/florin
/aacute/iacute/oacute/uacute/ntilde/Ntilde/ordfeminine
/ordmasculine/questiondown/exclamdown/guillemotleft
/guillemotright/Aacute/Acircumflex/Agrave/cent/yen
/atilde/Atilde/currency/Ecircumflex/Edieresis/Egrave
/dotlessi/Iacute/Icircumflex/Idieresis/Igrave/Oacute
/germandbls/Ocircumflex/Ograve/otilde/Otilde/Uacute
/Ucircumflex/Ugrave/macron/cedilla/periodcentered
Ich 127 97 getinterval astore pop
/Ienc { /ncs Ich def /nfn ed /bfn ed /bfd bfn findfont def
/nf bfd maxlength dict def bfd{exch dup dup /FID ne exch
/Encoding ne and {exch nf 3 1 roll put}{pop pop} ifelse }
forall nf/FontName nfn put nf/Encoding ncs put nfn nf
definefont pop}bfd
/Ienc0 { /ncs Ich def /nfn ed /bfn ed /lnw ed /bfd bfn
findfont def /nf bfd maxlength 4 add dict def bfd{exch dup
dup /FID ne exch /Encoding ne and
{exch nf 3 1 roll put}{pop pop} ifelse }forall

```

```

nf/FontName nfn put nf/Encoding ncs put
nf/PaintType 2 put nf/StrokeWidth lnw put
nfn nf definefont pop}bdf
/IencSO { /nfn ed /bfn ed /lnw ed /bfd bfn findfont def
/nf bfd maxlength 4 add dict def bfd{exch dup
/FID ne { exch nf 3 1 roll put}{pop pop} ifelse }forall
nf/FontName nfn put nf/PaintType 2 put
nf/StrokeWidth lnw put nfn nf definefont pop}bdf
/Helvetica /Flfon1 Ienc
[191 0 0.00 191 -18 -142] /Flfon1 cfnt
0.000 sg
2737 3026 itr mt
(Dies ist ein Text)
show
6 sw
sv np [] 0 sd
2764 3749 itr mt
4071 3749 itr l2
st rs
sv np
1352 2572 itr mt
2424 2572 itr l2
2424 3513 itr l2
1352 3513 itr l2
1352 2572 itr l2
cp
st rs
sv np
3456 4438 itr mt
currentpoint translate
np 1.000 1.000 scale
0 0 549 0.000 360.000 arc sem
st rs
sv np
5156 4438 itr mt
5823 4573 itr l2

```

```
5823 3984 itr l2
4973 3984 itr l2
4973 4354 itr l2
5156 4438 itr l2
gs 0.000 sg
ef gr
cp
st rs
rs end
%%Trailer
%%DocumentFonts: Helvetica
```

Abbildung 48.1 EPS-Datei, erzeugt mit Freelance Plus

Die EPS-Datei besteht aus den eigentlichen PostScript-Anweisungen und den Strukturkonventionen zur Programmsteuerung.

Die EPS-Strukturkonventionen

Die Strukturkonventionen von EPS sehen für jedes Programm einen *Prolog* und ein *Script* vor. Im Prolog finden sich anwendungsabhängige Definitionen, während im Script die eigentlichen PostScript-Anweisungen stehen. Die Teile werden durch verschiedene Strukturinformationen beschrieben, die als PostScript-Kommentare mit dem Zeichen % eingeleitet werden. Der Interpretierer ignoriert dann die komplette Zeile, während ein lesendes Programm die darin enthaltenen Informationen auswerten kann.

Um die Strukturinformationen von anderen Kommentarzeilen zu unterscheiden, beginnen die Infos mit den Zeichen:

```
%!  
%%
```

gefolgt von einem beliebigen ASCII-String. Eine EPS-Datei beginnt immer mit der folgenden Zeile, die eine Datei als PostScript-Datei gemäß den Adobe-Konventionen auszeichnet:

```
%!PS-Adobe-2.0 EPSF-1.2
```

Weiterhin enthält die Zeile noch das Schlüsselwort:

```
PS-Adobe-
```

gefolgt von einer Versionsnummer.

Header Comments

Unmittelbar nach der Versionszeile finden sich weitere Kommentare mit zusätzlichen Informationen.

```
%%DocumentFonts: font1, font2 ...
```

Mit diesem Kommentar lassen sich die benutzten Fontnamen angeben. Diese Anweisung kann aber auch an das Ende der Datei verschoben werden. Dann muß im Header folgende Zeile erscheinen (Abbildung 48.1):

```
%%DocumentFonts: (atend)
```

Diese Anweisung legt den Dokumententitel fest:

```
%%Title: title
```

Hier wird häufig der Dateiname mitgespeichert. Das dateierzeugende Programm oder die zuständige Person sowie das Erzeugungsdatum sind in diesen Zeilen notiert:

```
%Creator: text
```

```
%%CreationDate: datum
```

Im Datum kann auch die Zeit als String mit aufgenommen werden. Diese Anweisung benennt einen Empfänger für das Dokument:

```
%%For: text
```

Die Zahl der Seiten wird mit der nächsten Sequenz gesetzt:

```
%%Pages: xx
```

Der Wert *xx* sollte eine nicht negative Dezimalzahl umfassen und die Zahl der auf dem Drucker auszugebenden Seiten spezifizieren. Erzeugt das Programm keine Seiten, ist eine 0 einzutragen. Die Spezifikation *attend* ist ebenfalls erlaubt.

Die Bounding-Box umschließt alle Markierungen des Programmes mit einem Rechteck. Die nächste Anweisung enthält als Parameter vier Integerwerte, die die linke obere und die rechte untere Ecke der Bounding-Box im Initialkoordinatensystem angeben:

```
%%BoundingBox: 0 0 648 468
```

Folgende Anweisung schließt den Header ab:

```
%%EndComments
```

Body Comments

Die auftretenden Kommentare dienen im wesentlichen zur Markierung verschiedener Abschnitte im eigentlichen Programmteil. Nachfolgend findet sich eine Zusammenstellung möglicher Kommentare (aus Abbildung 48.1):

```
%%BeginProcSet:  
%%EndProcSet  
%%EndProlog  
%%BeginSetup  
%%EndSetup
```

Der Prolog wird durch einen entsprechenden Kommentar (*EndProlog*) abgeschlossen. Der Beginn der Beschreibung einer Seite läßt sich durch diese Sequenz markieren:

```
%%Page: label ordial
```

In *label* und *ordial* werden die entsprechenden Nummern eingetragen. Zeichensätze für einzelne Seiten, die durch ein Leseprogramm ausgewertet werden können, lassen sich so spezifizieren:

```
%%PageFonts: font1, font2 ...
```

Trailer Comments

Den Abschluß einer EPS-Datei bildet der Anhang (*Trailer*), eingeleitet durch den Kommentar

```
%%Trailer
```

Falls im Kopf die Anweisung mit *attend* an das Ende verschoben wurde, können mit der folgenden Anweisung noch die im Dokument verwendeten Zeichensätze aufgeführt werden:

```
%%DocumentFonts: font1, font2 ...
```

Entsprechende Beispiele finden sich in Abbildung 48.1.

PostScript-Anweisungen (Level 1)

Die Sprache PostScript arbeitet nach dem Prinzip der umgekehrten polnischen Notation und besitzt verschiedene Anweisungen zur Ausgabe von Schrift und Grafik. Für den Umgang mit der Sprache seien die von Adobe bei Addison Wesley publizierten Titel (z. B. PostScript Language Reference) als Lektüre empfohlen. Nachfolgend werden die wichtigsten PostScript-Anweisungen kurz beschrieben.

abs

Format: num abs

Wandelt den Inhalt von *num* in eine absolute Zahl um und legt das Ergebnis auf dem Stack ab.

add

Format: num1 num2 add

Addiert die beiden Parameter *num1* und *num2* und legt das Ergebnis auf dem Stack ab.

and

Format: bool1 bool2 and

Verknüpft die beiden logischen Werte *bool1* und *bool2* über die And-Funktion und legt das Ergebnis auf dem Stack ab.

arc

Format: x y r ang1 ang2 arc

Zeichnet am Punkt x,y einen Kreisbogen mit dem Radius r . Die Parameter *ang1* und *ang2* geben die Winkel des Kreisbogens an.

arcn

Format: x y r ang1 ang2 arcn

Zeichnet am Punkt x,y im Uhrzeigersinn einen Kreisbogen mit dem Radius r . Die Parameter *ang1* und *ang2* geben die Winkel des Kreisbogens an.

arcto

Format: x1 y1 x2 y2 arcto

Zeichnet vom aktuellen Punkt x_0,y_0 eine Gerade zum Punkt x_1,y_1 und von dort einen Kreisbogen zum Punkt x_2,y_2 . Nach dem Aufruf stehen auf dem Stack vier Parameter mit den Tangentenpunkten.

ashow

Format: ax ay string ashow

Gibt einen Text (*string*) aus, wobei die Dicken mit *ax* und *ay* korrigiert werden.

atan

Format: num den atan0

Berechnet aus dem Parameter *num* den Winkel in Grad und legt das Ergebnis auf dem Stack ab.

awidthshow

Format: cx cy char ax ay string awidthshow

Gibt einen Text (*string*) aus, wobei die Dicken mit *ax* und *ay* korrigiert werden. Zusätzlich wird jedes Zeichen *char* im Text mit den Parametern *cx* und *cy* korrigiert.

bind

Format: `proc bind`

Ersetzt den Operatornamen einer Prozedur durch deren Werte und legt sie auf dem Stack ab.

bitshift

Format: `int1 shift bitshift`

Verschiebt den Wert von *int1* um *shift* nach links. Ist der Wert negativ, werden die Bits nach rechts verschoben. Das Ergebnis findet sich anschließend auf dem Stack.

ceiling

Format: `num ceiling`

Liefert den nächsten benachbarten Wert von *num* auf dem Stack zurück.

charpath

Format: `string bool charpath`

Liefert den Pfad, der durch die Anwendung von *show* erzeugt wurde.

clip

Format: `clip`

Schneidet die inneren Flächen eines Clipping-Pfades aus.

clippath

Format: `clippath`

Definiert den aktuellen Clipping-Pfad.

closepath

Format: `closepath`

Schließt den aktuellen Subpfad durch ein gerades Liniensegment vom aktuellen Punkt zum Startpunkt.

copypage

Format: `copypage`

Gibt den Inhalt der aktuellen Seite auf dem Drucker aus, ohne die Seite neu zu initialisieren. Dadurch kann ein Teilbild probeweise ausgegeben werden.

cos

Format: angle cos

Berechnet den Cosinuswert des Winkels und gibt das Ergebnis auf dem Stack zurück.

currentpoint

Format: currentpoint

Liefert die aktuellen XY-Koordinaten auf dem Stack zurück.

curveto

Format: x1 y1 x2 y2 x3 y3 curveto

Zeichnet eine Bézierkurve durch die drei angegebenen Punkte.

cvi

Format: num cvi

Konvertiert den Wert von *num* in eine Integergröße und gibt diese auf dem Stack zurück.

cvn

Format: string cvn

Konvertiert eine Zahl in Form eines Strings in eine Integergröße und gibt diese auf dem Stack zurück.

cvr

Format: string cvr

num cvr

Konvertiert den Wert von *num* oder *string* in eine Fließkommagröße und gibt diese auf dem Stack zurück.

cvrs

Format: num string radix cvrs

Konvertiert eine Zahl *num* in einen String mit der Basis *radix* und gibt das Ergebnis auf dem Stack zurück.

cvS

Format: num string cvS

Konvertiert den Wert eines beliebigen Elements in einen Text und gibt diesen auf dem Stack zurück.

div

Format: num1 num2 div

Dividiert die Zahl *num1* durch *num2* und legt das Ergebnis auf dem Stack ab.

dup

Format: element dup

Dupliziert das oberste Element des Stacks.

eoclip

Format: eoclip

Schneidet die inneren Flächen des aktuellen Clipping-Pfades mit den inneren Flächen des aktuellen Pfades.

eofill

Format: eofill

Füllt die inneren Flächen des aktuellen Pfades mit der aktuellen Farbe aus.

eq

Format: elem1 elem2 eq

Vergleicht zwei beliebige Elemente miteinander und legt als Ergebnis *True* oder *False* auf dem Stack ab.

exch

Format: elem1 elem2 exch

Vertauscht die beiden obersten Stackelemente.

exec

Format: oper exec

Legt den Operanden auf den Stack und führt ihn direkt aus.

exit

Format: exit

Beendet die Ausführung einer Schleife (*for*, *loop*, *repeat*, *forall*).

exp

Format: base exp exp

Berechnet aus der Basis *base* und dem Exponenten *exp* einen Fließkommawert.

fill

Format: fill

Füllt die Fläche des aktuellen Pfades mit der aktuellen Farbe.

findfont

Format: font findfont

Sucht einen durch *font* beschriebenen Namen aus dem Font-Dictionary.

for

Format: start step end { proc } for

Konstruiert eine Schleife, die mit dem Wert von *start* beginnt. Der Schleifenindex wird mit der Schrittweite *step* so lange erhöht, bis der Endwert *end* erreicht ist. Bei jedem Durchlauf werden die in den geschweiften Klammern ({}) stehenden Anweisungen ausgeführt.

ge

Format: num1 num2 ge

Führt zwischen *num1* und *num2* einen Vergleich auf *greater equal* aus und legt das Ergebnis als Boole-Wert auf dem Stack ab.

grestore

Format: grestore

Restauriert den mit *gsave* gesicherten Grafikstatus.

gsave

Format: gsave

Sichert den Grafikstatus.

gt

Format: num1 num2 gt

Führt zwischen *num1* und *num2* einen Vergleich auf *greater* aus und legt das Ergebnis als Boole-Wert auf dem Stack ab.

idiv

Format: int1 int2 idiv

Führt eine Integerdivision zwischen *int1/int2* aus und legt das Ergebnis auf dem Stack ab.

if

Format: `bool { proc } if`

Bearbeitet die in den Klammern stehenden Anweisungen, falls *bool* gleich *True* ist.

ifelse

Format: `bool { proc1 } { proc2 } ifelse`

Bearbeitet die in den Klammern stehenden Anweisungen *proc1*, falls:

`bool = True`

andernfalls werden die Anweisungen *proc2* ausgeführt.

image

Format: `width height bits_per_sample matrix proc image`

Überträgt ein Rasterbild auf die aktuelle Seite. Die Parameter *width* und *height* spezifizieren die Abmessungen des Bildes; mit *proc* werden die Bilddaten eingelesen.

imagemask

Format: `width height invert matrix proc imagemask`

Überträgt ein Rasterbild auf die aktuelle Seite. Die Parameter *width* und *height* spezifizieren die Abmessungen des Bildes, mit *proc* werden die Bilddaten eingelesen. Der logische Wert *invert* definiert dabei, ob die Bits direkt oder invertiert (*0 Bits zeichnen -> invert = False*) gezeichnet werden.

index

Format: `a1 a2...ax n index`

Kopiert das *n-te* Stackelement oben auf den Stack, gezählt vom obersten Element (0).

le

Format: `num1 num2 le`

Führt zwischen *num1* und *num2* einen Vergleich auf *less or equal* aus und legt das Ergebnis als Booleschen Wert auf dem Stack ab.

lineto

Format: `x y lineto`

Fügt an den aktuellen Pfad ein gerades Liniensegment von der aktuellen Position zum Punkt *x,y* an.

ln

Format: num1 ln

Liefert den Fließkommawert von $\ln(num1)$ auf den Stack zurück.

log

Format: num1 log

Liefert den Fließkommawert von $\log(num1)$ auf den Stack zurück.

loop

Format: { proc } loop

Führt die Anweisungen der Schleife { *proc* } so lange aus, bis ein *exit*-Operator geladen wird.

lt

Format: num1 num2 lt

Führt einen Vergleich auf *less than num1* und *num2* aus und legt das Ergebnis als Boole-Wert auf dem Stack ab.

mod

Format: int1 int2 mod

Liefert den Rest der Division von $int1/int2$ auf dem Stack zurück (Modulofunktion).

moveto

Format: x y moveto

Beginnt einen neuen Pfad und setzt den aktuellen Punkt auf die mit *x,y* angegebenen Koordinaten.

mul

Format: num1 num2 mul

Liefert das Ergebnis der Multiplikation von *num1* und *num2* auf dem Stack ab.

neg

Format: num1 neg

Invertiert den Wert von *num1* und legt das Ergebnis auf dem Stack ab.

newpath

Format: newpath

Legt einen neuen Pfad auf der aktuellen Seite an; der aktuelle Punkt ist dann undefiniert.

not

Format: bool1 not

Invertiert den Wert der logischen Variablen und gibt das Ergebnis auf dem Stack zurück.

or

Format: bool1 bool2 or

Verknüpft die beiden Operatoren mit *or* und liefert das Ergebnis auf dem Stack ab.

pop

Format: pop

Entfernt das oberste Element vom Stack.

quit

Format: quit

Beendet die Arbeit des Interpreters.

rand

Format: rand

Liefert eine Zufallszahl (Integer) auf dem Stack zurück.

rcurveto

Format: dx1, dy1 dx2 dy2 dx3 dy3 rcurveto

Zeichnet eine Bézierkurve vom aktuellen Pfad relativ zu den Koordinatenangaben *dx,dy*.

repeat

Format: count { proc } repeat

Führt die Anweisungen in *proc count*-mal aus.

rlinto

Format: dx dy rlineto

Erzeugt einen neuen Subpfad durch relative Verschiebung um die Strecke *dx,dy*.

roll

Format: elem1...elemx n j roll

Verschiebt *n* Elemente des Stacks um *j* Positionen im Kreis.

round

Format: num1 round

Rundet den Wert *num1* auf den nächstliegenden Integerwert.

scale

Format: x y scale

Verändert die Skalierung der x- und y-Achsen.

scalefont

Format: font scale scalefont

Skaliert die Größe eines Zeichensatzes vor der Ausgabe.

search

Format: string such search

Sucht *string* nach der Zeichenkette *such* ab und legt das Ergebnis als:

Reststring, such, Anfangsteil, bool

auf dem Stack ab. Der oberste Wert auf dem Stack gibt an, ob der String gefunden wurde (*bool = True*).

setfont

Format: font setfont

Aktiviert den angegebenen Zeichensatz als aktuellen Font.

setgray

Format: num setgray

Legt die Schwärzung für die Zeichenoperationen fest. Der Wert von *num* kann zwischen 0 (Weiß) und 1 (Schwarz) liegen.

setlinewidth

Format: num setlinewidth

Setzt die Breite von Linien auf einen definierten Wert.

show

Format: (text) show

Gibt den auf dem Stack stehenden Text im aktuellen Pfad aus.

showpage

Format: showpage

Kopiert die aktuelle Seite auf das Ausgabegerät.

sin

Format: angle sin

Berechnet den Sinus des angegebenen Winkels und gibt das Ergebnis als Fließkommawert auf dem Stack zurück.

sqrt

Format: num sqrt

Berechnet die Wurzel aus *num* und gibt das Ergebnis als Fließkommawert auf dem Stack zurück.

string

Format: int string

Erzeugt ein Stringobjekt der Länge *int*.

stringwidth

Format: string stringwidth

Ermittelt die Länge des angegebenen Strings.

stroke

Format: stroke

Zieht entlang des aktuellen Pfades eine Linie. Erst damit werden Objekte in der aktuellen Farbe gezeichnet.

sub

Format: num1 num2 sub

Subtrahiert *num1-num2* und liefert das Ergebnis auf dem Stack ab.

truncate

Format: num truncate

Entfernt den Nachkommateil von *num* und liefert das Ergebnis auf dem Stack zurück.

xor

Format: bool1 bool2 xor

Verknüpft die beiden Werte *bool1* und *bool2* mit Xor und legt das Ergebnis auf dem Stack ab.

PostScript Level 2

PostScript Level 2 stellt eine Erweiterung des Befehlsvorrats in Level 1 dar. Neben verbesserten Möglichkeiten zur Komprimierung von PostScript-Programmen wurden neuere Verfahren implementiert:

- ▶ Verbesserungen in der Speicherverwaltung und Definition eines Dictionary-Operands.
- ▶ Einführung von Resources (Sammlung von benannten Objekten) und Filtern zum Import/Export von Dateien.
- ▶ Binäre Kodierung der Programme zur optimalen Übertragung der Programme; User Paths, um die Pfade für Operatoren zu definieren.
- ▶ Befehle, um Formulare (FORMS) zu erzeugen und wiederzuverwenden; Definition von Mustern (Pattern).
- ▶ Definition verschiedener Farbräume, um Farbausgaben unter PostScript zu unterstützen; Unterstützung des CIE-Farbmodells und Befehle zur Ausgabe von Bildern.
- ▶ Erweiterungen in den Text- und Grafikoperatoren; Parameter für den Interpreter und Device-Setup-Befehle.

Nachfolgend finden Sie eine kurze Übersicht über die neuen Befehle in Level 2.

<< mark

Der Befehl legt das Objekt *mark* auf dem Operandenstack ab; dieser Befehl verhält sich wie der *mark*- oder *[-*-Operator.

>> mark

Format: `mark key1 value1 keyn valuen >> dict`

Erzeugt ein Dictionary mit den spezifizierten *key*-Werten. Als Operanden werden *mark* und eine gerade Anzahl von Operanden (*key*) erwartet. Diese Schlüssel werden in das Dictionary eingefügt. Der folgende Befehl:

```
<< /Duplex true /PageSize [612 792] /Collate false >> setpagedevice
```

erzeugt ein Dictionary mit drei Einträgen, welches sofort zum *setpagedevice*-Operator weitergegeben wird.

arct

Format: `x1 y1 x2 y2 r arct`

Der Befehl fügt ein Kreissegment an den aktuellen Pfad an. Vorher kann eine gerade Linie auftreten, damit ein Kreissegment erzeugt wird.

Die Punkte $x_1 y_1$ und $x_2 y_2$ definieren die Ecken, in die der Kreis einzufügen ist. Vom Startpunkt $x_0 y_0$ wird ein Pfad zu $x_1 y_1$ und dann zu $x_2 y_2$ konstruiert. Anschließend erzeugt der Befehl ein Kreissegment mit dem Radius r an den Tangenten. Letztlich erzeugt

der Befehl eine abgerundete Ecke in einem Viereck. Das folgende Beispiel verdeutlicht die Anwendung des Befehls:

```
newpath
0 0 moveto          % auf Startpunkt
0 10 10 10 4 arct  % runde Ecke zeichnen
10 10 lineto        % Linie zum Endpunkt zeichnen
```

Der Pfad wird dabei vom Startpunkt bis zum angegebenen Endpunkt konstruiert.

cshow

Format: `proc string cshow`

Ruft die Prozedur *proc* für jede Operation im Font-Mapping-Algorithmus einmal auf. Beim Aufruf von *proc* enthält der Stack den selektierten Zeichencode und die x-, y-Koordinaten des Vektors für die Ausgabe. *cshow* gibt keine Zeichen aus und verändert auch nicht die aktuelle Zeichenposition. Die Ausgabe des Textes erfolgt über *proc*, d.h., der Befehl läßt sich zur Ausgabe und Positionierung spezieller Zeichen verwenden.

currentblackgeneration

Format: `currentblackgeneration proc`

Gibt die aktuelle *blackgeneration*-Funktion für den Grafikstatus zurück.

currentcacheparams

Format: `currentcacheparams mark size lower upper`

Legt ein *mark*-Objekt, gefolgt von den aktuellen Cacheparametern, auf dem Operandenstack ab. Die Parameterzahl ist dabei variabel.

currentcmykcolor

Format: `currentcmykcolor cyan magenta yellow black`

Gibt die aktuelle Farbe des Grafikstatus im CMYK-Farbsystem auf dem Stack zurück.

Ist das Farbsystem auf *DeviceCMYK* gesetzt, gibt der Befehl die Farben zurück, die mit *setcmykcolor* oder *setcolor* gesetzt wurden. Bei *DeviceRGB* oder *DeviceGray* werden die Farbwerte erst in den CMYK-Farbraum konvertiert.

currentcolor

Format: `currentcolor comp1 comp2 compn`

Der Befehl ermittelt die aktuellen Farbkomponenten und gibt diese auf dem Stack zurück.

currentcolorrendering

Format: `currentcolorrendering dict`

Der Befehl gibt den Wert des CIE-Color-Rendering-Dictionary zurück.

currentcolorscreen

Format: `currentcolorscreen redfreq redang redproc
greenfreq greenang greenproc
bluefreq blueang blueproc
grayfreq grayang grayproc`

Der Befehl gibt alle 12 Halbton-Bildparameter zurück, falls der aktuelle Halbton-Screen mit *setcolorscreen* oder *currentcolorscreen* definiert wurde. Wurde der Halbton-Screen mit *sethalftone* oder *currentcolorscreen* definiert, gibt der Befehl die Werte 60,0 und das Halftone Dictionary viermal zurück.

currentcolorspace

Format: `currentcolorspace array`

Der Befehl gibt ein Feld mit den Werten des Farbraums zurück. Dieses Feld kann zum Beispiel den Namen des Farbraums enthalten.

currentcolortransfer

Format: `currentcolortransfer redproc greenproc blueproc grayproc`

Mit dem Befehl werden die aktuellen Transferprozeduren für die vier Farben zurückgegeben. Wurden die Funktionen mit *setcolortransfer* oder *currentcolortransfer* definiert, enthält der Stack 4 Operanden. Mit *settransfer* gibt *currentcolortransfer* nur eine Funktion zurück.

currentdevparams

Format: `string currentdev currentdevparams dict`

Der Befehl gibt das Dictionary mit allen Keys und aktuellen Werten der Einheit (Device) zurück, die durch *string* angegeben wurde.

currentglobal

Format: `currentglobal bool`

Der Befehl gibt den aktuellen VM Allocation Mode zurück.

currentgstate

Format: `gstate currentgstate gstate`

Der Befehl ersetzt den aktuellen Wert von *gstate* durch eine Kopie des aktuellen Grafikstatus. Weiterhin wird *gstate* auf dem Operandenstack hinterlassen.

currenthalftone

Format: `currenthalftone halftone`

Der Befehl gibt das aktuelle Halftone Dictionary des Grafikstatus zurück.

currentobjectformat

Format: `currentobjectformat int`

Der Befehl gibt den aktuellen *Object Format*-Parameter zurück.

currentoverprint

Format: `currentoverprint bool`

Gibt den Wert des *Overprint*-Parameters zurück.

currentpacking

Format: `currentpacking bool`

Vom Befehl wird das Feld *Packing Mode* zurückgegeben.

currentpagedevice

Format: `currentpagedevice dict`

Der Befehl gibt ein *Dictionary* (read only) zurück, welches ein seitenorientiertes Ausgabegerät beschreibt. Der Befehl erzeugt bei Bedarf ein neues Dictionary. Ist das aktuelle Ausgabegerät nicht seitenorientiert, ist das zurückgegebene Dictionary leer.

currentshared

Format: `currentshared bool`

Der Befehl besitzt die gleiche Semantik wie *currentglobal* und wird aus Kompatibilitätsgründen beibehalten.

currentstrokeadjust

Format: `currentstrokeadjust bool`

Der Befehl gibt den aktuellen Grafikstatus-Parameter *currentstrokeadjust* zurück.

currentsystemparams

Format: `currentsystemparams dict`

Gibt ein Dictionary mit den aktuellen Systemparametern der Implementierung zurück.

currentundercolorremoval

Format: `currentundercolorremoval proc`

Vom Befehl wird die aktuelle *colorremoval*-Funktion zurückgegeben.

currentuserparams

Format: `currentuserparams dict`

Gibt das Dictionary mit den aktuellen Benutzerparametern zurück.

defineuserobject

Format: `index any defineuserobject`

Erzeugt eine Verknüpfung zwischen der positiven Integerzahl *index* und dem Objekt *any* im *UserObject*. Der Befehl speichert ein Objekt *any* an die in *index* angegebene Position im *array* des *UserObject*.

deletefile

Format: `filename deletefile`

Löscht die angegebene Datei aus dem Speicher. Gibt es die Datei nicht, löst der Interpreter einen *undefinedfilename*-Fehler aus.

execuserobject

Format: `index execuserobject`

Der Befehl führt das User-Object aus, welches in *index* angegeben wird. Das Objekt muß mit *defineuserobject* definiert worden sein, sonst erfolgt eine Fehlermeldung.

filenameforall

Format: `template proc scratch filenameforall`

Der Befehl führt alle Dateien auf, deren Namen mit dem in *template* angegebenen Muster übereinstimmen. Die Dateinamen werden in den angegebenen *scratch*-String kopiert, und dann wird die angegebene Prozedur *proc* aufgerufen. Die Zeichen im Parameter *template* dürfen auch Wildcards * ? enthalten. Wobei * mehrere Zeichen ersetzt und ? für ein Wildcard-Zeichen steht. Beginnt ein Zeichen mit \, wird das folgende Zeichen als Literal verwendet. Beginnt *template* mit %, erfolgt der Vergleich auf komplette Dateinamen in der Form %device%file.

Für den Inhalt der Prozedur *proc* gibt es keine Einschränkungen. Bei jedem gefundenen Dateinamen wird diese Prozedur aufgerufen.

filenameposition

Format: `file filenameposition position`

Der Befehl gibt die aktuelle Position innerhalb der angegebenen Datei in *position* (positive Integerzahl) zurück.

filter

Format: `scr|tgt para1 ... paran filter file`

Vom Befehl wird eine gefilterte Datei erzeugt und in *file* zurückgegeben. Der *scr | tgt*-Operand definiert die unterlagerte Datenquelle, aus der der Filter die Daten liest. Die Quelle kann eine Datei, ein String oder eine Prozedur sein. Die Parameter *para_x* definieren die Anpassung des speziellen Filters. Meist benötigen die Filter jedoch keine Parameter. Tabelle 48.1 gibt die Namen der Standardfilternamen an:

Name	Name
ASCIHexEncode	ASCIHexDecode
ASCII85Encode	ASCII85Decode
LZWEncode	LZWDecode
RunLengthEncode	RunLengtDecode
CCITTFaxEncode	CCITTFaxDecode
NullEncode	NullDecode

Tabelle 48.1 Standardfilter

findencoding

Format: `key findencoding array`

Der Befehl übernimmt den in *key* angegebenen Encoding-Vektor und legt diesen auf dem Stack ab.

findresource

Format: `key category findresource instance`

Der Befehl versucht, eine benannte Ressourceninstanz der spezifizierten Kategorie zu übernehmen. *category* ist ein Name oder ein Stringobjekt, welches die Ressourceninstanz angibt (z. B. ein Font). Der Befehl gibt eine Instanz der Ressource auf dem Stack zurück.

gcheck

Format: `any gcheck bool`

Mit dem Befehl läßt sich prüfen, ob der angegebene Operand *any* zu den grundlegenden PostScript-Befehlen gehört oder in der globalen VM als zusammengesetzter Operand existiert. In diesen Fällen wird der Wert *true* zurückgegeben. Der Wert *false* wird zurückgegeben, falls ein zusammengesetzter Operand in der lokalen VM vorhanden ist.

globaldict

Format: `globaldict dict`

Der Befehl legt das Objekt *globaldict* auf dem Operandenstack ab.

GlobalFontDirectory

Format: GlobalFontDirectory dict

Mit dem Befehl wird ein Dictionary oder ein definierter Font auf dem Operandenstack hinterlegt.

glyphshow

Format: name glyphshow

Der Befehl zeigt ein einzelnes Zeichen aus dem aktuellen Font. Das Zeichen ist in *name* anzugeben. *glyphshow* umgeht dabei die aktuelle *Font-Encoding*-Funktion und kann damit auf alle Zeichen eines Fonts zugreifen. Die Wirkungsweise des Operators ist aber abhängig vom verwendeten Font (Type 1, Type 3).

ineofill

Format: x y ineofill bool
userpath ineofill bool

Der Befehl besitzt zwei Varianten und ist vergleichbar mit *infill*. Der Test wird lediglich auf *eofill* und nicht auf *fill* durchgeführt.

infill

Format: x y infill bool
userpath infill bool

Der Befehl besitzt zwei Varianten. Der erste Befehl gibt den Wert *true* zurück, falls der Bildpunkt *x,y* im Benutzerkoordinatensystem bei Anwendung des *fill*-Kommandos gezeichnet wird. Im zweiten Fall gibt der Befehl den Wert *true* zurück, falls Punkte im *user-path* bei Anwendung des *fill*-Operators gezeichnet werden.

instroke

Format: x y instroke bool
userpath instroke bool

Der Befehl besitzt zwei Varianten. Der erste Befehl arbeitet ähnlich wie *infill*, wobei der Test sich jedoch auf die Funktion *stroke* bezieht.

inueofill

Format: x y userpath inueofill bool
userpath1 userpath2 inueofill bool

Der Befehl besitzt zwei Varianten und wirkt ähnlich wie *inufill*, wobei sich der Test auf den Befehl *inueofill* bezieht.

inufill

Format: x y userpath inufill bool
userpath1 userpath2 inufill bool

Der Befehl besitzt zwei Varianten. Der erste Befehl gibt den Wert *true* zurück, falls der Bildpunkt *x,y* im Benutzerkoordinatensystem bei Anwendung des *ufill*-Kommandos im *Userpath* gezeichnet wird. Im zweiten Fall werden Punkte in *userpath1* als Appertur interpretiert. Dann gibt der Befehl den Wert *true* zurück, falls Punkte der Appertur im *userpath2* bei Anwendung des *ufill*-Operators gezeichnet werden.

inustroke

Format: x y userpath inustroke bool
x y userpath matrix inustroke bool
userpath1 userpath2 inustroke bool
userpath1 userpath2 matrix inustroke bool

Der Befehl besitzt mehrere Varianten. Der erste Befehl testet, ob ein Bildpunkt *x,y* auf dem aktuellen Device mit *inustroke* gezeichnet wird, und gibt dann den Wert *true* zurück. Im zweiten Befehl wird nach der Interpretation des User-Pfads die *Matrix* zur CTM (coordinate transfer matrix) hinzugefügt. Im dritten und vierten Befehl werden die Pixel im *userpath1* als *Appertur* interpretiert. Wird ein solches Pixel von *ustroke* gezeichnet, gibt *inustroke* den Wert *true* zurück.

ISOLatin1Encoding

Format: ISOLatin1Encoding array

Der Befehl legt einen ISO-Latin1-Encoding-Vektor auf dem Operandenstack ab. Dieser Vektor ist ein Array mit 256 Einträgen, welches durch die Zeichencodes indiziert wird.

languagelevel

Format: languagelevel int

Der Befehl ermittelt die Sprachversion des PostScript-Interpreters und gibt diese auf dem Stack zurück. Der Wert 2 signalisiert, daß alle Level 2-Befehle implementiert sind.

makepattern

Format: dict matrix makepattern pattern

Der Befehl erzeugt ein *Muster*. Zuerst wird geprüft, ob *dict* ein Pattern-Dictionary ist, welches alle erforderlichen Werte enthält. Dann wird eine Kopie in der lokalen VM erzeugt. Der Inhalt von *matrix* wird in der CTM gesichert. Als Resultat wird das *Muster-Dictionary* auf dem Stack zurückgegeben. Das *Muster* ist dann über *setpattern* aus dem Dictionary abrufbar.

packedarray

Format: any₀ ... any_n n packedarray packedarray

Der Befehl erzeugt aus den Einträgen any₀ ... any_n ein gepacktes Array der Länge *n* und legt es auf dem Operandenstack ab.

printobject

Format: obj tag printobject

Schreibt eine binäre Objektsequenz in die Standard-Ausgabedatei. Die binäre Objektsequenz enthält ein Top-Level Array der Länge 1. Dieses Element stellt die Kodierung für *obj* dar. *tag* ist ein Wert zwischen 0 und 255 und bezieht sich auf das *top level object*.

product

Format: product string

Mit dem Befehl läßt sich ein *string* (read only) abrufen, der die Umgebung des PostScript-Interpreters angibt.

realtime

Format: realtime int

Der Befehl erlaubt die Abfrage der Uhrzeit, die in Millisekunden in *int* zurückgegeben wird.

rectclip

Format: x y width height rectclip
numarray rectclip
numstring rectclip

Der Befehl schneidet ein Rechteck aus dem aktuellen Clipping-Pfad. In der ersten Variante wird ein Rechteckpfad anhand des Startpunkts und der Abmessungen erzeugt. In den beiden anderen Varianten wird der Operand als *array* oder als *string* angegeben. Damit lassen sich mehrere Rechtecke angeben. Nach der Berechnung des Clipping-Pfades wird *current path* wie bei *newpath* auf *empty* gesetzt.

rectfill

Format: x y width height rectfill
numarray rectfill
numstring rectfill

Der Befehl besitzt mehrere Varianten. Mit diesem Befehl läßt sich ein Pfad, bestehend aus einem oder mehreren Rechtecken, in der aktuellen Farbe füllen. Die Parameter besitzen die gleiche Bedeutung wie bei *rectclip*.

rectstroke

Format: x y width height rectstroke
x y width height matrix rectstroke
numarray rectstroke
numarray matrix rectstroke
numstring rectstroke
numstring matrix rectstroke

Der Befehl besitzt mehrere Varianten. Er führt einen *stroke*-Befehl auf einen Pfad mit einem oder mehreren Rechtecken aus. Ist der Parameter *matrix* vorhanden, fügt der Interpreter den Inhalt in der CTM ein. Die restlichen Parameter besitzen die Bedeutung wie bei *rectclip*.

rectviewclip

Format: x y width height rectviewclip
numarray rectviewclip
numstring rectviewclip

Der Befehl ersetzt den aktuellen Bildausschnitt (*view clip*) durch einen Rechteckpfad. Für die Parameter gelten die gleichen Bedingungen wie bei *rectclip*.

renamefile

Format: old new renamefile

Der Befehl benennt eine Datei mit dem Namen *old* in den neuen Namen *new* um. Die Parameter *old* und *new* sind Strings. Existiert die Datei nicht, wird ein *undefinedfilename*-Fehler ausgelöst. Ist eine Umbenennung nicht erlaubt, wird ein *invalidfileaccess*-Error ausgelöst.

resourceforall

Format: template proc scratch category resourceforall

Der Befehl zählt alle Namen der Instanzen einer angegebenen Ressourcenkategorie auf. *category* ist der Objektname einer Ressourcenkategorie (z. B. Fonts). In *template* wird ein String übergeben, der als Suchmuster für die Namen interpretiert wird. Dabei sind auch die Wildcards ** ?* erlaubt. Mit dem Zeichen ** wird das folgende Zeichen als Literal interpretiert (z. B. bedeutet ***, daß ein *** nicht als Wildcard zu interpretieren ist). Die gefundenen Namen werden im String *scratch* gespeichert. Für jeden gefundenen Namen wird die Prozedur *proc* aufgerufen.

resourcestatus

Format: key category resourcestatus status size bool

Der Befehl gibt den Status der in *key* angegebenen Ressourceninstanz zurück. In *category* läßt sich eine Kategorie der Ressource (z. B. Font) festlegen. Existiert die Ressource, gibt

der Befehl in *bool* den Wert *true* zurück. Dann enthält der Stack zwei zusätzliche Parameter. *status* kann folgende Werte aufweisen:

0 = in VM per *defineresource* definiert

1 = in VM durch den vorherigen Befehl *findresource* definiert

2 = nicht in VM definiert, aber über externen Speicher verfügbar

Der Integerwert *size* gibt an, wie viele Bytes die Ressource ungefähr in der VM verbraucht. Existiert die Ressource nicht, gibt der Befehl nur den Wert *false* auf dem Stack zurück.

rootfont

Format: `rootfont font`

Der Befehl gibt den zuletzt mit *setfont* oder *selectfont* ausgewählten Font zurück. Normalerweise ist der Wert mit der durch *currentfont* ermittelten Größe identisch. Beim Aufruf aus *cshow* gibt der Befehl den *composite font* zurück.

scheck

Format: `any scheck bool`

Der Befehl besitzt die gleiche Funktion wie *gcheck* und wird aus Kompatibilitätsgründen beibehalten.

selectfont

Format: `key scale selectfont`
`key matrix selectfont`

Der Befehl übernimmt einen Font, dessen Namen in *key* angegeben wird. Der Font wird bei der Übernahme gemäß *scale* oder *matrix* transformiert. In Level 1 konnte der gleiche Effekt mit der Sequenz:

```
/Times Roman findfont  
10 scalefont  
setfont
```

erreicht werden. In Level 2 ist nur noch die Anweisung:

```
/Times Roman 10 selectfont
```

erforderlich.

serialnumber

Format: `serialnumber int`

Der Befehl ermittelt die Seriennummer des Interpreters und gibt diese als Integerwert auf dem Stack zurück.

setbbox

Format: `llx lly urx ury setbbox`

Der Befehl legt die Bounding-Box des aktuellen Pfades fest, wobei die Bounding-Box durch zwei Koordinatenpaare definiert wird. *llx lly* definiert die linke untere Ecke (lower left corner), *urx* und *ury* definieren die obere rechte Ecke (upper right corner). Die Bounding-Box bleibt so lange erhalten, wie der Pfad gültig ist.

setblackgeneration

Format: `proc setblackgeneration`

Der Befehl definiert die Parameter *proc* der Prozedur *blackgeneration* im Grafik-State. Als Operand wird in *proc* eine Prozedur übergeben. Die Prozedur muß sich mit einer Zahl zwischen 0.0 und 1.0 aufrufen lassen. Aus diesem Wert (Graustufe) wird dann der neue Wert durch die Prozedur bestimmt und durch diese auf dem Stack zurückgegeben.

setcachedevice2

Format: `w0x w0y llx lly urx ury w1x w1y vx vy setcachedevice2`

Der Befehl übergibt zwei Sets der Zeichenmetrik an die Fontmaschine. *w0x* und *w0y* definieren den Abstand zum aktuellen Punkt zum neuen *current point* für die Textausgabe im Modus 0. *llx*, *lly* und *urx*, *ury* definieren die Bounding-Box bezogen auf den aktuellen Punkt für die Zeichen. *w1x* und *w1y* definieren den Abstand vom aktuellen Punkt zum neuen *current point*, wenn der Text im Mode 1 ausgegeben wird. *vx* und *vy* geben dann den Abstand vom Punkt 0 zum Punkt 1 an. Nach *setcachedevice2* bis zum Ende von *BuildGlyph* oder *BuildChar* sind Befehle zur Farbauswahl oder *imagemask* nicht zulässig.

setcacheparams

Format: `mark size lower upper setcacheparams`

Mit dem Befehl werden die Cacheparameter gesetzt. Die Zahl der Parameter wird in *size* angegeben.

setcmykcolor

Format: `cyan magenta yellow black setcmykcolor`

Dieser Befehl erlaubt es, die aktuelle Farbe im CMYK-Farbraum zu definieren, wobei die Farbdefinition über die Farbanteile *cyan*, *magenta*, *yellow* und *black* erfolgt. Die Werte dieser Parameter müssen zwischen 0.0 und 1.0 liegen.

setcolor

Format: `comp1 ... compn setcolor`

Mit dem Befehl werden die aktuellen Farbparameter im *graphic state* gesetzt. Die Zahl der Komponenten *comp* und deren Werte variieren in Abhängigkeit vom verwendeten Farbsystem.

setcolorrendering

Format: dict setcolorrendering

Der Befehl installiert *dict* als das aktuelle CIE-basierte *color rendering*-Dictionary im Graphic-State. Damit läßt sich die Farbausgabe an bestimmte Geräte anpassen.

setcolorscreen

Format: redfreq redang redproc
greenfreq greenang greenproc
bluefreq blueang blueproc
grayfreq grayang grayproc setcolorscreen

Mit dem Befehl lassen sich die Halbton-Parameter im Graphic-State setzen. Die Parameter geben die Farbanteile Rot, Grün, Blau und Grau mit den jeweiligen Komponenten an.

setcolorspace

Format: array setcolorspace
name setcolorspace

Der Befehl definiert den zu verwendenden Farbraum. Das *array* muß dabei im Format *[key param₁ ... param_n]* angegeben werden. In der zweiten Variante reicht es, den Namen des Farbsystems (DeviceGray, DeviceRGB, DeviceCMYK, Pattern) anzugeben.

setcolortransfer

Format: redproc greenproc blueproc grayproc setcolortransfer

Der Befehl definiert eine Transferfunktion für die Grafikmaschine. Jeder der Operanden für Rot, Grün, Blau und Grau muß eine PostScript-Prozedur sein, die Parameter im Bereich zwischen 0.0 und 1.0 erwartet. Dieser Parameter wird durch die Transferfunktion in einen Wert im Bereich 0.0 bis 1.0 umgerechnet.

setdevparams

Format: string dict setdevparams

Mit dem Befehl lassen sich ein oder mehrere Parameter für ein in *string* angegebenes Gerät übergeben. Die Schlüssel und Werte befinden sich in *dict*. Jeder Wert wird dabei durch einen Schlüssel *key* im Dictionary definiert.

setfileposition

Format: file position setfileposition

Der Befehl repositioniert den Zeiger einer geöffneten Datei *file* auf die in *position* angegebene Position. *position* muß eine positive Integerzahl sein. Bei einer Positionierung über das Dateiende hinaus hängt die Reaktion von der Implementierung des Dateisystems ab.

setglobal

Format: bool setglobal

Der Befehl setzt den VM Allocation Mode. Mit *true* wird der Modus auf *global* festgelegt, während *false* den VM-Modus auf *lokal* einstellt. Damit wirken sich *save*- und *restore*-Befehle auf die lokale oder globale VM aus.

setgstate

Format: gstate setgstate

Der Befehl ersetzt den aktuellen Grafikstatus (graphics state) durch den Inhalt des Objekts *gstate*.

sethalftone

Format: halftone sethalftone

Mit dem Befehl wird der Halftonparameter im aktuellen *graphic state* gesetzt.

setobjectformat

Format: int setobjectformat

Der Befehl legt das Objektformat für binäre Sequenzen von *printobject* und *writeobject* gemäß Tabelle 48.2 fest.

Code	Objekt
0	Disable binäre Kodierung
1	High-order byte first, IEEE Standard Real Format
2	Low-order byte first, IEEE Standard Real Format
3	High-order byte first, native Real Format
4	Low-order byte first, native Real Format

Tabelle 48.2 Kodierung der binären Objektformate

Der Wert 0 sperrt alle binären Kodierungen für die Ausgabewerte. Damit werden die Ausgaben als ASCII-Strings kodiert.

setoverprint

Format: bool setoverprint

Mit diesem Befehl wird der *setoverprint*-Parameter im *graphic state* gesetzt. Der Befehl wird verwendet, wenn die Einheit Farbseparationen unterstützt. Mit *false* werden die anderen Separationen gelöscht, und mit *true* bleiben diese erhalten.

setpacking

Format: `bool setpacking`

Der Befehl setzt den Packmodus für Arrays gemäß dem spezifizierten Wert. Mit *true* werden gepackte Arrays selektiert.

setpagedevice

Format: `dict setpagedevice`

Mit dem Befehl kann ein Systemadministrator ein neues Rasterausgabegerät im Interpreter installieren.

setpattern

Format: `pattern setpattern`
`comp1 compn pattern setpattern`

Der Befehl ermöglicht die Definition eines Musters in der aktuellen Farbe im *graphic state*. In *pattern* wird das Muster als *Dictionary* definiert (wird durch *setpattern* erzeugt). Definiert *pattern* ein farbloses Muster, enthalten die Parameter *comp_x* die betreffenden Farben, die dem Muster unterlegt werden.

setrgbcolor

Format: `red green blue setrgbcolor`

Eine Farbe im RGB-Farbraum läßt sich mit *setrgbcolor* definieren. Die Werte für *red*, *green* und *blue* müssen dabei zwischen 0.0 und 1.0 liegen.

setshared

Format: `bool setshared`

Der Befehl besitzt die gleiche Semantik wie die Anweisung *setglobal* und wird aus Kompatibilitätsgründen beibehalten.

setstrokeadjust

Format: `bool setstrokeadjust`

Mit dem Befehl wird der Parameter *stroke adjust* im aktuellen Grafikstatus (*graphics state*) beeinflußt. Mit dem Wert *true* wird die Justierung für *stroke* aktiviert.

setsystemparms

Format: `dict setsystemparms`

Der Befehl erlaubt es, einen oder mehrere Systemparameter zu setzen. Die Schlüssel *keys* und die Werte dieser Systemparameter sind in *dict* zu übergeben. Die Änderung der Systemparameter wird in der Regel durch ein Kennwort geschützt. Dieses Kennwort kann durch den Eintrag *Password* im *Dictionary* gesetzt werden.

setucacheparms

Format: mark blimit setucacheparms

Mit *setucacheparms* wird der Parameter des *user path cache* beeinflusst. Die Integerparameter liegen oberhalb des mit *mark* markierten Elements auf dem Stack. In *blimit* wird die Zahl der belegbaren Bytes für einen Pfad im *user path cache* festgelegt.

setundercolorremoval

Format: proc setundercolorremoval

Der Befehl übergibt eine Prozedur, mit der eine Farbsubtraktion für RGB, GRAY oder CMYK realisiert wird. Die der Prozedur übergebenen Werte liegen im Bereich zwischen 0.0 und 1.0. Der Interpreter erwartet die Ergebnisse im gleichen Wertebereich.

setuserparams

Format: dict setuserparams

Der Befehl besitzt die obige Syntax und ermöglicht es, User-Pparameter über den Dictionary-Eintrag *dict* zu setzen.

setvmthreshold

Format: int setvmthreshold

Mit dem Befehl wird die Schwelle für den Beginn der *Garbage Collection* gesetzt. Mit -1 wird die in der Implementierung definierte Schwelle verwendet.

shreddict

Format: shreddict dict

Der Befehl wirkt wie *globaldict* und wird aus Kompatibilitätsgründen zu Display-PostScript beibehalten.

sharedfontdirectory

Format: sharedfontdirectory dict

Der Befehl wirkt wie *globalfontdirectory* und wird aus Kompatibilitätsgründen zu Display-PostScript beibehalten.

startjob

Format: bool password startjob bool

Mit *startjob* wird ein neuer Job im PostScript-Interpreter gestartet.

uappend

Format: `userpath uappend`

Mit dem Befehl wird die *userpath*-Definition interpretiert und das Ergebnis an den aktuellen Pfad angehängt.

ucache

Format: `ucache`

Der Befehl veranlaßt, daß der aktuelle *user path* im Cache zu halten ist.

ucachestatus

Format: `ucachestatus mark bsize bmax rsize rmax blimit`

Der Befehl ermittelt den Status des User-Cache.

ueofill

Format: `userpath ueofill`

Der Befehl wirkt wie *ufill*, verwendet aber die *eofill*-Funktion anstelle von *fill*.

ufill

Format: `userpath fill`

Mit dem Befehl wird die Definition des *userpaths* interpretiert und der resultierende Pfad mit *fill* gefüllt. Die Operation wird dabei mit *gsave* und *grestore* eingeschlossen.

undef

Format: `dict key undef`

Der Befehl entfernt den Schlüssel *key* und den zugehörigen Wert aus dem angegebenen Dictionary *dict*.

undefinedresource

Dies ist ein Fehler, der auftreten kann, falls eine nicht definierte Ressource aufgerufen wird.

undefinefont

Format: `key undefinefont`

Der Befehl entfernt den Eintrag *key* und den zugehörigen Wert aus dem Font-Dictionary.

undefineresource

Format: `key category undefineresource`

Mit dem Befehl läßt sich die in *key* benannte Ressource der angegebenen Kategorie (z. B. Font) entfernen.

undefineuserobject

Format: `index undefineuserobject`

Der Befehl hebt die Verknüpfung zwischen dem User-Objekt und dem angegebenen Index auf.

upath

Format: `bool upath userpath`

Mit dem Befehl wird ein neues *user path*-Objekt erzeugt, welches dem aktuellen Pfad entspricht. Der Parameter *bool* definiert, ob *ucache* als erstes Element in den Pfad aufzunehmen ist.

UserObjects

Format: `UserObjects array`

Der Befehl gibt ein Array der User Objects des aktuellen *userdicts* zurück.

ustroke

Format: `userpath ustroke`
`userpath matrix ustroke`

Mit dem Befehl wird der Userpfad interpretiert und ein *stroke*-Befehl ausgeführt. *ustroke* wird mit *gsave* und *grestore* eingeschlossen.

ustrokepath

Format: `userpath ustrokepath`
`userpath matrix ustrokepath`

Der Befehl ersetzt den aktuellen Pfad mit einem Pfad, der bei Anwendung von *ustroke* entsteht.

vmreclaim

Format: `int vmreclaim`

Der Befehl kontrolliert die *Garbage Collection* des Speichers über den Parameter *int*. Mit *-2* wird die automatische Speicherbereinigung in der lokalen und globalen VM abgeschaltet. Mit *-1* bezieht sich die Sperre nur auf die lokale VM. Mit *0* wird die automatische Speicherbereinigung zugelassen. *1* bewirkt eine sofortige Speicherbereinigung in der lokalen VM, während *2* sich auf beide VMs bezieht.

writeobject

Format: `file obj tag writeobject`

Der Befehl schreibt eine Sequenz binärer Objekte in die angegebene Datei. Der Dateiname wird in *file* angegeben. Der Befehl wirkt wie *printobject*.

xyshow

Format: `string numarray xyshow`
`string numstring xyshow`

Mit diesem Befehl wird ein *string* ähnlich wie bei *show* ausgegeben. Nach der Ausgabe eines Zeichens werden aus *numarray* oder *numstring* zwei aufeinanderfolgende Zahlen gelesen, die die Koordinaten der nächsten Ausgabeposition im User-Koordinatensystem angeben. Die erste Zahl gibt die relative x-Position zum zuletzt ausgegebenen Zeichen an. Der andere Parameter bezieht sich auf die y-Position.

yshow

Format: `string numarray yshow`
`string numstring yshow`

Der Befehl ist vergleichbar *xyshow*, wobei jedoch nur ein Wert für y aus *numarray* oder *numstring* bezogen wird.

Die obige Aufstellung enthält nur die wichtigsten Informationen zu den PostScript Level-2-Befehlen. Die Beschreibung aller Feinheiten würde ein eigenes Buch umfassen. Hier möchte ich auf das bei Addison-Wesley publizierte Werk »PostScript Language Reference Manual« verweisen.

Index

A

Adobe Illustrator *s.* AI-Format
AI-Format 515
 Clipping-Operatoren 525
 Color Operators 523
 Compound Paths-Kommandos 525
 Graphic State Operators 523
 Graphs-Operatoren 529
 Group Operators 524
 Locked Object Operator 523
 Path Definition Commands 524
 Path Painting-Kommandos 525
 Script-Body 521
 Text 526
Animator CEL-Format 589
Animator PIC-Format 589
Apple QuickTime-Format *s.* QTM
AutoCAD *s.* DXF
AutoCAD Binary DXF 580
Autocad Drawing Exchange Format *s.* DXF
Autodesk 3D Studio-Format *s.* FLC
Autodesk Animator-Format *s.* FLI
AVI
 AVI_PALCHANGE-CHUNK 608
 CHUNK 602
 Format 599
 Header 601
 idx1-CHUNK 609
 JUNK-CHUNK 609
 movi-CHUNK 607
 rec-CHUNK 607
 Stream Data-CHUNK (strd) 607
 Stream Format-CHUNK (strf) 605
 strl-CHUNK 603

B

BIFF-Recordtypen *s.* EXCEL
Bildschirm-Split 20
Binary Interchange Format (BIFF) *s.* EXCEL
BMP (Windows)
 BMP4-Header 615
 Datenbereich 616
 erweiterter Header 611
 RLE-Komprimierung 617

C

CAPTURE File Format (SCR) 835
CAS Fax Format (DCX) 631

CGA 829

 Color Map speichern 829
 Farbinformation 829
 Kodierungen 829

CGM

 Alternate Character Set Index 652
 Application Data 654
 ASCII-Steuercodes 640
 Aspect Source Flags 654
 Auxiliary Color 648
 Background Color 648
 Basiskodierung 645
 binäre Kodierung 633
 BitStream-Format 645
 Cell Array 650
 Character Coding Announcer 647
 Character Expansion Factor 651
 Character Height 652
 Character Orientation 652
 Character Set Index 652
 Character Set List 647
 Character Spacing 652
 Circle 650
 Circular ARC 3 Point 650
 Circular ARC 3 Point Close 650
 Circular ARC Centre 650
 Circular ARC Centre Close 650
 Clip Indicator 649
 Clip Rectangle 649
 Color Index Precision 646
 Color Precision 646
 Color Selection Mode 648
 Color Table 653
 Color Value Extent 647
 Defaults Replacement 647
 Description 646
 Disjoint Polyline 649
 Edge Bundle Index 653
 Edge Color 653
 Edge Type 653
 Edge Visibility 653
 Edge Width 653
 Edge Width Specification Mode 648
 Element List 647
 Ellipse 650
 Elliptical Arc 650
 Elliptical Arc Close 650
 ESCAPE 654
 Fill Bundle Index 652

CGM (Forts.)

- Fill Color 652
- Fill Reference Point 653
- Font List 647
- Generalized Drawing Primitive 650
- Hatch Index 653
- Header (erweitert) 636
- Header (Kurzform) 636
- Index Precision 646
- Integer Precision 646
- Interior Style 652
- ISO-kodierte Steuercode 642
- Kodierungsformat 633
- Line Bundle Index 651
- Line Color 651
- Line Type 651
- Line Width 651
- Line Width Specification Mode 648
- Marker Bundle Index 651
- Marker Color 651
- Marker Size 651
- Marker Size Specification Mode 648
- Marker Type 651
- Maximum Color Index 647
- Message 654
- Metafile-Anweisungen 645
- Pattern Index 653
- Pattern Size 653
- Pattern Table 653
- Polygon 649
- Polygon Set 649
- Polyline 649
- Polymarker 649
- Precision Parameter Encoding 648
- Real Precision 646
- Rectangle 650
- Restricted Text 649
- Scaling Mode 648
- Steuercodes (binär) 637
- Text 649
- Text Alignment 652
- Text Bundle Index 651
- Text Color 652
- Text Font Index 651
- Text Path 652
- Text Precision 651
- Transparency 649
- VDC Extent 648
- VDC Space/Range 648
- VDC Type 646
- verkettete Parameterliste 637
- Version 646

CMF

- Header 965
- Instrument-Block 967
- Komprimierung 974
- Music-Block 969
- Pause command 969
- Steuerbefehle 971
- Ton ein/aus 970, 971

Computer Graphic Metafile Format s. CGM

Creative-Music-File-Header s. CMF

Creative Voice-Format s. VOC

CUT-Format 658

D

Data Interchange Format s. DIF

DCX

- Header 631

DeLuxe Paint 740

DIF (Data Interchange Format) 311

- COMMENT 314
- DATA 313
- Datei 311
- Datensatzstruktur 317
- Datum 318
- DISPLAYUNITS 316
- Ende 313
- Header
 - Analyse von Zeitreihen 315
 - MAJORSTART 315
 - Satztypen 312
 - UNITS 316
- Headeraufbau 311
- Kommentare 314
- LABEL 314
- MINORSTART 315
- numerische Daten 317
- Periodicity 315
- signifikante Daten 316
- SIZE 315
- Spaltenbreite in Byte 315
- Spaltenname 316
- Spaltenüberschrift 314
- Special Datatype 317
- Stringdaten 318
- TABLE 312
- TRUELENGTH 316
- TUPLES 313
- Value Indicator 318
- VECTORS 312
- Vektorenanzahl 312
- Zeilenanzahl 313
- Zeitangabe 315

DOC
 Dateiaufbau 331
 File Information Block 331
 Main-Stream 331
 Textbereichs 340
 Zeichen- und Absatzformatierung 343
 Dr. Halo-Format 655
 Drawing Exchange Binary-Format (DXB) 581
 DRW s. Micrografx-Formate
 DXF (AutoCAD Drawing Exchange Format) 537
 3DFACE 577
 3DLINE 567
 APPID 556
 ARC 569
 ATTRIB 573
 AutoCAD Binary DXF 580
 Bildbeschreibung 566
 Block-Flag 564
 Blocks (Table) 564
 CIRCLE 568
 Dateiabscnitte 537
 Dateiaufbau 537
 DIMENSION 579
 DIMSTYLE 556
 Drawing Exchange Binary-Format (DXB) 581
 Entities Section 566
 3DFACE 577
 3DLINE 567
 ARC 569
 ATTRIB 573
 CIRCLE 568
 DIMENSION 579
 INSERT 572
 LINE 567
 Objekte 566
 POINT 568
 POLYLINE 574
 SEQEND 577
 SHAPE 572
 SOLID 570
 TEXT 570
 TRACE 569
 VERTEX 576
 VIEWPORT 577
 Gruppencode 537
 Headeraufbau 548
 INSERT 572
 LAYER 555, 558
 LINE 567
 LTYPE 555, 559
 POINT 568
 POLYLINE 574
 Schlüsselworte (Table Section) 555
 DXF (Forts.)
 SEQEND 577
 SHAPE 572
 SOLID 570
 STYLE 560
 Table-Section 554
 TEXT 570
 TRACE 569
 UCS 562
 VERTEX 576
 VIEW 562
 VIEWPORT 577
 VPORT 563

E
 EMF
 Header 947
 Records 947
 Encapsulated PostScript Format s. EPS
 Enhanced Metafile-Format s. EMF
 EPS
 abs 1036
 add 1037
 and 1037
 Anweisungen 1036
 arc 1037
 arcn 1037
 arcto 1037
 ashow 1037
 atan 1037
 awidthshow 1037
 bind 1038
 bitshift 1038
 Body Comments 1035
 BoundingBox 1035
 ceiling 1038
 charpath 1038
 clip 1038
 clippath 1038
 closepath 1038
 coppypage 1038
 cos 1039
 Creator 1035
 currentpoint 1039
 curveto 1039
 cvi 1039
 cvn 1039
 cvrs 1039
 cvs 1039
 cvt 1039
 div 1040
 DocumentFonts 1035
 dup 1040

EPS (Forts.)
EndComments 1035
eoclip 1040
eofill 1040
eq 1040
exch 1040
exec 1040
exit 1040
exp 1040
fill 1041
findfont 1041
For 1035
for 1041
ge 1041
grestore 1041
gsave 1041
gt 1041
Header Comments 1035
idiv 1041
if 1042
ifelse 1042
image 1042
imagemask 1042
index 1042
Kommentare 1035
le 1042
lineto 1042
ln 1043
log 1043
loop 1043
lt 1043
mod 1043
moveto 1043
mul 1043
neg 1043
newpath 1043
not 1044
or 1044
Page 1036
pop 1044
quit 1044
rand 1044
rcurveto 1044
repeat 1044
rlinto 1044
roll 1044
round 1045
scale 1045
scalefont 1045
search 1045
setfont 1045
setgray 1045

EPS (Forts.)
setlinewidth 1045
show 1045
showpage 1046
sin 1046
sqrt 1046
string 1046
stringwidth 1046
stroke 1046
Strukturkonventionen 1034
sub 1046
title 1035
Trailer Comments 1036
truncate 1046
Versionsnummer 1034
xor 1046
EXCEL 87
1904 (Recordtyp 22H) 218
ADDIN (Recordtyp 87H) 98
ADDMENU (Recordtyp C2H) 98
ARRAY (Recordtyp 21H) 99
AUTOFILTER (Recordtyp 9EH) 101
AUTOFILTERINFO (Recordtyp 9DH) 102
BACKUP (Recordtyp 40H) 102
Bildobjekt (Picture) 174
BLANK (Recordtyp 201H) 103
BOF (Recordtyp 09H) 104
BOOKBOOL (Recordtyp DAH) 105
BOOLERR (Recordtyp 05H) 106
BOTTOMMARGIN (Recordtyp 29) 107
BOUNDSHEET (Recordtyp 85) 107
BUILTINFMTCOUNT (Recordtyp 56) 108
CALCCOUNT (Recordtyp 0CH) 108
CALCMODE (Recordtyp 0DH) 108
CF (Recordtyp 1B1H) 109
Chart-Objekt 169
CODENAME (Recordtyp 42H) 109
CODEPAGE (Recordtyp 42H) 110
COLINFO (Recordtyp 7DH) 110
COLUMNDEFAULT (Recordtyp 20H) 112
COLWIDTH (Recordtyp 24H) 112
CONDFMT (Recordtyp 1B0H) 111
CONTINUE (Recordtyp 3CH) 112
COORDLIST (Recordtyp A9H) 113
COUNTRY (Recordtyp 8CH) 113
CRN (Recordtyp 5AH) 113
DBCELL (Recordtyp D7H) 115
DCON (Recordtyp 50H) 115
DCONNAME (Recordtyp 52H) 116
DCONREF (Recordtyp 51H) 116
DEFAULTROWHEIGHT (Recordtyp 25H)
117

EXCEL (Forts.)

DEFCOLWIDTH (Recordtyp 55H) 118
 DELMENU (Recordtyp C3H) 118
 DELTA (Recordtyp 10H) 118
 DIMENSIONS (Recordtyp 200H) 119
 DOCROUTE (Recordtyp B8H) 119
 DSF (Recordtyp 161H) 120
 DV (Recordtyp 1BEH) 120
 DVAL (Recordtyp 1B2H) 121
 EDG (Recordtyp 88H) 121
 EFONT (Recordtyp 45H) 122
 Ellipsenobjekt 167
 EOF (Recordtyp 0AH) 122
 EXTERNNAME (Recordtyp 23H) 123
 EXTERNSHEET (Recordtyp 17H) 125
 EXTRNCOUNT (Recordtyp 16H) 122
 EXTSSST (Recordtyp FFH) 127
 FILEPASS (Recordtyp 2FH) 127
 FILESHARING (Recordtyp 5BH) 127
 FILESHARING2 (Recordtyp 1A5H) 128
 FITERMODE (Recordtyp 5BH) 128
 FNGROUPCOUNT (Recordtyp 9CH) 129
 FNGROUPNAME (Recordtyp 9AH) 129
 FNPROTO (Recordtyp A2H) 129
 FONT (Recordtyp 31H) 130
 FONT2 (Recordtyp 86H) 132
 FOOTER (Recordtyp 15H) 132
 FORMAT (Recordtyp 1EH) 132
 FORMATCOUNT (Recordtyp 1EH) 133
 FORMULA (Recordtyp 06H) 133
 GCW (Recordtyp 5CH) 145
 GRIDSET (Recordtyp 82H) 145
 Gruppe 175
 GUTS(Recordtyp 80H) 146
 HCENTER(Recordtyp 83H) 146
 HEADER (Recordtyp 14H) 146
 HIDEOBJ (Recordtyp 8DH) 147
 HLINK (Recordtyp 1B8H) 147
 HORIZONTAL PAGE BREAKS (Recordtyp 1BH) 147
 IMDATA (Recordtyp 7FH) 148
 INDEX (Recordtyp 0BH) 148
 INTEGER (Recordtyp 02H) 149
 INTERFACEEND (Recordtyp E2H) 150
 INTERFACEHDR (Recordtyp E1H) 150
 INTL (Recordtyp 61H) 150
 ITERATION (Recordtyp 11H) 151
 IXFE (Recordtyp 44H) 151
 Kreisbogenobjekt (Arc) 168
 LABEL (Recordtyp 04H) 151
 LABELSST (Recordtyp FDH) 152

EXCEL (Forts.)

LEFTMARGIN (Recordtyp 26H) 152
 LH (Recordtyp 8BH) 153
 LHNGRAPH (Recordtyp 95H) 153
 LHRECORD (Recordtyp 95H) 153
 Linienobjekt 164
 LPR (Recordtyp 98H) 154
 MMS (Recordtyp C1H) 155
 MSODRAWING (Recordtyp ECH) 155
 MSODRAWINGGROUP (Recordtyp EBH) 155
 MSODRAWINGSELECTION (Recordtyp EDH) 155
 MULBLANK (Recordtyp BEH) 156
 MULRK (Recordtyp BDH) 156
 NAME 156
 NOTE (Recordtyp 1CH) 160
 NUMBER (Recordtyp 203H) 161
 OBJ (Recordtyp 5DH) 162
 OBJPRO (Recordtyp D3H) 179
 OBJPROTECT (Recordtyp 63H) 179
 OLESIZE (Recordtyp DEH) 179
 PALETTE (Recordtyp 92H) 179
 PANE (Recordtyp 41H) 180
 PARAMQRY (Recordtyp DCH) 180
 PASSWORD (Recordtyp 13H) 181
 PLS (Recordtyp 4DH) 181
 Polygon 175
 Precision (Recordtyp 0EH) 183
 PRINTGRIDLINES (Recordtyp 2BH) 183
 PRINTHEADERS (Recordtyp 2AH) 183
 PROT4REV (Recordtyp 1AFH) 184
 PROT4REVPASS (Recordtyp 1BCH) 184
 PROTECT (Recordtyp 12H) 183
 PUB (Recordtyp 89H) 184
 QSI (Recordtyp 1ADH) 185
 RECIPNAME (Recordtyp B9H) 185
 Recordaufbau 87
 REFMODE (Recordtyp 0FH) 186
 REFRESHALL (Recordtyp 1B7H) 186
 RIGHTMARGIN (Recordtyp 27H) 186
 RK (Recordtyp 27EH) 186
 ROW (Recordtyp 08H) 188
 RSTRING (Recordtyp D6H) 189
 SAVERECAL (Recordtyp 5FH) 189
 SCENARIO (Recordtyp AFH) 190
 SCENMAN (Recordtyp AEH) 190
 SCENPROTECT (Recordtyp DDH) 191
 Schaltfläche (Button) 172
 SCL (Recordtyp A0H) 191
 SELECTION (Recordtyp 1DH) 191

EXCEL (Forts.)
 SETUP (Recordtyp A1H) 192
 SHRFMLA (Recordtyp BCH) 193
 SORT (Recordtyp 90H) 193
 SOUND (Recordtyp 96H) 194
 SST (Recordtyp FCH) 194
 STANDARDWIDTH (Recordtyp 99H) 194
 STRING (Recordtyp 07H) 195
 STYLE (Recordtyp 293H) 195
 SUB (Recordtyp 91H) 196
 SUBBOOK (Recordtyp 1AEH) 197
 SX-Records 197
 SYNC (Recordtyp 97H) 198
 TABID (Recordtyp 13DH) 198
 TABIDCONF (Recordtyp EAH) 198
 TABLE (Recordtyp 36H) 199
 TABLE2 (Recordtyp 37H) 200
 TEMPLATE (Recordtyp 60H) 200
 Text to Objekt 169
 TOPMARGIN (Recordtyp 28H) 201
 TXO (Recordtyp 1B6H) 201
 UDESC (Recordtyp DFH) 202
 UNCALCED (Recordtyp 5EH) 202
 Unicode-Zeichenketten 89
 USERVIEW (Recordtyp 1A9H) 202
 USERVIEWBEGIN (Recordtyp 1AAH) 203
 USERVIEWEND (Recordtyp 1ABH) 204
 USESELFS (Recordtyp 160H) 204
 VCENTER (Recordtyp 84H) 204
 VERTICALPAGEBREAKS (Recordtyp 1AH) 205
 WINDOW 1 (Recordtyp 3DH) 205
 WINDOW 2 (Recordtyp 3EH) 206
 WINDOWPROTECT (Recordtyp 19H) 207
 WRITEACCESS (Recordtyp 5CH) 208
 WRITEPROT (Recordtyp 86H) 208
 WSBOL (Recordtyp 81H) 208
 XCT (Recordtyp 59H) 209
 XF (Recordtyp 43H) 209
 Extended Markup Language s. XML

F

Farbtabelle 448
 Fax CCITT Group 3-Verfahren 886
 Fax CCITT Group 4-Verfahren 886
 Fax-Komprimierung 918
 FKP 334
 FLC

COLOR_256-CHUNK (Typ 4) 594
 DELTA_FLC-CHUNK (Typ 7) 594
 FLC_BLACK-CHUNK (Typ 13) 596
 FLC_BYTE_RUN-CHUNK (Typ 15) 596
 Frames 593

FLC (Forts.)

Header 591
 LITERAL-CHUNK (Typ 16) 597
 PREFIX-CHUNK (Typ F100H) 593
 PSTAMP-CHUNK (Typ 18) 597

FLI

COLOR_64-CHUNK (Type 11) 585
 DELTA_FLI-CHUNK (Type 12) 586
 FLI_BLACK-CHUNK (Type 13) 587
 FLI_BYTE_RUN-CHUNK (Type 15) 587
 FLI_COPY-CHUNK (Type 16) 588
 Frames 584
 Header 583
 Fontfamilie 443

G

GDP (Generalized Drawing Primitives)

Arc 699
 Bar 699
 Circle 701
 Ellipse 701
 Elliptical Arc 702
 Elliptical Pie 702
 Filled Rounded Rectangle 703
 Justified Graphics Text 704
 Pie 700
 Rounded Rectangle 703
 s. auch GEM

GEM

Arc GDP 699
 Bar GDP 699
 Circle GDP 701
 Ellipse GDP 701
 Elliptical Arc GDP 702
 Elliptical Pie GDP 702
 Fill Area 698
 Filled Rounded Rectangle GDP 703
 Generalized Drawing Primitives (GDP) 699
 Justified Graphics Text GDP 704
 Pie GDP 700
 Poly Line 697
 Poly Marker 697
 Rounded Rectangle GDP 703
 Set Character Baseline Vector 705
 Set Character Height 705, 713
 Set Color Mode 705
 Set Fill Color Index 710
 Set Fill Interior Style 709
 Set Fill Parameter Visibility 712
 Set Fill Style Index 710
 Set Graphic Text Alignment 711
 Set Graphic Text Special Effects 713
 Set Polyline Color Index 707

GEM (Forts.)

- Set Polyline End Style 714
- Set Polyline Type 706
- Set Polyline Width 707
- Set Polymarker Color Index 708
- Set Polymarker Height 708
- Set Polymarker Type 707
- Set Text Color Index 709
- Set Text Font 709
- Set User Defined Fill Pattern 714
- Set User Defined Linestyle Pattern 714
- Set Writing Mode 710
- Text 698

GEM Image File-Format s. IMG

GEM Metafile-Format 693

- Bildgröße 694
- Ende 716
- Headeraufbau 693
- Kodierung der Schrifteffekte 713
- Koordinatensysteme 694
- Objekte 695
- Programmer's Toolkit 716
- Steuercodes 695
- Versionsnummer 683

Generalized Drawing Primitives s. GDP

GIF (Graphics Interchange Format)

- Application Extension-Block 678
- Comment Extension-Block 676
- Extension-Block 667
- Global Color Map-Block 664
- Graphic Control Extension-Block 674
- Image Descriptor-Block 665
- Local Color Map-Block 667
- Logical Screen Descriptor-Block 662
- LZW-Komprimierung 669
- LZW-Verfahren 668
- Pixel Aspect Ratio 664
- Plain Text Extension-Block 677
- Raster Data-Block 668
- Raster Data-Subblocks 674
- Resolution-Flag in GIF89a 663
- Terminator-Block 674, 679

Graphicraft 740

Graphics Interchange Format s. GIF

GRF s. Micrografx-Formate

H

HP-GL/2 (Hewlett Packard Graphic Language)

- 1009
- Absolute Arc Tree Point 1015
- Absolute Character Size 1024
- Absolute Direction 1022
- Advance Full Page 1013

HP-GL/2 (Forts.)

- Alternate Font Definition 1021
- Anchor Corner 1019
- Arc Absolute 1014
- Arc Relative 1015
- Befehle 1009
- Befehlsgruppen (Liste) 1010
- Begin Plot 1025
- Character Fill Mode 1021
- Character Group 1021
- Character Plot 1022
- Character Slant 1024
- Chord Tolerance Mode 1025
- Circle 1015
- Configuration and Status Group 1012
- Define Label Terminator 1022
- Define Variable Text Path 1022
- Digitize Clear 1028
- Digitize Point 1029
- Digitizing Extensions 1028
- Dual-Context Extension 1028
- Edge Polygon 1017
- Edge Rectangle Absolute 1017
- Edge Rectangle Relative 1017
- Edge Wedge 1017
- Enable Cutter 1025
- Enter PCL Mode 1028
- Extra Space 1023
- Fill Polygon 1018
- Fill Rectangle Absolute 1018
- Fill Rectangle Relative 1018
- Fill Type 1019
- Fill Wedge 1018
- Frame Advance 1025
- Initialize 1012
- Input P1/P2 1012
- Input Relative 1013
- Input Window 1013
- Label 1023
- Label Origin 1023
- Line and Fill Attributes Group 1019
- Line Attributes 1019
- Line Type 1020
- Media Type 1026
- Merge Control 1025
- Message 1026
- Not Ready 1026
- Number of Pens 1027
- Output Digitized Position/Pen Status 1029
- Output Error 1026
- Output Hard-Clip Limits 1026
- Output Identification 1026
- Output P1/P2 1026

HP-GL/2 (Forts.)

- Output Status 1026
- Palette Extension 1027
- Pen Color Assignment 1027
- Pen Down 1015
- Pen Up 1016
- Pen Width 1020
- Pen Width Unit Selection 1021
- Plot Absolute 1015
- Plot Relative 1016
- Plot Size 1026
- Polygon Group 1017
- Polygon Mode 1018
- Polyline Encoded 1016
- Primary Font Selection 1028
- Quality Level 1027
- Raster Fill 1020
- Relative Arc Tree Point 1017
- Relative Character Size 1024
- Relative Direction 1022
- Replot 1014
- Reset 1028
- Rotate Coordinate System 1013
- Scalable/Bitmap Fonts 1028
- Scale 1014
- Screened Vectors 1027
- Secondary Font Selection 1028
- Select Alternate Font 1023
- Select Pen 1020
- Select Standard Font 1024
- Set Color Range 1027
- Set Default Values 1012
- Standard Font Definition 1023
- Symbol Mode 1020
- Technical Graphics Extension 1024
- Transparency Mode 1027
- Transparent Data 1024
- User-Defined Line Type 1020
- Vector Group 1014
- Velocity Select 1027

HTML

- Aufzählung 1097
- Dateiformat 1089
- Grafik 1098
- Grundstruktur 1090
- Hintergrundfarbe 1099
- horizontale Linien 1096
- Numerierung 1097
- Schriftfarbe 1095
- Schriftgröße 1095
- Sonderzeichen 1091
- Tabellen 1101
- Tag 1090
- Überschriften 1093

I

IFD

- Tags 879

IFF (Interchange File Format)

- 8SVX-CHUNK 744
- ATAK-CHUNK 746
- Bitmap Header-CHUNK 738
- Blockstruktur (CHUNK) 736
- BODY-CHUNK 745
 - Musical Instrument 746
 - One Shot Sound 745
- CAMG-CHUNK 743
- CHRS-CHUNK 750
- CHUNK 733
- CHUNK-Signaturen 737
- CLUT-CHUNK 743
- COLR-CHUNK 751
- COMM-CHUNK 747
- Dateiaufbau 733
- Dateiheader (FORM) 733
- DEST-CHUNK 742
- DOC-CHUNK 751
- FONS-CHUNK 749
- FONT-CHUNK 750
- FOOT/HEAD-CHUNK 751
- FORM 733
- FSCC-CHUNK 753
- FTXT FORM 749
- GRAB-CHUNK 742
- Header
 - 8SVX-Form 744
- Headeraufbau 734
- HeaderSignatur 736
- ILBM-Form-CHUNKs 738
- INS1-CHUNK 749
- INST-CHUNK 748
- Musical Instrument 746
- NAME-CHUNK 745
- One Shot Sound 745
- PAGE-CHUNK 753
- PARA-CHUNK 752
- PCTS-CHUNK 754
- PINF-CHUNK 754
- RLSE-CHUNK 746
- SHDR-CHUNK 748
- SMUS FORM 748
- SNDD-CHUNK 747
- SPRT-CHUNK 742
- TABS-CHUNK 752
- TEXT-CHUNK 753
- TRAK-CHUNK 749
- Voice Header-CHUNK (VHDR) 744
- WORD FORM 750

- IGES 719
 - Directory-Entry-Section 722
 - Global-Section 720
 - Parameter-Data-Section 724
 - Start-Section 720
 - Terminate-Section 724
- IMG (GEM Image File Format) 681
 - Bildkomprimierung 686
 - Bit String-Record 687
 - Bits pro Pixel 683
 - Dateiaufbau 682
 - Datenspeicherung 685, 686
 - GEM-Versionsnummer 683
 - Headeraufbau 682
 - Kodierungen 686
 - Komprimierungsverfahren 686
 - Kopfsatz 682
 - Pattern Run-Record 688
 - Pixel 681
 - Pixelkodierung 686
 - Solid Run-Kodierung 687
 - Vertical Replication Count-Verfahren 689
- Informationsgruppe 454
- Initial Graphics Exchange Language s. IGES
- Interchange File Format s. IFF
- ISO
 - Metafile-Kodierung (CGM) 642
 - ISO 646-7-Bit-Zeichensatz 1071
 - ISO 646-Kodierung 642
- J**
 - JFIF
 - Application (APP0-)Marker 759
 - Define Arithmetic Coding-(DAC-)Marker 762
 - Define Huffman Table-(DHT)-Marker 762
 - Define Quantisation Table-(DQT)-Marker 762
 - Define Restart Interval-(DRI)-Marker 763
 - End of Image-(EOI)-Marker 758
 - Extension APP0-(SOI)-Marker 760
 - Start of Frame-(SOF)-Marker 763
 - Start of Image-(SOI)-Marker 758
 - Start of Scan-(SOS)-Marker 766
 - JPEG-Komprimierung 923
- K**
 - Komprimierungsverfahren 686
 - CCITT/3 1-D 886
 - Fax 918
 - Fax CCITT Group 3 886
 - Fax CCITT Group 4 886
 - JPEG 923
 - Komprimierungsverfahren (Forts.)
 - LZW 923
 - PackBit 887, 917
 - private 887
 - Run Length Encoding (RLE) 832
 - TIFF-Format 917
- L**
 - LMBCS 55
 - LOTUS 1-2-3 s. WKS/WK1, WK3
 - LOTUS Multibyte Character Set 55
 - LZW-Komprimierung 923
 - LZW-Verfahren 668
- M**
 - MAC-Datenbereich 774
 - MAC-Header 772
 - MAC-Packbit 775
 - MAC-Print-Format (MAC) 771
 - MAC-Picture-Format s. PICT
 - Metafile-Anweisungen 645
 - Alternate Character Set Index 652
 - Application Data 654
 - Aspect Source Flags 654
 - Auxiliary Color 648
 - Background Color 648
 - Cell Array 650
 - Character Coding Announcer 647
 - Character Expansion Factor 651
 - Character Height 652
 - Character Orientation 652
 - Character Set Index 652
 - Character Set List 647
 - Character Spacing 652
 - Circle 650
 - Circular ARC 3 Point 650
 - Circular ARC 3 Point Close 650
 - Circular ARC Centre 650
 - Circular ARC Centre Close 650
 - Clip Indicator 649
 - Clip Rectangle 649
 - Color Index Precision 646
 - Color Precision 646
 - Color Selection Mode 648
 - Color Table 653
 - Color Value Extent 647
 - Defaults Replacement 647
 - Description 646
 - Disjoint Polyline 649
 - Edge Bundle Index 653
 - Edge Color 653
 - Edge Type 653
 - Edge Visibility 653

Metafile-Anweisungen (Forts.)

- Edge Width 653
- Edge Width Specification Mode 648
- Element List 647
- Ellipse 650
- Elliptical Arc 650
- Elliptical Arc Close 650
- ESCAPE 654
- Fill Bundle Index 652
- Fill Color 652
- Fill Reference Point 653
- Font List 647
- Generalized Drawing Primitive 650
- Hatch Index 653
- Index Precision 646
- Integer Precision 646
- Interior Style 652
- Line Bundle Index 651
- Line Color 651
- Line Type 651
- Line Width 651
- Line Width Specification Mode 648
- Marker Bundle Index 651
- Marker Color 651
- Marker Size 651
- Marker Size Specification Mode 648
- Marker Type 651
- Maximum Color Index 647
- Message 654
- Pattern Index 653
- Pattern Size 653
- Pattern Table 653
- Polygon 649
- Polygon Set 649
- Polyline 649
- Polymarker 649
- Precision Parameter Encoding 648
- Real Precision 646
- Rectangle 650
- Restricted Text 649
- Scaling Mode 648
- Text 649
- Text Alignment 652
- Text Bundle Index 651
- Text Color 652
- Text Font Index 651
- Text Path 652
- Text Precision 651
- Transparency 649
- VDC Extent 648
- VDC Space/Range 648
- VDC Type 646
- Version 646

- Micrografx-Formate (PIC, DRW, GRF) 789
 - CHART_SKIP_SYMBOLS (Typ 44, 2CH) 792
 - DRW_BACKGROUND (Typ 1, 01H) 793
 - DRW_BAND (Typ 32, 20H) 793
 - DRW_BITMAP (Typ 20, 14H) 794
 - DRW_COLOR (Typ 9, 09H) 794
 - DRW_COLOR_FLAG (Typ 10, 0AH) 794
 - DRW_COLOR_TABLE (Typ 35, 23H) 794
 - DRW_COMMENT (Typ 18, 12H) 795
 - DRW_CURR_OVERLAY (Typ 16, 10H) 795
 - DRW_DIMENSIONS (Typ 24, 18H) 795
 - DRW_EOF (Typ 254, FEH) 795
 - DRW_FACENAME (Typ 2, 02H) 796
 - DRW_FONT (Typ 21, 15H) 796
 - DRW_GRADIENT (Typ 30, 1EH) 797
 - DRW_GRID (Typ 22, 16H) 798
 - DRW_ID (Typ 4, 04H) 798
 - DRW_INFO (Typ 19, 13H) 798
 - DRW_LOCKED (Typ 29, 1DH) 798
 - DRW_MAX_LINK_ID (Typ 37, 25H) 799
 - DRW_OLD_GRID (Typ 15, 0FH) 799
 - DRW_OVERLAY (Typ 5, 05H) 800
 - DRW_OVERLAY_NAME (Typ 23, 17H) 800
 - DRW_PAGE (Typ 27, 1BH) 800
 - DRW_PATTERN (Typ 28, 1CH) 801
 - DRW_POLYGON (Typ 6, 06H) 801
 - DRW_RESOLUTION (Typ 25, 19H) 801
 - DRW_SYMBOL (Typ 7, 07H) 802
 - DRW_SYMBOLVERSION (Typ 33, 21H) 814
 - DRW_TEXT (Typ 8, 08H) 814
 - DRW_TEXTEXTRA (Typ 36, 24H) 814
 - DRW_TEXTHDR (Typ 31, 1FH) 814
 - DRW_TEXTPARA (Typ 34, 22H) 816
 - DRW_VERSION (Typ 3, 03H) 816
 - DRW_VIEW (Typ 14, 0EH) 817
 - DRW_VISIBLE (Typ 17, 11H) 817
 - Recordaufbau Version 1 803
 - Recordaufbau Version 2 805
 - Recordaufbau Version 3 807
 - Recordaufbau Version 4 809
 - Recordaufbau Version 5 811
 - SYMBOL-Record 803
 - VERSION_REC (Typ 255, FFH) 817

MIDI

- Betriebsarten 995
- Channel Prefix (32) 1002
- Copyright Note (02) 1001
- Cue Point (07) 1002
- Delta Time 987
- End Of System Exclusive (EOX) 998
- End of Track (47) 1002
- Format 985
- Header-CHUNK 985

MIDI (Forts.)

- Instrument Name (04) 1001
- Key Signature (89) 1003
- Local Control 995
- Long-Format 999
- Lyric (05) 1002
- Marker (06) 1002
- Meta-Events 1000
- Note off 992
- Note on 989
- Real Time System-Exclusive-Befehl 999
- Sequence Number (00) 1001
- Sequence/Track Name (03) 1001
- Sequencer Specific (127) 1003
- Set Temp (81) 1002
- Short-Format 1000
- SMPTE-Offset (84) 1003
- Song-Position-Pointer 997
- Song-Select 998
- Steuerbefehle 993
- System-Common-Befehle 997
- System-Exclusive-Befehle (SOX) 998
- Text (01) 1001
- Time Signature (88) 1003
- Timing 997
- Track-CHUNK 987
- Track-CHUNK Befehle 988
- Tune-Request 998
- Universal System Exclusive 1000

Miscellaneous Group 389

MOD

- Format 981
- Instrument Daten 983
- Noten-Block 982

MSP

- Datenbereich 820
- Header 819
- Indextabelle 820

O

OS/2 Bitmap Format

- Version 1.2 621
- Version 2.0 623

P

- PAL-Format 658
- Pattern Run-Format 688
- PBM-Formate (PBM, PGM, PNM, PPM) 823
- PBM-Header 823
- PCL
 - EPS (Encapsulated PostScript-Format) 1031
- PCPAINT/Pictor-Format (PIC) 837
- PCPAINT/Pictor-Header 837

PCX

- Abbildung in 826
- Bildkomprimierung 832
- Bitmap 825
- CGA Color Map speichern 829
- Color Map 829
- Color Planes (VGA) 830
- Datei (Hexdump) 833
- Dateiaufbau 834
- Datenkodierung 831
- Farbebenen (VGA) 830
- Farbpaletteninformation (CGA) 829
- Headeraufbau 827
- RLE-Verfahren 832
- Speicherung von Farbebenen 825

Periodizität 315

PGM-Header 823

Photoshop-Dateiformat s. PSD

PIC s. Micrografx-Formate

PICT

- Bilddaten-Records 780
- Datenbereich 778
- Header 778
- PicFrame-Record 779
- PicSize-Record 778
- Reserved Header-Record 779

Picture Element 681

Pixelkodierung 686

- Bit String-Record 687
- Pattern Run-Record 688
- Solid Run-Format 687
- Vertical Replication Count 689

PNG

- Background Color-CHUNK 847
- Bilddatenbereich 851
- CHUNK-Struktur 844
- Compressed Textual Data-CHUNK 851
- Dateiaufbau 843
- Daten-CHUNK 847
- Deflate/Inflate-Komprimierung 853
- Filterung 852
- Gamma-CHUNK 848
- Header-CHUNK 845
- Histogramm-CHUNK 848
- Image Last-Modification Time-CHUNK 850
- Interlacing 852
- Palette-CHUNK 846
- Physical Pixel Dimensions-CHUNK 848
- Primary Chromaticities-CHUNK 847
- Signatur 844
- Significant Bits-CHUNK 849
- Textual Data-CHUNK 849
- Trailer-CHUNK 847
- Transparency-CHUNK 850

Portable Document Format (PDF) 1067
PostScript 1031
 Anweisungen 1036
 arct 1047
 currentcmykcolor 1048
 currentcolor 1048
 currentcolorrendering 1049
 currentcolorscreen 1049
 currentcolorspace 1049
 currentcolortransfer 1049
 currentdevparams 1049
 currentglobal 1049
 currentgstate 1049
 currenthalftone 1050
 currentobjektformat 1050
 currentoverprint 1050
 currentpagedevice 1050
 currentshared 1050
 currentstrokeadjust 1050
 currentsystemparams 1050
 currentundercolorremoval 1050
 currentuserparams 1051
 defineuserobject 1051
 deletefile 1051
 EPS-Strukturkonventionen 1034
 execuserobject 1051
 filenameforall 1051
 filenameposition 1051
 filter 1052
 findeencoding 1052
 findresource 1052
 gcheck 1052
 globaldict 1052
 GlobalFontDirectory 1053
 glyphshow 1053
 ineofill 1053
 infill 1053
 instroke 1053
 inueofill 1053
 inufill 1054
 inustroke 1054
 ISOLatin1Encoding 1054
 languagelevel 1054
 makepattern 1054
 packedarray 1055
 printobject 1055
 product 1055
 realtime 1055
 rectclip 1055
 rectfill 1055
 rectstroke 1056
 rectviewclip 1056
 renamefile 1056

PostScript (Forts.)
 resourceforall 1056
 resourcestatus 1056
 rpptfont 1057
 scheck 1057
 selectfont 1057
 serialnumber 1057
 setbbox 1058
 setblackgeneration 1058
 setcachedevice2 1058
 setcacheparams 1058
 setcmykcolor 1058
 setcolor 1058
 setcolorrendering 1059
 setcolorscreen 1059
 setcolorspace 1059
 setcolortransfer 1059
 setdevparams 1059
 setfileposition 1059
 setglobal 1060
 setgstate 1060
 sethalftone 1060
 setobjectformat 1060
 setoverprint 1060
 setpacking 1061
 setpagedevice 1061
 setpattern 1061
 setrgbcolor 1061
 setshared 1061
 setsystemparams 1061
 setucacheparams 1062
 setundercolorremoval 1062
 setuserparams 1062
 setvmthreshold 1062
 shareddict 1062
 sharedfontdirectory 1062
 startjob 1062
 uappend 1063
 ucache 1063
 ucachestatus 1063
 ueofill 1063
 ufill 1063
 undef 1063
 undefinedresource 1063
 undefinefont 1063
 undefineresource 1063
 undefineuserobject 1064
 upath 1064
 UserObjects 1064
 ustroke 1064
 ustrokepath 1064
 vmreclaim 1064
 writeobject 1064

PostScript (Forts.)
 xyshow 1065
 yshow 1065
PPM-Header 824
printer point 713
PSD 533
 Bilddatenbereich 535
 Header 533
 MAC-Packbit-Kodierung 536
 Mode Data-Block 534
 Resource Data-Block 535

Q

QTM 855
 Media Header-Atom 860
 Media-Atom 859
 Movie Directory-Atom 856
 Movie Header-Atom 857
 Track Directory-Atom 858
 Track Header-Atom 858
Quattro Pro (WQ1)
 ANNDROP 275
 ANNGRID 274
 ANNHEAPDATA 271
 ANNINDEX 270
 ANNOBJECT 268
 ANNSNAPS 274
 ANNVIS 274
 ANNVISUAL 267
 Berechnungsformel 229
 BLANK 226
 BOF (Recordtyp 0000H) 219
 CALC_COUNT 258
 CALC_MODE 220
 CALC_ORDER 220
 CELL_PTR_INDEX 262
 COLUMN_WIDTH 225
 COLUMN_WIDTH_2 226
 CONDITION 266
 CURSOR_SYNC 221
 CURSOR_WINDOW_1_2 259
 DISPGRID 275
 EOF 220
 EXTNMRANGE 263
 EXTWKST 263
 Fließkommazahlen 227
 FONTS 264
 FOOTER 238
 FORMULA 229
 GNAME 273
 GRAPH 241
 H_RANGE 237
 HEADER 239

Quattro Pro (WQ1) (Forts.)
 HIDDEN_VECTOR1 260
 HIDDEN_VECTOR2 260
 INTEGER 227
 KEY_RANGE1 237
 KEY_RANGE2 238
 KRANGE3 267
 KRANGE4 267
 KRANGE5 267
 LABEL 228
 LABEL_FORMAT 240
 lineare Regressionsauswertungen 261
 MACOROLIB 263
 MARGINS 239
 MATRIX_RANGES 262
 NAME 226
 NAMED_GRAPH 249
 NUMBER 227
 OPTIMIZE 271
 ORIENTATION 273
 PARSE_RANGES 261
 PRINT_RANGE 236
 PROTECT 238
 QUERY_RANGE 235
 RHRANGE 276
 RHSIZE 276
 SAVE_RANGE 221
 SCALE 273
 SETUP 239
 SEXTGRAPH 272
 SLIDER 272
 SOLVE 272
 SORT_RANGE 236
 STRINGVAL 259
 STYLECELL 266
 STYLECPI 265
 TABLE 234
 TITLES 240
 UNFORMATTED 258
 Window_Split 221
 WINDOW1 222
 WINDOW2 226
 WINTL 264
 WKS_PASSWORD 260
QuickTime-Format s. QTM

R

RAS
 Datenbereich 863
 Header 861
 Palettenbereich 863
Resource Interchange File Format (RIFF) 599

RGB-Werte 828
Rich Text Format s. RTF
RIFF-Spezifikation 599
RLE (Run Length Encoding-Verfahren) 832
RTF 437
 Absatzabstände 478
 Absatzausrichtung 477
 Absatzformatierung 476
 Absatzpositionierung 485
 Absatzränder (Paragraph borders) 483
 Absatzschattierung (Paragraph Shading) 484
 Abschnittsformate 469
 Alternativer Font 444
 Anmerkungen 467
 Aufzählungen 480
 Basiskommandos 440
 Bidirektionale Ausrichtung 475
 Bidirektionale Kontrolle 479
 Bidirektionale Steuerung in Tabellen 489
 Bidirektionale Zeichensteuerung 469
 Bilddaten 503
 Bullets 480
 Callout-Elemente 507
 Character formatting 491
 Click-and-Type 467
 Columns 470
 control symbols 438
 control words 512
 Dateiaufbau 438
 Dateitabelle 447
 destination control words 440
 Document formatting 455
 Document Views und Zoom Level 457
 Dokumentbereich 454
 Dokumentformatierung 455
 Drawing Grid 468
 Drawing Objects 505
 drawing objects control words 505
 Eigenschaften 495
 Einrückung 478
 Farbtabelle 448
 Fontnummer 443
 Fonttabelle 442
 Formatvorlagen 448
 Formularoptionen 466
 Fuß- und Endnoten 458
 Grafiken 500
 Header 440
 Headers/Footers 475
 Kinsoku-Zeichen (FarEast) 467
 Kompatibilitätsoptionen 463
 Kontrollwörter 440

RTF (Forts.)
 Kopf- und Fußzeilen 475
 Kopf-/Fußzeilen 475
 Linked-Stil 463
 List Levels 451
 List Override Level 453
 List Override-Tabelle 453
 List Table 450
 Numerierungen 480
 objects control words 503
 Objekte 503
 Objektkontrollwörter 503
 Page Borders 468
 Paragraph Formatting 476
 picture control words 500
 Positionierung und Größenangaben 507
 Referenz auf Dokumente 479
 Revisionsinformationen 467
 RTF-Reader 438
 Schattierung 511
 Schattierung und Hintergrund der Tabellenzellen 490
 Section Break 470
 Section Formatting 469
 Section formatting 469
 Seiteneinstellung 462
 Seiteninformationen 471
 Seitennumerierung 472
 Seitenränder 468
 Sonderzeichen 439
 Spalten 470
 special control words 497
 Spezialzeichen 497
 Tabellendefinitionen 488
 Tabellenzeile 489
 Tabulatoren 479
 Textfeld 508
 Unicode-Unterstützung 441
 Vertikale Ausrichtung 475
 Zeichenformatierung 491
 Zeichensatz 440
 Zeichnungen 505
 Zeilennumerierung 471
 Zeilenumbruch in Positionsrahmen 487
Run Length Encoding-Verfahren (RLE) 832

S

Satztyp
 COL-FORMAT 326
 COMMENT 314
 DATA 313, 326
 DISPLAYUNITS 316

Satztyp (Forts.)

- GDISP-FORMAT 325
- LABEL 314
- MAJORSTART 315
- MINORSTART 315
- PERIODICITY 315
- ROW-FORMAT 326
- SIZE 315
- TABLE 312, 324
- TRUELENGTH 316
- TUPLES 313, 325
- UNITS 316
- VECTORS 312, 324

SDI (Super Data Interchange Format) 323

- COL-FORMAT 326
- Darstellungsformat einer Zeile 326
- DATA 326
- Data Definition Entry 329
- Datei 323
- Datenteil 327
- Endemarkierung 329
- Formeln 330
- Formula Entry 330
- GDISP-FORMAT 325, 326
- Header 323
- Level Display Formatting Entry 330
- Numeric Entry 328
- numerische Werte 328
- Origin Specifier 329
- Recordtypen 324
- Repeat Count Entry 330
- ROW-FORMAT 326
- Satzarten 327
- Steuerzeichen 323
- String mit Zellkoordinaten 329
- TABLE 324
- Text Entry 328
- TUPLES 325
- VECTORS 324
- Zeilenmarkierung 329

SGML 1071

- Anweisungen 1074
- Dateistruktur 1071
- Dokumentenstruktur 1072
- Kopfteil 1073
- Notation 1073

Solid Run Format 687

Sound

- CMF-Format 965
- MIDI Header 985
- MOD-Header 981
- RIFF-Header 1005
- Voice Format (VOC) 975

Sound (Forts.)

- WAV-DATA-CHUNK 1006
- WAV-FMT-CHUNK 1006

Spezialformat

- Kodierung 23

SPIFF

- Header 767

Standard Generalised Markup Language

- s. SGML

Standard MIDI-Format (SMF) 985

SUN Rasterformat s. RAS

Super Data Interchange-Format s. SDI

SYLK

- Bereich 289
- Boardered Flag 290
- Boundary-Record 284
- Cell-Format-Record 284
- Dateiende 292
- Defaultformat 282
- End of SYLK 292
- Ersatzname 289
- Expression 285
- Filename 289
- File-Substitution 289
- Formaterweiterungen für CHART 294
- Format-Record 280
- Formatsteuerzeichen 281
- Formelformat 282
- GA-Record 301
- GC-Record 298
- GD-Record 303
- GE-Record 300
- GF-Record 306
- GI-Record 305
- GL-Record 299
- GM-Record 308
- GN-Record 308
- GP-Record 302
- GR-Record 306
- GS-Record 298
- GT-Record 307
- GW-Record 304
- GX-Record 304
- GY-Record 304
- GZ-Record 308
- ID-Satz 279
- Kalkulationsformel 285
- Kommaformat 282
- Koordinaten 290
- Name 288
- Name Bereich 287
- Name Link 288
- Name Record 286

- SYLK (Forts.)
 - Protected 284
 - Quellbereich 289
 - Shared Expression 285
 - Shared Values 285
 - Spaltenadressierung 285
 - Spaltenformatierung 281
 - Split Window 290
 - Split Window horizontal 290
 - Split Window vertical 290
 - Textausrichtung 281
 - Widthformat 282
 - Window 289
 - Window Number 289
 - Zeilenadressierung 285
 - Zeilenformatierung 281
 - Zellformat 280
 - Zellkoordinaten 280, 284
 - Zellwert 284
- Symphony
 - Berechnungsformel 28
 - Dateiformat 17
 - DIF 311
 - Satzaufbau 17
- Symbolic Link-Format s. SYLK

T

- Tag Image File Format s. TIFF
- TARGA-Format (TGA) 865
 - Erweiterungen 873
 - Extension Area 874
 - Footer 876
 - Header 865
 - Palettendaten 869
 - Scan Line- und Color Correction-Tabellen 875
 - Stamp Image 875
 - Version 2.0 873
- Technical Graphics Extension (HP-GL/2)
 - Begin Plot 1025
 - Chord Tolerance Mode 1025
 - Download Character 1025
 - Enable Cutter 1025
 - Frame Advance 1025
 - Media Type 1026
 - Merge Control 1025
 - Message 1026
 - Not Ready 1026
 - Output Error 1026
 - Output Hard-Clip Limits 1026
 - Output Identification 1026
 - Output P1/P2 1026
 - Output Status 1026

- Technical Graphics Extension (Forts.)
 - Plot Size 1026
 - Quality Level 1027
 - Sort 1027
 - Velocity Select 1027
- TIFF
 - Artist-Tag 902
 - Aufbau 879
 - BitsPerSample-Tag 885
 - CellLength-Tag 889
 - CellWidth-Tag 889
 - ColorMap-Tag 904
 - ColorResponseCurve-Tag 901
 - ColorResponseUnit-Tag 900
 - Compression-Tag 886
 - DateTime-Tag 902
 - DocumentName-Tag 890
 - DOTRange-Tag 908
 - ExtrasSamples-Tag 908
 - FillOrder-Tag 889
 - FreeByteCount-Tag 897
 - FreeOffsets-Tag 897
 - GrayResponseCurve-Tag 898
 - GrayResponseUnit-Tag 897
 - HalftoneHints-Tag 905
 - Header 877
 - Host-Computer-Tag 902
 - Image File Directory (IFD) 877, 878
 - ImageDescriptor-Tag 890
 - ImageLength-Tag 885
 - ImageWidth-Tag 885
 - InkNames-Tag 907
 - InkSet-Tag 906
 - JPEGACTables-Tag 915
 - JPEGDCTables-Tag 915
 - JPEGInterchangeFormatLength-Tag 913
 - JPEGInterchangeFormat-Tag 913
 - JPEGLossLessPredictors-Tag 914
 - JPEGPointTransforms-Tag 914
 - JPEGProc-Tag 912
 - JPEGQTables-Tag 915
 - JPEGRestartInterval-Tag 913
 - Komprimierungsverfahren 917
 - Make-Tag 891
 - MaxSampleValue-Tag 894
 - MinSampleValue-Tag 894
 - Model-Tag 891
 - NewSubFile-Tag 883
 - NumberOfInks-Tag 907
 - Orientation-Tag 892
 - PageName-Tag 896
 - PageNumber-Tag 900
 - PhotometricInterpretation-Tag 887

TIFF (Forts.)

- PlanarConfiguration-Tag 895
- Predictor-Tag 903
- PrimaryChromaticities-Tag 904
- ReferenceBlackWhite-Tag 912
- ResolutionUnit-Tag 899
- RowsPerStrip-Tag 893
- SampleFormat-Tag 909
- SamplesPerPixel-Tag 893
- SMaxSampleValue-Tag 909
- SMinSampleValue-Tag 909
- Software-Tag 901
- Strip Offset Pointer 892
- Strip Offset-Tags 886
- StripByteCounts-Tag 893
- StripOffset-Tag 891
- Subfile Type-Tag 884
- T4Options-Tag 898
- T6Options-Tag 899
- Tag-Beschreibung 883
- Tags 879
- TargetPrinter-Tag 908
- Thresholding-Tag 888
- TileByteCount-Tag 906
- TileLength-Tag 905
- TileOffsets-Tag 906
- TileWidth-Tag 905
- TransferRange-Tag 910
- White Point-Tag 903
- XPosition-Tag 896
- XResolution-Tag 894
- YCbCrCoefficient-Tag 910
- YCbCrPositioning-Tag 911
- YCbCrSubSampling-Tag 911
- YPosition-Tag 896
- YResolution-Tag 895

twips 500

V

Variable Length Multibyte-Steuercodes 357, 403

- Advance to Page Position 373
- Auto Reference Definition 385
- Auto Reference Tag 386
- Begin marked text 381
- Begin style 395
- Box Group 391
- Character/Space Width 373
- Color 363
- Comment 390
- Conditional End of Page 389
- Date Function 387
- Define Columns 365

Variable Length Multibyte-Steuercodes (Forts.)

- Define marked text 382
- Define Math Columns 365
- Display Group 387
- Embedded Printer Command 389
- End marked text 382
- End of included Subdocumen 387
- End of Line 377
- End of Page 376
- End style 396
- Endnote 380
- Endnote Options 367
- Endnotes print here 384
- Figure 391
- Font Change 364
- Fontauswahl 363
- Footer A 379
- Footer B 379
- Footnote 380
- Footnote Options 366
- Footnote/Endnote Group 380
- Force odd/even Page 373
- Form 362
- Format Group 376
- Generate Group 381
- Global ON 396
- Graph Box Information 377
- Graph Box Options 368, 370
- Gruppendifinition 364
- Header A 378
- Header B 379
- Header/Footer Group 378
- Horizontal line 394
- Hyphenation Zone Set 359
- Include Subdocument 386
- Index entry 383
- Kerning 390
- Left/Right Margin Set 358
- Line Numbering 372
- Miscellaneous Group 389
- Overstrike 388
- Page Number Position 361
- Paragraph Number 388
- Paragraph Number Definitions 366
- Save Page Information 384
- Set Alignment Character 370
- Set Endnote Number 371
- Set Footnote Number 371
- Set Graph Box Number 375
- Set Group Subfunktionen 370
- Set Language 375
- Set Lines per Inch 357
- Set Page Number 372

- Variable Length Multibyte-Steuercodes (Forts.)
 - Set Underline Mode 370
 - Space Extension 374
 - Spacing Set 358
 - Start of included Subdocum 386
 - Style Group 395
 - Style OFF 397
 - Tab Set 359
 - Table 393
 - Table of authority entry 383
 - Text box 393
 - Top/Bottom Margin Set 360
 - User defined text box 394
 - Vertical line 395
- Variable Length Multibyte-Steuercodes (WP 5.1)
 - Beginning of Column 429
 - Beginning of ROW 430
 - Enhanced Merge 431
 - Equation Nested 434
 - Style Group 428
 - Table End of Line Codes Group 429
 - Table End of Pages Codes Group 431
 - Table Off at 431
 - Unknown Function 435
 - Vertical line 428
- VOC
 - ASCII-Text-Block (05) 978
 - Continuation-Block (02) 977
 - Data-Block (01) 976
 - End Repeat Loop-Block (07) 979
 - Extended-Block (08) 979
 - Format 975
 - Header 975
 - Marker-Block (04) 978
 - Repeat Loop-Block (06) 979
 - Silence-Block (03) 978
 - Terminator Block (00) 976
 - Datenblöcke 976
- W**
 - WAV 1005
 - DATA-CHUNK 1006
 - FMT-CHUNK 1006
 - Header 1005
 - WebCGM 633
 - Window_Split 20
 - WINDOW_WIDTH_2 24
 - WINDOWS 2.0 PAINT File-Format s. MSP
 - Windows Bitmap-Format s. BMP
 - Windows Metafile-Format s. WMF
 - Windows RLE-Format (RLE) 619
 - Windows WAV-Format s. WAV
 - WK1 s. WKS/WK1
 - WK3
 - BOF-Record 53
 - CALCSET-Record 57
 - COLUMNWIDTH-Record 61
 - CPA 84
 - DATAFILLNUMS-Record 64
 - DTLABELCELL-Record 84
 - DTLABELMISC-Record 83
 - DUPFMT-Record 74
 - EOF-Record 57
 - ERRCELL-Record 74
 - FILESEAL-Record 63
 - Format 17
 - FORMAT-Record 72
 - FORMULA-Record 77
 - FORMULASTRING-Record 83
 - FRM-Fileformat 86
 - GBLFMT-Record 73
 - GRAPHMAIN-Record 68
 - GRAPHSTRING-Record 71
 - GRAPHWINDOW 84
 - HIDDENCOLUMN-Record 61
 - HIDDENSHEET-Record 86
 - LABEL-Record 75
 - LPTAUTO-Record 85
 - NACELL-Record 75
 - NUMBERCELL-Record 76
 - PASSWORD-Record 57
 - PRINTMAIN-Record 65
 - PRINTSTRING-Record 67
 - QUERY-Record 85
 - SHEETCELLPTR-Record 59
 - SHEETLAYOUT-Record 60
 - SMALLNUMCELL-Record 76
 - SORTKEYDIR-Record 63
 - SYSTEMRANGE-Record 62
 - USERRANGE-Record 62
 - WINDOWSET-Record 58
 - XFORMAT 83
 - ZEROFORCE-Record 63
 - WKS/WK1
 - Abmessungen Druckrand 39
 - aktuelle Cursorposition 22
 - Aufbau 17
 - Berechnungsformel 28
 - Bereichsname 24
 - BLANK 25
 - BOF 18
 - CALC_COUNT 47
 - CALC_MODE 19

WKS/WK1 (Forts.)

CALC_ORDER 19
 CELL_PTR_INDEX 51
 COLUMN_WIDTH_1 24
 COLUMN_WIDTH_2 24
 CURSOR_SYNC 20
 CURSOR_WINDOW_1_2 48
 Dateien 17
 Datentabelle 32
 DIF 311
 Header 312
 Distribution Range 35
 EOF 18
 FILL_RANGE 35
 Fließkommazahlen 26
 FOOTER 37
 Format 17
 Formatierung der Druckausgabe 48
 Formelberechnung 20
 FORMULA 28
 Fußzeile 37
 Ganzzahlen 25
 Grafik 43
 Grafikerstellung 39
 GRAPH 39
 H_RANGE 35
 HEADER 37
 HIDDEN_VECTOR1 49
 HIDDEN_VECTOR2 49
 hidden-Spalte 49
 INTEGER 25
 KEY_RANGE1 35
 KEY_RANGE2 36
 Kodierung 23, 25
 Kopfzeile 37
 LABEL 27
 LABEL_FORMAT 38
 lineare Regressionsauswertungen 50
 MARGINS 38
 MATRIX_RANGES 51
 NAME 24
 NAMED_GRAPH 43
 NUMBER 26
 PARSE_RANGES 50
 PRINT_RANGE 34
 PROTECT 36
 QUERY_RANGE 33
 Randbegrenzungen 38
 Rechenblattausschnitt 34
 Recordtypen 18
 REGRESS_RANGES 50
 Satzaufbau 17
 SAVE_RANGE 20

WKS/WK1 (Forts.)

Schutz des Arbeitsblatts 36
 SDI 323
 SETUP 38
 Setup-Zeichensatz 38
 SORT_RANGE 34
 Spaltenbreite 24
 SYLK 279
 TABLE 32
 TITLES 39
 UNFORMATTED 48
 WINDOW_SPLIT 20
 Window_Split 20
 WINDOW1 21, 22
 WINDOW2 24
 WK3-Format 53
 WKS_PASSWORD 48
 Zahlenformat 25
 Zellformat 22

WMF

AnimatePalette 929
 BitBlt 930
 CHORD 931
 CREATEBITMAP 931
 CREATEBITMAPINDIRECT 931
 CREATEBRUSH 932
 CREATEBRUSHINDIRECT 932
 CREATEFONTINDIRECT 933
 CREATEPALETTE 933
 CREATEPATTERNBRUSH 934
 CREATEPENINDIRECT 934
 CREATEREGION 935
 DELETEOBJECT 935
 DRAWTEXT 936
 ELLIPSE 936
 Enhanced Metafile-Format (EMF) 946
 ESCAPE 936
 EXCLUDECLIPRECT 937
 EXTTEXTOUT 937
 Header 925
 LINETO 937
 Metafile-Records 926
 MOVETO 938
 OFFSETCLIPRGN 938
 OFFSETVIEWPORTORG 938
 OFFSETWINDOWORG 938
 PAINTREGION 938
 PATBLT 939
 PIE 939
 Placable Metafiles 926
 POLYGON 939
 POLYLINE 939
 POLYPOLYGON 940

WMF (Forts.)

- Records 926
- RECTANGLE 940
- RESICEPALETTE 940
- ROUNDRECT 941
- SCALEVIEWPORT 941
- SCALEWINDOWEXT 941
- SETBKCOLOR 942
- SETBKMODE 942
- SETDIBTODEV 942
- SETPALENTRIES 942
- SETPIXEL 943
- SETPOLYFILLMODE 943
- SETROP2 943
- SETSTRETCHBLTMODE 943
- SETTEXTALIGN 943
- SETTEXTCHAREXTRA 943
- SETTEXTCOLOR 943
- SETTEXTJUSTIFICATION 943
- SETWINDOWEXT 944
- SETWINDOWORG 944
- STRETCHBLT 944
- STRETCHDIB 945
- TEXTOUT 946

Word Perfect Graphic File Format s. WPG

WordPerfect 345, 350

- 5.1-Format 397
- anwenderdefinierter Block (Graph Box) 370
- Box Group 391, 423
 - Subfunktionen 423
- Codebelegung 352
- Datenbereich 350
- Display Group 387
- Drucker 389
- Einlesen von Bildern 391
- Farbe für Textausgabe 363
- Festlegung der Fonts 364
- Fixed Length Multibyte-Steuercodes 354
- Fixed Length Multibyte-Steuercodes (5.1) 401
- Fontauswahl 363
- Format des Indexblocks 347
- Format Group 376
- Fußnoten 366
- Generate Group 381
- Grafikdaten 350
- Graph Box Options 368
- Gruppendefinition 364
- Header 345, 397
- Header-Präfix 345, 397
- Indizes 348, 399
- Literaturhinweise 367, 371

WordPerfect (Forts.)

- Miscellaneous Group 389
- Multibyte-Steuercodes 352
- Pakettypen 348, 399
- Paragraph Number 388
- Präfix 345
- Seitennumerierung 361, 372
- Set Group 370
- Single Byte Functions (5.1) 400
- Spaltendefinition 365
- Steuercodes 354
- Style Group 395
- Subfunktionen 363, 364, 370, 376, 378, 380, 381, 387, 389, 391, 395
- Tabellen (Graph Boxes) 370
- Textbereich (5.1) 400
- Textblock (Graph Boxes) 370
- Textformat 362
- Trennung 359
- Variable Length Multibyte-Steuercodes 357, 403
- WPG-Records 951
- Zeichenbreite 373
- Zeichensätze 375
- Zeilennumerierung 372
- Zeilenrand 360

World Wide Web 1089

WPG (Word Perfect Graphic File Format) 951

- Bitmap 957
- Bitmap 2 962
- Color Map 959
- Ellipse 956
- End WPG Data 960
- Fill Attributes 953
- Graphic Text 958
- Graphic Text Attributes 958
- Graphics Text 2 963
- Headeraufbau 951
- Kurve 961
- Line 955
- Marker Attributes 954
- Output Attributes 961
- Polygon 956
- Polyline 955
- Polymarker 955
- PostScript Data 960
- Records 951
- Rectangle 956
- Start Bild 962
- Start Chart 963
- Start WPG Data 960
- Start WPG2 964

X

XML 1081

Attribute 1084

CDATA-Abschnitte 1085

Dateistruktur 1081

Elemente 1083

Entity 1084

Extended Backus-Naur Form (EBNF) 1086

Kommentare 1085

Z

Zeichensätze

ISO 646-7-Bit 1071

PCX-Dateien 834

Zeichensatz 444

ZSoft Paintbrush File Format s. PCX

