CTfile Formats

December 1999

MDL®
Information Systems, Inc.

December 1999

U.S. GOVERNMENT RESTRICTED RIGHTS NOTICE

Table of Contents

Chapter 1 **Introduction**

Change Log 1-1
Standard CTfiles 1-2

Chapter 2 **The Connection Table [CTAB]**

The Counts Line 2-3
The Atom Block 2-4
The Bond Block 2-6
The Atom List Block 2-7
The Stext Block. 2-7
The Properties Block. 2-8
The Properties Block for 3D Features. 2-19
    3D features count line. 2-21
    3D features detail lines 2-21
    3D data constraints 2-31
Stereo Notes 2-33

Chapter 3 **Molfiles**

The Header Block. 3-2

Chapter 4 **RGfiles**

Chapter 5 **SDfiles**

    SDfile after a CFS search 5-4

Chapter 6 **Rxnfiles**

Header Block 6-1
Reactants/Products 6-3
Molfile Blocks. 6-3

Chapter 7 RDfiles

RDfile Header 7-2
Molecule and Reaction Identifiers 7-2
Data-field Identifier 7-3
Data 7-3

Chapter 8 Atom Limit Enhancements

Phantom Extra Atom 8-1
Superatom Attachment Point 8-2
Superatom Class 8-3
Large REGNO 8-3
Sgroup Bracket Style 8-3

Chapter 9 Moving CTfiles On and Off the Clipboard in ISIS

Clipboard Objects 9-1
Hints on Creating a Reader/Writer For CT 9-2
Copying from the Clipboard 9-2
  Copying to the clipboard. 9-3
Sample Code For Copying or Pasting a CTfile in MS Windows 9-4

Chapter 10 The Extended Molfile Format

Specifications For Atom and Bond Descriptions 10-3
  Conventions 10-4
The Extended Connection Table 10-5
  CTAB block 10-5
  Counts line 10-5
  Atom block. 10-6
  Bond block. 10-9
  Link atom line 10-11
  Sgroup block 10-12
  3D block 10-20
The Extended Rgroup Query Molfile 10-23
  Rgroup block 10-24
  Rgroup logic lines. 10-26

Chapter 1

Introduction

MDL Information Systems supports a number of file formats for representation and communication of chemical information. This document describes the formats for MDL's CTfiles (chemical table files):

- Part I (Chapters 2 through 9) describes the standard CTfile formats.
- Part II (Chapter 10) describes the extended molfile format. All extended molfiles can be easily identified by the "V3000" version stamp in the header portion of the file. You are most likely to encounter the extended molfile format in CTfiles written from ISIS/Host or ISIS/Desktop version 2.0 or higher.

Change Log

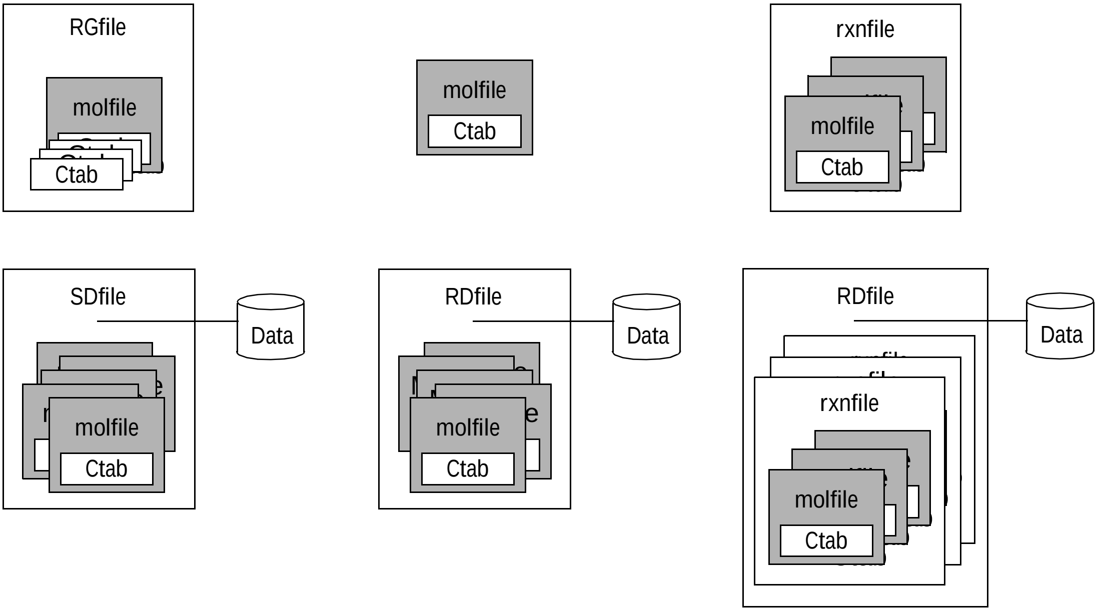
The following are the changes in this document:

Change	**Page(s)**
December, 1999	
Updated entries in "Atom List"	2-11
December, 1998	
Updated "Example of an SDfile"	5-3
August, 1998	
Added STBOX field	10-9
June, 1997	
Added Atom Attachment Order	2-11
Added new ATTCHORD field	10-6, 10-8

Change	**Page(s)**
October, 1996	
Minor corrections	2-3, 2-8
Enhanced description of connection table properties block	2-8
Added Sgroup bracket style	8-3, 10-14, 10-19,

Standard CTfiles

The following figure illustrates the relationship between the various file formats described below:



molfiles — Molecule files: Each molfile describes a single molecular structure which can contain disjoint fragments.

RGfiles — Rgroup files: An RGfile describes a single molecular query with Rgroups. Each RGfile is a combination of Ctabs defining the root molecule and each member of each Rgroup in the query.

rxnfiles

Reaction files: Each rxnfile contains the structural information for the reactants and products of a single reaction. MDL currently has two types of rxnfiles: the REACCS type and the CPSS-rxnfile written by CPSS programs (CPSS-rxnfiles are not described in this document.) CPSS programs cannot read a REACCS rxnfile; however, REACCS can read and write CPSS-rxnfiles for transfer to CPSS.

SDfiles

Structure-data files: An SDfile contains structures and data for any number of molecules. Together with RDfiles, SDfiles are the primary format for large-scale data transfer between MDL databases.

RDfiles

Reaction-data files: Similar to SDfiles in concept, the RDfile is a more general format that can include reactions as well as molecules, together with their associated data. Although RDfiles are used primarily by ISIS and REACCS, MACCS-II can also read and write RDfiles except for the reaction structure information (indicated by the square brackets in Table 1-1). CPSS reads and writes RDfiles with embedded molfiles and CPSS-rxnfiles (indicated by the curly brackets in Table 1-1).

Table 1-1 shows which CTfiles MDL programs can read and write.

Table 1-1 MDL Program

CTfile Type	MACCS-II	REACCS	ISIS	CPSS
molfiles	+	+	+	+
RGfiles	+		+	
rxnfiles		+	+	{+}
SDfiles	+		+	+
RDfiles	[+]	+	+	{+}

Some of the structural and query properties described in this document are generic in their applicability, while others are peculiar to certain CTfile types (see Table 1-2). The applicability of each property is identified in subsequent chapters by the icons shown in Table 1-2.

Table 1-2 Properties and identifying icons applicable to various CTfile types

Icon	Property	molfile	RGfile	SDfile	rxnfile	RDfile
G	Generic	+	+	+	+	+
Sg	Sgroup	+	+	+		
Rg	Rgroup	+	+	+		
3D	3D	+	+	+		
CP	CPSS	+		+	+	+
Rx	Reaction				+	+
Q	Query	+	+		+	

PART I
Standard File Formats

Chapter 2

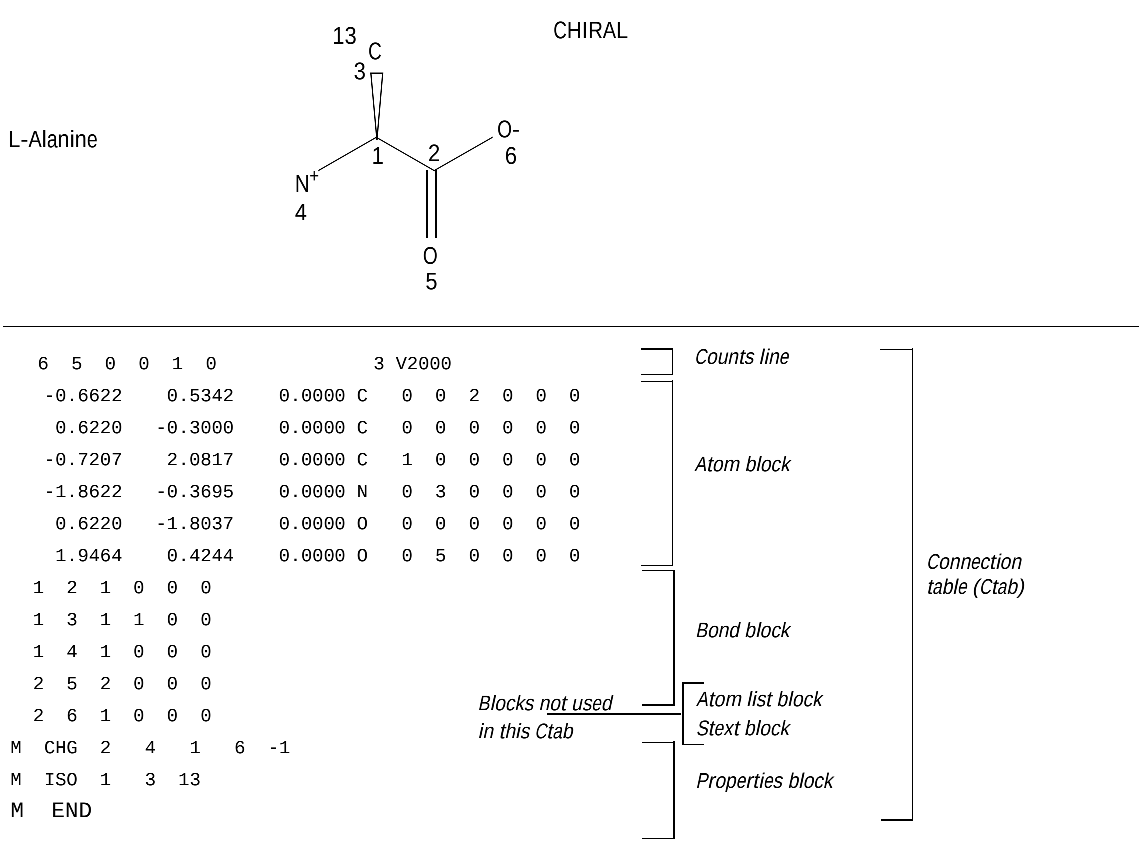
The Connection Table [CTAB]

Ctab

A connection table (Ctab) contains information describing the structural relationships and properties of a collection of atoms. The atoms may be wholly or partially connected by bonds. Such collections may, for example, describe molecules, molecular fragments, substructures, substituent groups, polymers, alloys, formulations, mixtures, and unconnected atoms. The connection table is fundamental to all of MDL's file formats.

Figure 2-1 shows the connection table of a simple molecule (alanine) with the various data blocks identified. The connection table corresponds to the following alanine molecule. The atom numbers on the structure correspond to atom numbers in the Ctab. An atom number is assigned according to the order of the atom in the Atom Block.

Figure 2-1 Connection table organization illustrated using alanine



```
  6  5  0  0  1  0            3 V2000
   -0.6622    0.5342    0.0000 C   0  0  2  0  0  0
    0.6220   -0.3000    0.0000 C   0  0  0  0  0  0
   -0.7207    2.0817    0.0000 C   1  0  0  0  0  0
   -1.8622   -0.3695    0.0000 N   0  3  0  0  0  0
    0.6220   -1.8037    0.0000 O   0  0  0  0  0  0
    1.9464    0.4244    0.0000 O   0  5  0  0  0  0
  1  2  1  0  0  0
  1  3  1  1  0  0
  1  4  1  0  0  0
  2  5  2  0  0  0
  2  6  1  0  0  0
M  CHG  2   4   1   6  -1
M  ISO  1   3  13
M  END
```

Counts line

Atom block

Bond block

Blocks not used in this Ctab

Atom list block

Stext block

Properties block

Connection table (Ctab)

The format for a Ctab block is:

- Counts line: Important specifications here relate to the number of atoms, bonds, and atom lists, the chiral flag setting, and the Ctab version.
- Atom block: Specifies the atomic symbol and any mass difference, charge, stereochemistry, and associated hydrogens for each atom.
- Bond block: Specifies the two atoms connected by the bond, the bond type, and any bond stereochemistry and topology (chain or ring properties) for each bond.
- Atom list block: Identifies the atom (number) of the list and the atoms in the list.

- Stext (structural text descriptor) block: Used by ISIS and CPSS programs.
- Properties block: Provides for future expandability of Ctab features, while maintaining compatibility with earlier Ctab configurations.

The detailed format for each block outlined above follows:

Note: A blank *numerical* entry on any line should be read as “0” (zero). Spaces are significant and correspond to one or more of the following:

- Absence of an entry
- Empty character positions within an entry
- Spaces between entries; single unless specifically noted otherwise

The Counts Line

```
aaabbblllfffcccsssxxxrrrpppiiimmmvvvvvv
```

Where:

aaa	= number of atoms (current max 255)*	G
bbb	= number of bonds (current max 255)*	G
lll	= number of atom lists (max 30)*	Q
fff	= (obsolete)	
ccc	= chiral flag: 0=not chiral, 1=chiral	G
sss	= number of stext entries	CP
xxx	= number of reaction components + 1	CP
rrr	= number of reactants	CP
ppp	= number of products	CP
iii	= number of intermediates	CP
mmm	= number of lines of additional properties, including the `M  END` line. No longer supported and default set to 999	G
vvvvvv	= Ctab version: `‘V2000’` or `‘V3000’`	G

* These limits apply to MACCS-II, REACCS, and the ISIS/Host Reaction Gateway, but *not* to the ISIS/Host Molecule Gateway or ISIS/Desktop.

For example, the counts line in the Ctab shown in Figure 2-1 shows six atoms, five bonds, the CHIRAL flag *on*, and three lines in the properties block:

```
  6  5  0  0  1  0              3 V2000
```

The Atom Block

The Atom Block is made up of atom lines, one line per atom with the following format:

```
xxxxx.xxxxyyyyy.yyyyzzzzz.zzzz aaaddcccssshhhbbbvvvHHHrrriiimmmnnneee
```

where the values are described in Table 2-1.

Table 2-1 Meaning of values in the atom block

Field	Meaning	Values	Notes
`x y z`	atom coordinates		G
`aaa`	atom symbol	entry in periodic table or L for atom list, A, Q, * for unspecified atom, and LP for lone pair, or R# for Rgroup label	G Q G 3D Rg
`dd`	mass difference	-3, -2, -1, 0, 1, 2, 3, 4 (0 if value beyond these limits)	G Difference from mass in periodic table. Wider range of values allowed by M ISO line, below. Retained for compatibility with older Ctabs, M ISO takes precedence.
`ccc`	charge	0 = uncharged or value other than these, 1 = +3, 2 = +2, 3 = +1, 4 = doublet (^), 5 = -1, 6 = -2, 7 = -3	G Wider range of values in M CHG and M RAD lines below. Retained for compatibility with older Ctabs, M CHG and M RAD lines take precedence.
`sss`	atom stereo parity	0 = not stereo, 1 = odd, 2 = even, 3 = either or unmarked stereo center	G Ignored when read. See stereo notes on page 2-33.
`hhh`	hydrogen count + 1	1 = H0, 2 = H1, 3 = H2, 4 = H3, 5 = H4	Q H0 means no H atoms allowed unless explicitly drawn. Hn means atom must have *n* or more H's in excess of explicit H's.
`bbb`	stereo care box	0 = ignore stereo configuration of this double bond atom, 1 = stereo configuration of double bond atom must match	Q Double bond stereochemistry is considered during SSS only if both ends of the bond are marked with stereo care boxes.

Table 2-1 Meaning of values in the atom block (Continued)

Field	Meaning	Values	Notes
vvv	valence	0 = no marking (default) (1 to 14) = (1 to 14) 15 = zero valence	G Shows number of bonds to this atom, including bonds to implied H's.
HHH	H0 designator	0 = not specified, 1 = no H atoms allowed	CP Redundant with hydrogen count information. May be unsupported in future releases of MDL software.
rrr	reaction component type	reactant = 1, product = 2, intermediate = 3	CP
iii	reaction component number	0 to (n-1)	CP
mmm	atom-atom mapping number	1 - number of atoms	Rx
nnn	inversion/retention flag	0 = property not applied 1 = configuration is inverted, 2 = configuration is retained,	Rx
eee	exact change flag	0 = property not applied, 1 = change on atom must be exactly as shown	Rx Q

Note: With Ctab version V2000, the dd and ccc fields have been superseded by the M ISO, M CHG, and M RAD lines in the properties block, described below. For compatibility, all releases since MACCS-II 2.0, REACCS 8.1, and ISIS 1.0:

- Write appropriate values in both places if the values are in the old range.
- Use the atom block fields if there are no M ISO, M CHG, or M RAD lines in the properties block.

Support for these atom block fields may be removed in future releases of MDL software.

The Bond Block

The Bond Block is made up of bond lines, one line per bond, with the following format:

```
111222tttsssxxxrrrccc
```

where the values are described in Table 2-2.

Table 2-2 Meaning of values in the bond block

Field	Meaning	Values	Notes
`111`	first atom number	1 - number of atoms	G
`222`	second atom number	1 - number of atoms	G
`ttt`	bond type	1 = Single, 2 = Double, 3 = Triple, 4 = Aromatic, 5 = Single or Double, 6 = Single or Aromatic, 7 = Double or Aromatic, 8 = Any	Q Values 4 through 8 are for SSS queries only.
`sss`	bond stereo	Single bonds: 0 = not stereo, 1 = Up, 4 = Either, 6 = Down, Double bonds: 0 = Use x-, y-, z-coords from atom block to determine cis or trans, 3 = Cis or trans (either) double bond	G The wedge (pointed) end of the stereo bond is at the first atom (Field `111` above)
`xxx`	not used		
`rrr`	bond topology	0 = Either, 1 = Ring, 2 = Chain	Q SSS queries only.
`ccc`	reacting center status	0 = unmarked, 1 = a center, -1 = not a center, Additional: 2 = no change, 4 = bond made/broken, 8 = bond order changes 12 = 4+8 (both made/broken and changes); 5 = (4 + 1), 9 = (8 + 1), and 13 = (12 + 1) are also possible	Rx (query only)

The Atom List Block Q

Note: Newer programs use the M ALS item in the properties block in place of the atom list block. The atom list block is retained for compatibility, but information in an M ALS item supersedes atom list block information.

Made up of atom list lines, one line per list, with the following format:

```
aaa kSSSSn 111 222 333 444 555
```

where:

`aaa`	= number of atom (L) where list is attached
`k`	= T = [NOT] list, F = normal list
`n`	= number of entries in list; maximum is 5
`111...555`	= atomic number of each atom on the list
`S`	= space

The Stext Block CP

The Stext Block is made up of two-line entries with the following format:

```
xxxxx.xxxxyyyyy.yyyy
TTTT...
```

where:

`x y`	= stext coordinate
`T`	= stext text

The Properties Block

The Properties Block is made up of `mmm` lines of additional properties, where `mmm` is the number in the counts line described above. If a version stamp is present, `mmm` is ignored and the file is read until an `M  END` line is encountered. Currently `mmm` is no longer supported and set to 999 as the default.

Most lines in the properties block are identified by a prefix of the form `M  XXX` where two spaces separate the `M` and `XXX`. Exceptions are:

- `A  aaa`, `V  aaa vvvvvv`, and `G  aaappp,` which indicate ISIS and CPSS properties: atom alias, atom value, and group abbreviation (called residue in ISIS), respectively. **CP**
- `S  SKPnnn` which causes the next `nnn` lines to be ignored.

The prefix: `M  END` terminates the properties block.

Variables in the formats can change properties but keep the same letter designation. For example, on the Charge, Radical, or Isotope lines, the "uniformity" of the `vvv` designates a general property identifier. On Sgroup property lines, the `sss` uniformity is used as an Sgroup index identifier.

All lines that are not understood by the program are ignored.

The descriptions below use the following conventions for values in field widths of 3:

`n15`	number of entries on line; value = 1 to 15
`nn8`	number of entries on line; value = 1 to 8
`nn6`	number of entries on line; value = 1 to 6
`nn4`	number of entries on line; value = 1 to 4
`nn2`	number of entries on line; value = 1 or 2
`nn1`	number of entries on line; value = 1
`aaa`	atom number; value = (1 to number of atoms)

The format for the properties included in this block follows. The format shows one entry; ellipses (. . .) indicate additional entries.

Atom Alias CP

```
A  aaa
x...
```

`aaa:`	Atom number
`x...`	Alias text

Atom Value CP

```
V  aaa v...
```

`aaa:` Atom number

`v...` Value text

Group Abbreviation CP

```
G  aaappp
x...
```

`aaa:` Atom number

`ppp:` Atom number

`x...` Abbreviation label.

Abbreviation is required for compatibility with CPSS. CPSS allowed abbreviations with only one attachment. The attachment is denoted by two atom numbers, aaa and ppp. All of the atoms on the aaa side of the bond formed by aaa-ppp are abbreviated. The coordinates of the abbreviation are the coordinates of aaa. The text of the abbreviation is on the following line (x...). In current versions of ISIS, abbreviations can have any number of attachments and are written out using the Sgroup appendixes. However, any ISIS abbreviations that do have one attachment are also written out in the CPSS-style, again for compatibility with CPSS, but this behavior might not be supported in future versions.

Charge G

```
M  CHGnn8 aaa vvv ...
```

`vvv:` -15 to +15. Default of 0 = uncharged atom. When present, this property supersedes *all* charge and radical values in the atom block, forcing a 0 charge on all atoms not listed in an M CHG or M RAD line.

Radical G

```
M  RADnn8 aaa vvv ...
```

`vvv:` Default of 0 = no radical, 1 = singlet (:), 2 = doublet (^), 3 = triplet (^^).When present, this property supersedes *all* charge and radical values in the atom block, forcing a 0 (zero) charge and radical on all atoms not listed in an M CHG or M RAD line.

Isotope G

```
M  ISOnn8 aaa vvv ...
```

`vvv:` Absolute mass differing from natural abundance (as specified by PTABLE.DAT) within the range -18 to +12. When present, this property supersedes *all* isotope values in the atom block. Default (no entry) is natural abundance.

Ring Bond Count Q

```
M  RBDnn8 aaa vvv ...
```

`vvv:` Number of ring bonds allowed: default of 0 = off, -1 = no ring bonds (r0),-2 = as drawn (r*); 2 = (r2), 3 = (r3), 4 or more = (r4).

Substitution Count Q

```
M  SUBnn8 aaa vvv ...
```

`vvv:` Number of substitutions allowed: default of 0 = off, -1 = no substitution (s0),-2 = as drawn (s*); 1, 2, 3, 4, 5 = (s1) through (s5), 6 or more = (s6).

Unsaturated Atom Q

```
M  UNSnn8 aaa vvv ...
```

`vvv:` At least one multiple bond: default of 0 = off, 1 = on.

Link Atom Q

```
M  LINnn4 aaa vvv bbb ccc ...
```

vvv,bbb,ccc: Link atom (aaa) and its substituents, other than bbb and ccc, may be repeated 1 to vvv times, (vvv > = 2).

Atom List Q

```
M  ALS aaannn e 11112222333344445555...
```

aaa: Atom number, value = (1 to #atoms).

nnn: Number of entries on line (16 maximum).

e: Exclusion, value is T if a ’NOT’ list, F if a normal list.

1111...: Atom symbol of list entry in field of width 4.

Note: This line contains the atom symbol rather than the atom number used in the atom list block. Any data found in this item supersedes data from the atom list block. The number of entries can exceed the fixed limit of *5* in the atom list block entry.

Attachment Point Rg

```
M  APOnn2 aaa vvv ...
```

vvv: Indicates whether atom aaa of the Rgroup member is the first attachment point (vvv = 1), second attachment point (vvv = 2), both attachment points (vvv = 3); default of 0 = no attachment.

Atom Attachment Order Rg

```
M  AAL aaann2 111 v1v 222 v2v ...
```

aaa: Atom index of the Rgroup usage atom

nn2: Number of pairs of entries that follow on the line

111: Atom index of a neighbor of aaa

v1v: Attachment order for the aaa-111 bond

222: Atom index of a neighbor of aaa

v2v: Attachment order for the aaa-222 bond

Note: v1v and v2v are either 1 or 2 for the simple doubly attached Rgroup member.

This appendix provides explicit attachment list order information for R# atoms. The appendix contains atom neighbor index and atom neighbor value pairs. The atom neighbor value information identifies the atom neighbor index as the *ith* attachment. The implied ordering in V2000 molfiles is by atom index order for the neighbors of Rgroup usage atoms. If atom index order conflicts with the desired neighbor ordering at the R# atom, this appendix allows you to override to this default order.

If v1v=1 and v2v=2, ISIS/Host only writes this appendix if 111 is greater than 222. Note, however, that the attachment values can be written in any order.

Rgroup Label Location Rg

```
M  RGPnn8 aaa rrr ...
```

`rrr:` Rgroup number, value from 1 to 32, labels position of Rgroup on root.

Rgroup Logic, Unsatisfied Sites, Range of Occurrence Rg

```
M  LOGnn1 rrr iii hhh ooo
```

`rrr:` Rgroup number, value from 1 to 32.

`iii:` Number of another Rgroup which must only be satisfied if `rrr` is satisfied (IF `rrr` THEN `iii`).

`hhh:` RestH property of Rgroup `rrr`; default is 0 = off, 1 = on. If this property is applied (on), sites labeled with Rgroup `rrr` may only be substituted with a member of the Rgroup or with H.

`ooo:` Range of Rgroup occurrence required: n = exactly n, n - m = n through m,> n = greater than n, < n = fewer than n, default (blank) is > 0. Any non-contradictory combination of the preceding values is also allowed; for example: 1, 3-7, 9, >11.

Sgroup Type Sg

```
M  STYnn8 sss ttt ...
```

`sss:` Sgroup number.

`ttt:` SUP = superatom, MUL = multiple group, SRU = SRU type, MON = monomer, MER = Mer type, COP = copolymer, CRO = crosslink, MOD = modification, GRA = graft, COM = component, MIX = mixture, FOR = formulation, DAT = data Sgroup, ANY = any polymer, GEN = generic.

Note: For a given Sgroup, an STY line giving its type must appear before any other line that supplies information about it. For a data Sgroup, an SDT line must describe the data field before the SCD and SED lines that contain the data (see Data Sgroup Data below). When a data Sgroup is linked to another Sgroup, the Sgroup must already have been defined.

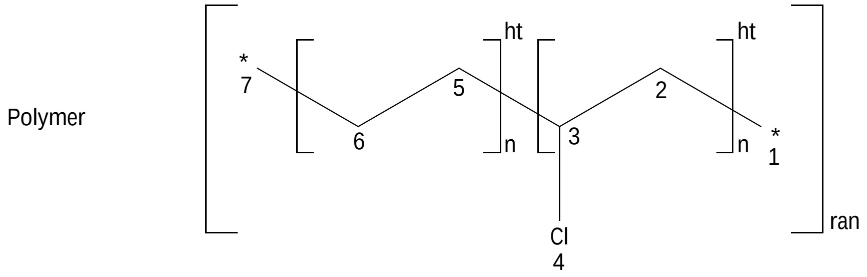
Sgroups can be in any order on the Sgroup Type line. Brackets are drawn around Sgroups with the M SDI lines defining the coordinates.

Sgroup Subtype Sg

```
M  SSTnn8 sss ttt ...
```

`ttt:` Polymer Sgroup subtypes: ALT = alternating, RAN = random, BLO = block.

Figure 2-2 Ctab organization of an Sgroup structure



```
GSMACCS-II10179110412D 1   0.00374     0.00000     0

  7  6  0  0  0  0            16 V2000
    2.9463    0.3489    0.0000 *   0  0  0  0  0  0
    1.6126    1.1189    0.0000 C   0  0  0  0  0  0
    0.2789    0.3489    0.0000 C   0  0  3  0  0  0
    0.2789   -1.1911    0.0000 Cl  0  0  0  0  0  0
   -1.0548    1.1190    0.0000 C   0  0  0  0  0  0
   -2.3885    0.3490    0.0000 C   0  0  0  0  0  0
   -3.9246    1.1470    0.0000 *   0  0  0  0  0  0
  1  2  1  0  0  0
  2  3  1  0  0  0
  3  4  1  0  0  0
  5  6  1  0  0  0
  5  3  1  0  0  0
  7  6  1  0  0  0
M  STY  3   1 SRU   2 SRU   3 COP
M  SST  1   3 RAN
M  SLB  3   1   5   2   6   3   7
M  SCN  2   1 HT    2 HT
M  SAL   1  2   5   6
M  SBL   1  2   5   6
M  SDI   1  4   -0.6103    1.2969   -0.6103    0.1710
M  SDI   1  4   -3.1565    0.1850   -3.1565    1.3110
M  SAL   2  3   2   3   4
M  SBL   2  2   1   5
M  SDI   2  4    2.2794    1.2969    2.2794    0.1709
M  SDI   2  4   -0.1657    0.1710   -0.1657    1.2969
M  SAL   3  7   1   2   3   4   5   6   7
M  SDI   3  4    3.6382    1.6391    3.6382   -1.7685
M  SDI   3  4   -4.7070   -1.7685   -4.7070    1.6391
M  END
```

Number of entries on line

Header block (see Chapter 3)

Counts line

Atom block

Bond block

Atom list block

Stext block

Ctab block

Type

Subtype

Label

Connectivity

General Sgroup Info

Sgroup 1

Sgroup 2

Sgroup 3

Sgroup properties

Sgroup Labels Sg

```
M  SLBnn8 sss vvv ...
```

`vvv:` Unique Sgroup identifier (for MACCS-II only, the integer label is from 1-512).

Sgroup Connectivity Sg

```
M  SCNnn8 sss ttt ...
```

`ttt:` HH = head-to-head, HT = head-to-tail, EU = either unknown. Left justified.

Sgroup Expansion Sg

```
M  SDS EXPn15 sss ...
```

`sss:` Sgroup index of expanded superatoms.

Sgroup Atom List Sg

```
M  SAL sssn15 aaa ...
```

`aaa:` Atoms in Sgroup `sss`.

Sgroup Bond List Sg

```
M  SBL sssn15 bbb ...
```

`bbb:` Bonds in Sgroup `sss`. (For data Sgroups, `bbb’s` are the containment bonds, for all other Sgroup types, `bbb`’s are crossing bonds.)

Multiple Group Parent Atom List Sg

```
M  SPA sssn15 aaa ...
```

`aaa:` Atoms in paradigmatic repeating unit of multiple group `sss`.

Note: To ensure that all current molfile readers consistently interpret chemical structures, multiple groups are written in their fully expanded state to the molfile. The `M SPA` atom list is a subset of the full atom list that is defined by the Sgroup Atom List `M SAL` entry.

Sgroup Subscript Sg

```
M  SMT sss m...
```

m...: Text of subscript Sgroup sss. (For multiple groups, m... is the text representation of the multiple group multiplier. For superatoms, m... is the text of the superatom label.

Sgroup Correspondence Sg

```
M  CRS sssnn6 bb1 bb2 bb3
```

bb1, bb2: Crossing bonds that share a common bracket.

bb3: Crossing bond in repeating unit that connect to bond bb1.

Sgroup Display Information Sg

```
M  SDI sssnn4 x1 y1 x2 y2
```

x1,y1,x2,y2: Coordinates of bracket endpoints (FORTRAN format 4F10.4).

Superatom Bond and Vector Information Sg

```
M  SBV sss bb1 x1 y1
```

bb1: Bond connecting to contracted superatom.

x1,y1: Vector for bond bb1 connecting to contracted superatom sss (FORTRAN format 2F10.4).

Data Sgroup Field Description Sg

```
M  SDT sss fff...fffgghhh...hhhiijjj...
```

sss: Index of data Sgroup.

fff...fff: 30 character field name (in MACCS-II no blanks, commas, or hyphens).

gg: Field type (in MACCS-II F = formatted, N = numeric, T = text).

hhh...hhh: 20-character field units or format.

`ii:` Nonblank if data line is a query rather than Sgroup data, MQ = MACCS-II query, IQ = ISIS query, *P*Q = *program name code* query.

`jjj...:` Data query operator (blank for MACCS-II).

Data Sgroup Display Information Sg

```
M  SDD sss xxxxx.xxxxyyyyy.yyyy eeefgh i jjjkkk ll m  noo
```

`sss:` Index of data Sgroup.

`x,y:` Coordinates (2F10.4).

`eee:` (Reserved for future use.)

`f:` Data display, A = attached, D = detached.

`g:` Absolute, relative placement, A = absolute, R = relative.

`h:` Display units, blank = no units displayed, U = display units.

`i:` (Reserved for future use.)

`jjj:` Number of characters to display (1...999 or ALL).

`kkk:` Number of lines to display (unused, always 1).

`ll:` (Reserved for future use.)

`m:` Tag character for tagged detached display (if non-blank).

`n:` Data display DASP position (1...9). (MACCS-II only)

`oo:` (Reserved for future use.)

Data Sgroup Data Sg

```
M  SCD sss d...
M  SED sss d...
```

`d...:` Line of data for data Sgroup `sss` (69 chars per line, columns 12-80)

Note: A line of data is entered as text in 69-character substrings. Each SCD line adds 69 characters to a text buffer (starting with successive SCDs at character positions 1, 70, and 139). Following zero or more

SCDs must be an SED, which may supply a final 69 characters. The SED initiates processing of the buffered line of text: trailing blanks are removed and right truncation to 200 characters is performed, numeric and formatted data are validated, and the line of data is added to data Sgroup sss. Left justification is not performed.

A data Sgroup may have more than one line of data, so more than one set of SCD and SED lines can be present for the same data Sgroup. The lines are added in the same order that they are encountered.

If 69 or fewer characters are to be entered on a line, they may be entered with a single SED not preceded by an SCD. On the other hand, if desired a line may be entered to a maximum of 3 SCDs followed by a blank SED that terminates the line. The set of SCD and SED lines describing one line of data for a given data Sgroup must appear together, with no intervening lines for other data Sgroups' data.

Sgroup Hierarchy Information Sg

```
M  SPLnn8 ccc ppp ...
```

`ccc:` Sgroup index of the child Sgroup.

`ppp:` Sgroup index of the parent Sgroup (ccc and ppp must already be defined via an STY line prior to encountering this line).

Sgroup Component Numbers Sg

```
M  SNCnn8 sss ooo ...
```

`sss:` Index of component Sgroup.

`ooo:` Integer component order (1...256). This limit applies only to MACCS-II.

3D Feature Properties 3D

```
M  $3Dnnn
```

`M  $3D...` See below for information on the properties block of a 3D molfile. These lines must all be contiguous.

End of Block

```
M  END
```

This entry goes at the end of the properties block and is required for molfiles which contain a version stamp in the counts line.

The Properties Block for 3D Features 3D

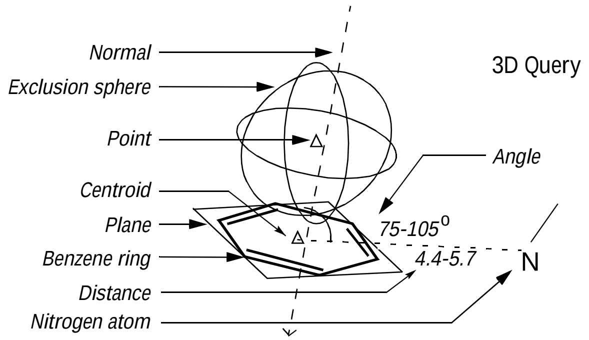
For each 3D feature, the properties block includes:

- One 3D features count line
- One or more 3D features detail lines

The characters `M  $3D` appear at the beginning of each line describing a 3D feature. The information for 3D features starts in column 7.

Figure 2-3 illustrates the molfile corresponding to the following 3D query:

Figure 2-3 Ctab organization of a 3D query



```
3D Query                                                       Header block (see Chapter 3)
  MACCS-II10179109553D 1   1.00000     0.00000     0

  8  7  0  0  0  0            18 V2000                          Counts line
    1.0252    0.2892    1.1122 C   0  0  0  0  0  0             Atom block
   -0.4562    0.6578    1.3156 C   0  0  0  0  0  0
   -1.4813    0.3687    0.2033 C   0  0  0  0  0  0
   -1.0252   -0.2892   -1.1122 C   0  0  0  0  0  0
    0.4562   -0.6578   -1.3156 C   0  0  0  0  0  0
    1.4813   -0.3687   -0.2033 C   0  0  0  0  0  0
    4.1401   -0.1989    1.3456 N   0  0  0  0  0  0
    4.6453    0.5081    1.7417 C   0  0  0  0  0  0
  1  2  1  0  0  0                                              Bond block
  2  3  2  0  0  0
  3  4  1  0  0  0
  4  5  2  0  0  0
  5  6  1  0  0  0
  6  1  2  0  0  0
  7  8  1  0  0  0                                              Atom list block
M  $3D  7                     3D Features Count                 Stext block
M  $3D -7  6                  Centroid                          Properties block
M  $3D  3
M  $3D  6  4  2
M  $3D -5 13                  Plane
M  $3D  6    0.0000
M  $3D  1  2  3  4  5  6
M  $3D -8  7                  Normal to Plane
M  $3D  9 10
M  $3D -3  6                  Point
M  $3D  9 11   -2.0000
M  $3D-16 12                  Exclusion sphere
M  $3D 12  1  0    1.5000
M  $3D-12 10                  Angle
M  $3D 12  9  7   75.0000  105.0000
M  $3D -9  3                  Distance
M  $3D  7  9    4.4000    5.7000
M  END
```

Ctab block (counts line through M END)

3D features count line

The first line in the properties block is the 3D features count line and has the following format:

```
M  $3Dnnn
```

where nnn is the number of 3D features on a model.

3D features detail lines

The lines following the 3D features count line describe each 3D feature on a model. Each 3D feature description consists of an identification line and one or more data lines:

- The identification line is the first line and contains the 3D feature's type identifier, color, and name.
- Each data line describes the construction of the 3D feature.

Identification line

The 3D feature identification line has the following format:

```
M  $3Dfffccc aaa...aaa ttt...ttt
```

where the variables represent:

fff	3D feature type
ccc	Color number (an internal MDL number which is terminal dependent)
aaa...aaa	3D feature name (up to 32 characters)
ttt...ttt	Text comments (up to 32 characters) used by MDL programs (see 3D data constraints on page 2-31)

Table 2-3 lists the 3D feature type identifiers.

Table 2-3 3D feature type identifiers

Identifier	Meaning
-1	Point defined by two points and a distance (in Angstroms)
-2	Point defined by two points and a percentage
-3	Point defined by a point, a normal line, and a distance
-4	Line defined by two or more points (A best fit line if more than two points)
-5	Plane defined by three or more points (A best fit plane if more than three points)
-6	Plane defined by a point and a line
-7	Centroid defined by points
-8	Normal line defined by a point and a plane
-9	Distance defined by two points and a range (in Angstroms)
-10	Distance defined by a point, line, and a range (in Angstroms)
-11	Distance defined by a point, plane, and a range (in Angstroms)
-12	Angle defined by three points and a range (in degrees)
-13	Angle defined by two intersecting lines and a range (in degrees)
-14	Angle defined by two intersecting planes and a range (in degrees)
-15	Dihedral angle defined by 4 points and a range (in degrees)
-16	Exclusion sphere defined by a point and a distance (in Angstroms)
-17	Fixed atoms in the model
nnn	A positive integer indicates atom or atom-pair data constraints

Data line

The 3D feature defines the data line format. Each 3D object is treated as a pseudoatom and identified in the connection table by a number. The 3D object numbers are assigned sequentially, starting with the next number greater than the number of atoms. The data line formats for the 3D feature types are:

Type **Description of Data Line**

-1 The data line for a point defined by two points and a distance (Å) has the following format:

```
M  $3Diiijjjddddd.dddd
```

where the variables represent:

`iii`	ID number of a point
`jjj`	ID number of a second point
`ddddd.dddd`	Distance from first point in direction of second point (Å), 0 if not used

The following example shows POINT_1 created from the atoms 1 and 3 with a constraint distance of 2Å.

The first line is the identification line. The second line is the data line.

```
M  $3D -1  4 POINT_1
M  $3D  1  3    2.0000
```

-2 The data line for a point defined by two points and a percentage has the format:

```
M  $3Diiijjjddddd.ddddd
```

where the variables represent:

`iii`	ID number of a point
`jjj`	ID number of a second point
`ddddd.dddd`	Distance (fractional) relative to distance between first and second points, 0 if not used

Type **Description of Data Line**

-3 The data line for a point defined by a point, a normal line, and a distance (Å) has the format:

```
M  $3Diiillldddddd.dddd
```

where the variables represent:

`iii`	ID number of a point
`lll`	ID number of a normal line
`ddddd.dddd`	Distance (Å), 0 if not used

Note: For chiral models, the distance value is signed to specify the same or opposite direction of the normal.

-4 The data lines for a best fit line defined by two or more points have the following format:

```
M  $3Dppptttttt.tttt
M  $3Diiijjj...zzz . . .
```

where the variables represent:

`ppp`	Number of points defining the line
`ttttt.tttt`	Deviation (Å), 0 if not used.
`iii`	Each `iii`, `jjj`, and `zzz` is the ID number `jjj` of an item in the model that defines the line
`jjj`	
`...`	
`zzz`	(to maximum of 20 items per data line)

The following line is defined by the four points 1, 14, 15, and 19 and has a deviation of 1.2Å. The first line is the identification line. The second and third lines are the data lines.

```
M  $3D -4  2 N_TO_AROM
M  $3D  4    1.2000
M  $3D  1 14 15 19
```

Type **Description of Data Line**

-5 The data lines for a plane defined by three or more points (a best fit plane if more than three points) have the following format:

```
M  $3Dppptttttt.tttt
M  $3Diiijjj...zzz
 ...
```

where the variables represent:

`ppp`	Number of points defining the line
`ttttt.tttt`	Deviation (Å), 0 if not used.
`iii`	Each `iii`, `jjj`, and `zzz` is the ID number `jjj` of an item in the model that defines the line
`jjj`	
`...`	
`zzz`	(to maximum of 20 items per data line)

The following line is defined by the four points 1, 14, 15, and 19 and has a deviation of 1.2Å. The first line is the identification line. The second and third lines are the data lines.

```
M  $3D -5  4 PLANE_2
M  $3D  3
M  $3D  1 5 14
```

-6 The data line for a plane defined by a point and a line has the following format:

```
M  $3Diiilll
```

where the variables represent:

`iii`	ID number of a point
`lll`	ID number of a line

The following plane is defined by the point 1 and the plane 16. The first line is the identification line. The second line is the data line.

```
M  $3D -6  3 PLANE_1
M  $3D  1 16
```

Type	Description of Data Line

-7 The data lines of a centroid defined by points have the following format:

```
M  $3Dppp
M  $3Diiijjj...zzz ...
```

where the variables represent:

ppp	Number of points defining the centroid
iii	Each iii, jjj, and zzz is the ID number jjj of an item in the model that defines the centroid
jjj	
...	
zzz	(maximum of 20 items per data line).

The following centroid, ARO_CENTER, is defined by 3 items: 6, 8, and 10. The first line is the identification line. The second and third lines are the data lines.

```
M  $3D -7  1 ARO_CENTER
M  $3D  3
M  $3D  6  8 10
```

-8 The data line for a normal line defined by a point and a plane has the following format:

```
M  $3Diiijjj
```

where the variables represent:

iii	ID number of a point
jjj	ID number of a plane

The following normal line, ARO_NORMAL, is defined by the point 14 and the plane 15. The first line is the identification line. The second line is the data line.

```
M  $3D -8  1 ARO_NORMAL
M  $3D 14 15
```

Type **Description of Data Line**

-9 The data line for a distance defined by two points and a range (Å) has the following format:

```
M  $3Diiijjjddddd.ddddzzzzz.zzzz
```

where the variables represent:

`iii`	ID number of a point
`jjj`	ID number of a second point
`ddddd.dddd`	Minimum distance (Å)
`zzzzz.zzzz`	Maximum distance (Å)

The following distance, L, is between items 1 and 14 and has a minimum distance of 4.9Å and a maximum distance of 6.0Å. The first line is the identification line. The second line is the data line.

```
M  $3D -9  6 L
M  $3D  1 14    4.9000    6.0000
```

-10 The data line for a distance defined by a point, line, and a range (Å) has the format:

```
M  $3Diiilllddddd.ddddzzzzz.zzzz
```

where the variables represent:

`iii`	ID number of a point
`lll`	ID number of a line
`ddddd.dddd`	Minimum distance (Å)
`zzzzz.zzzz`	Maximum distance (Å)

-11 The data line for a distance defined by a point, plane, and a range (Å) has the format:

```
M  $3Diiijjjddddd.ddddzzzzz.zzzz
```

where the variables represent:

`iii`	ID number of a point
`jjj`	ID number of a plane
`ddddd.dddd`	Minimum distance (Å)
`zzzzz.zzzz`	Maximum distance (Å)

Type **Description of Data Line**

-12 The data line for an angle defined by three points and a range (in degrees) has the following format:

```
M  $3Diiijjjkkkddddd.ddddzzzzz.zzzz
```

where the variables represent:

iii	ID number of a point
jjj	ID number of a second point
kkk	ID number of a third point
ddddd.dddd	Minimum degrees
zzzzz.zzzz	Maximum degrees

The following angle, THETA1, is defined by the three points: 5, 17, and 16. The minimum angle is 80° and the maximum is 105°. The first line is the identification line. The second line is the data line.

```
M  $3D-12  5 THETA1
M  $3D  5 17 16   80.0000  105.0000
```

-13 The data line for an angle defined by two lines and a range (in degrees) has the following format:

```
M  $3Dlllmmmddddd.ddddzzzzz.zzzz
```

where the variables represent:

lll	ID number of a line, mmm ID number of a second line
ddddd.dddd	Minimum degrees
zzzzz.zzzz	Maximum degrees

THETA2 is defined by the lines 27 and 26 with maximum and minimum angles of 45° and 80°. The first line is the identification line. The second line is the data line.

```
M  $3D-13  5 THETA2
M  $3D 27 26   45.0000   80.0000
```

Type **Description of Data Line**

-14 The data line for an angle defined by two planes and a range (in degrees) has the following format:

```
M  $3Diiijjjddddd.ddddzzzzz.zzzz
```

where the variables represent:

`iii`	ID number of a plane
`jjj`	ID numbers of a second plane
`ddddd.dddd`	Minimum degrees
`zzzzz.zzzz`	Maximum degrees

-15 The data line for a dihedral angle defined by four points and a range (in degrees) has the following format:

```
M  $3Diiijjjkkkl lldddd.ddddzzzzz.zzzz
```

where the variables represent:

`iii`	ID number of a point
`jjj`	ID number of a second point
`kkk`	ID number of a third point
`lll`	ID number of a fourth point
`ddddd.dddd`	Minimum degrees
`zzzzz.zzzz`	Maximum degrees

DIHED1 is defined by the items 7, 6, 4, and 8 with minimum and maximum angles of 45° and 80°, respectively. The first line is the identification line. The second line is the data line.

```
M  $3D-15  5 DIHED1
M  $3D  7  6  4  8   45.0000   80.0000
```

Type **Description of Data Line**

-16 The data lines for an exclusion sphere defined by a point and a distance (Å) have the following format:

```
M  $3Diiiuuuaaaddddd.dddd
M  $3Dbbbccc...zzz ...
```

where the variables represent:

`iii`	ID number of the center of the sphere
`uuu`	1 or 0. 1 means unconnected atoms are ignored within the exclusion sphere during a search; 0 otherwise
`aaa`	Number of allowed atoms
`ddddd.dddd`	Radius of sphere (Å)
`bbb`	Each `bbb`, `ccc`, and `zzz`
`ccc`	is an ID number of an allowed atom.
`...`	
`zzz`	(to maximum of 20 items per data line)

The following exclusion sphere is centered on point 24, has a radius of 5, and allows atom 9 within the sphere. The first line is the identification line. The second and third lines are the data lines.

```
M  $3D-16  7 EXCL_SPHERE
M  $3D 24  0  1    5.0000
M  $3D  9
```

Type **Description of Data Line**

-17 The data lines of the fixed atoms have the following format:

```
M  $3Dppp
M  $3Diiijjj...zzz ...
```

where the variables represent:

ppp	Number of fixed points
iii	Each iii, jjj, and zzz is an ID number of a fixed atom
jjj	
...	
zzz	(to maximum of 20 items per data line)

The following examples shows 4 fixed atoms. The first line is the identification line. The second and third lines are the data lines.

```
M  $3D-17
M  $3D  4
M  $3D  3  7 12 29
```

3D data constraints 3D Q

A positive integer is used as a type identifier to indicate an atom or atom-pair data constraint. Two lines are used to describe a data constraint. The lines have the following format:

```
M  $3Dnnncccaaa...aaabbbbbbbbpppppppppsss...sss
M  $3Diiijjjddd...ddd
```

where the variables represent:

nnn	Database-field number
ccc	Color
aaa...aaa	Database-field name (up to 30 characters)
bbbbbbbb	/BOX = box-number (source of data) (up to 8 characters)
ppppppppp	/DASP = n1, n2 where n1 and n2 are digits from 1-9 (data size and position) (up to 9 characters)

`sss...sss` /DISP = 3DN (name), 3DV (value), 3DQ (query), NOT (no text)

First three in any combination to maximum total of 15 characters

`iii` ID number of an atom

`jjj` ID number of a second atom for atom-pair data, 0 if data is atom data

`ddd...ddd` Data constraint (based on format from database) (up to 64 characters)

ISIS 3D data query syntax and MACCS-II 3D data query syntax are not identical. The ISIS data query requires a search operator, a blank space, then one or more operands. For more information on ISIS data query syntax, see the ISIS Help system entries on SBF (Search By Form) or QB (Query Builder) for entering text in a query. For information on MACCS-II data searches, see the *MACCS-II Command Language Reference*.

Note: For MACCS-II, the atom number 999 stands for all atoms. The MACCS-II wild card character (@) can be used in the data constraints.

The following example shows a numeric data constraint for the field CNDO.CHARGE on atom 12. The first line is the identification line. The second line is the data line.

```
M  $3D  7  0 CNDO.CHARGE
M  $3D 12  0   -0.3300   -0.1300
```

The following example shows a numeric data constraint for the field BOND.LENGTH on the atom pair 1 and 4. The first line is the identification line. The second line is the data line.

```
M  $3D  9  0 BOND.LENGTH
M  $3D  1  4    2.0500    1.8200
```

The following example shows a data constraint allowing any charge value for the field CHARGE on all the atoms. The first line is the identification line. The second line is the data line.

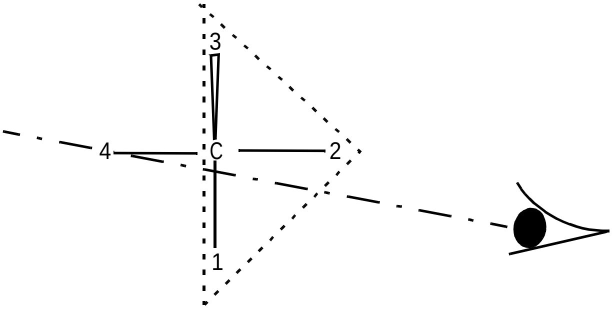
```
M  $3D 12  0 CHARGE
M  $3D999  0 @
```

Stereo Notes

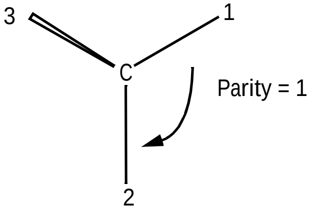
Parity can be illustrated as follows:

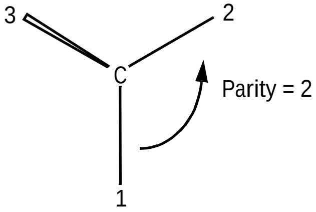
Mark a bond attached at a stereo center Up or Down to define the configuration. Number the atoms surrounding the stereo center with 1, 2, 3, and 4 in order of increasing atom number (position in the atom block) (a hydrogen atom should be considered the highest numbered atom, in this case atom 4). View the center from a position such that the bond connecting the highest-numbered atom (4) projects behind the plane formed by atoms 1, 2, and 3.

Note: In the figure, atoms 1, 2, and 4 are all in the plane of the paper, and atom 3 is above the plane.



Sighting towards atom number 4 through the plane (123), you see that the three remaining atoms can be arranged in either a clockwise or counterclockwise direction in ascending numerical order.

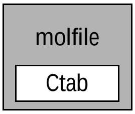




The Ctab lists a parity value of 1 for a clockwise arrangement at the stereo center and 2 for counterclockwise. A center with an Either bond has a parity value of 3. An unmarked stereo center is also assigned a value of 3. The first example above has a parity value of 2.

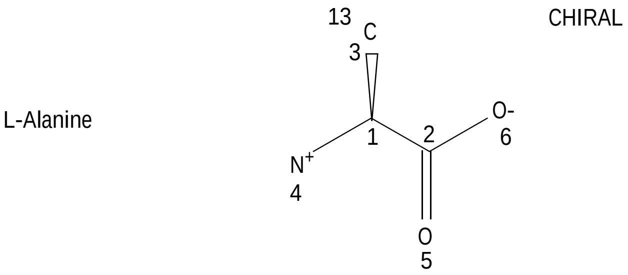
Chapter 3

Molfiles

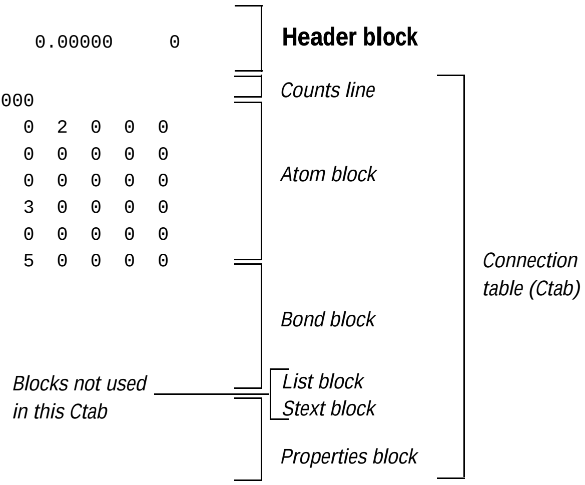


A molfile consists of a header block and a connection table. Figure 3-1 shows a molfile for alanine corresponding to the following structure:

Figure 3-1 Molfile organization illustrated using alanine



```
L-Alanine (13C)
GSMACCS-II10169115362D 1   0.00366     0.00000     0

  6  5  0  0  1  0            3 V2000
   -0.6622    0.5342    0.0000 C   0  0  2  0  0  0
    0.6220   -0.3000    0.0000 C   0  0  0  0  0  0
   -0.7207    2.0817    0.0000 C   1  0  0  0  0  0
   -1.8622   -0.3695    0.0000 N   0  3  0  0  0  0
    0.6220   -1.8037    0.0000 O   0  0  0  0  0  0
    1.9464    0.4244    0.0000 O   0  5  0  0  0  0
  1  2  1  0  0  0
  1  3  1  1  0  0
  1  4  1  0  0  0
  2  5  2  0  0  0
  2  6  1  0  0  0
M  CHG  2   4   1   6  -1
M  ISO  1   3  13
M  END
```



The format for a molfile is:

- Header block: This identifies the molfile with the molecule name, user's name, program, date, and other miscellaneous information and comments
- Ctab block (described in Chapter 2)

The detailed format for the header block follows.

The Header Block

Line 1: Molecule name. This line is unformatted, but like all other lines in a molfile may not extend beyond column 80.

Caution: This line must not contain any of the reserved tags that identify any of the other CTAB file types such as $MDL (RGfile), $$$$ (SDfile record separator), $RXN (rxnfile), or $RDFILE (RDfile headers).

Line 2: User's first and last initials (I), program name (P), date/time (M/D/Y,H:m), dimensional codes (d), scaling factors (S, s), energy (E) if modeling program input, internal registry number (R) if input through MDL form. This line has the format:

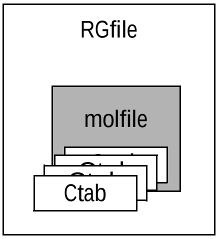
```
                  IIPPPPPPPPMMDDYYHHmmddSSssssssssssEEEEEEEEEEEERRRRRR
(FORTRAN:         A2  A8    <--A10--->A2I2   F10.5     F12.5     I6   )
```

A blank line can be substituted for line 2.

Line 3: A line for comments. If no comment is entered, a blank line must be present.

Chapter 4

RGfiles



The format of an RGfile (Rgroup query file) is shown below. Lines beginning with $ define the overall structure of the Rgroup query; the molfile header block is embedded in the Rgroup header block.

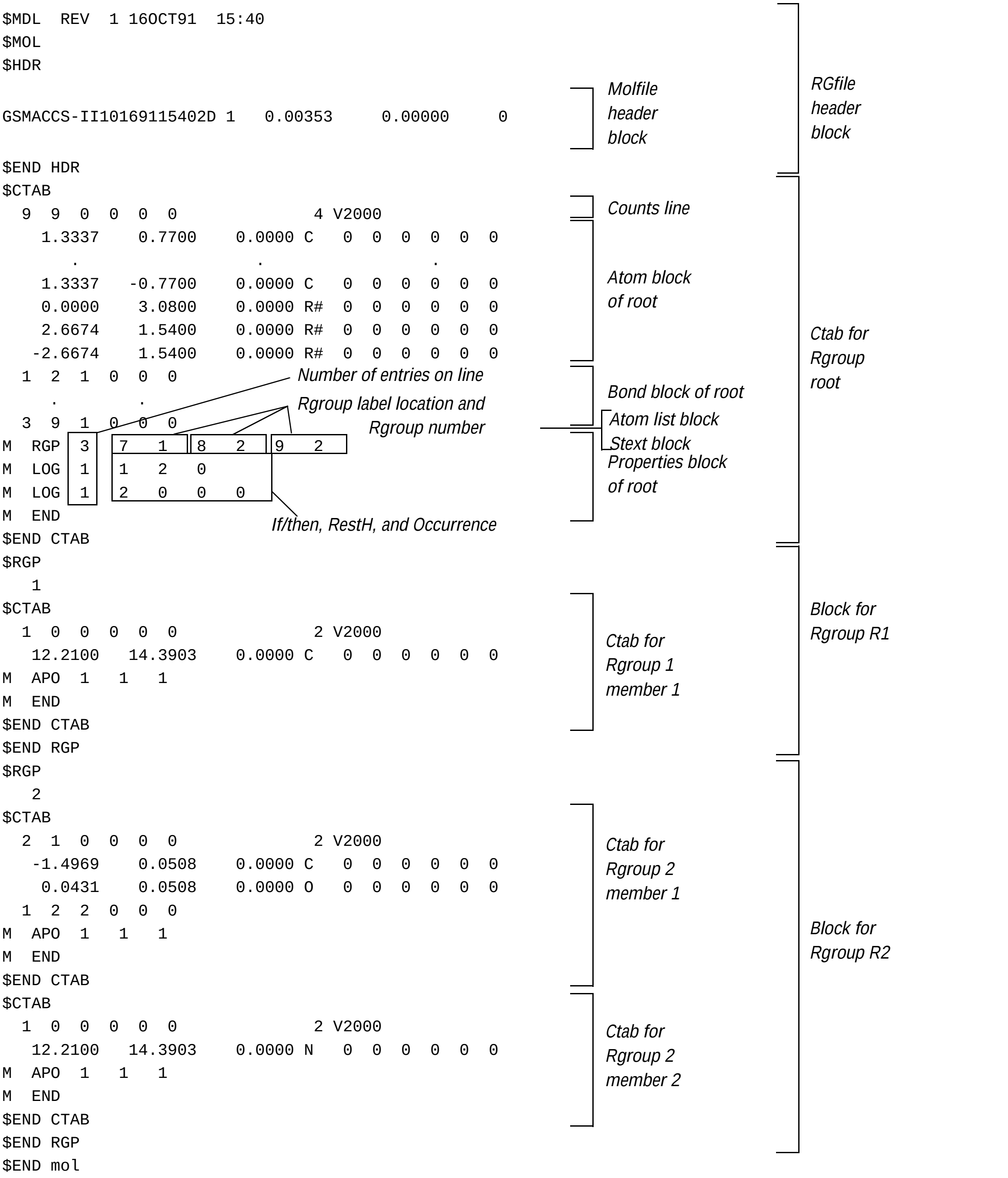
In addition to the primary connection table (Ctab block) for the root structure, a Ctab block defines each member (*m) within each Rgroup (*r).

```
            $MDL REV 1 date/time
            $MOL
            $HDR
            [Molfile Header Block (see Chapter 3) = name, pgm info, comment]
            $END HDR
            $CTAB
            [Ctab Block (see Chapter 2) = count + atoms + bonds + lists + props
            $END CTAB
            $RGP
            rrr   [where rrr = Rgroup number]
 *r         $CTAB
       *m   [Ctab Block]
            $END CTAB
            $END RGP
            $END mol
```

where:

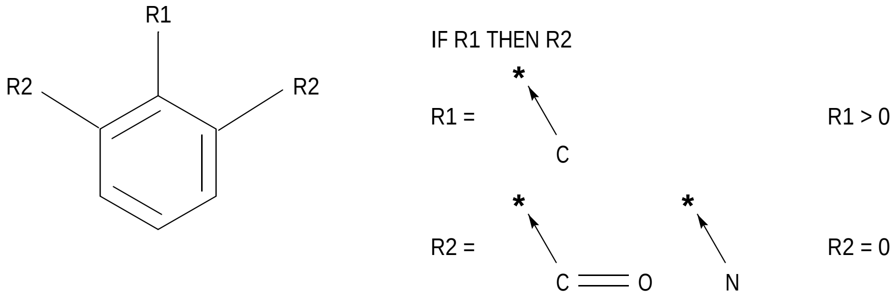
*r (Rgroup) is repeated to a maximum of 32
*m (member) is repeated to a maximum of 255 total atoms and bonds per Rgroup

Figure 4-1 Example of an RGfile (Rgroup query file)



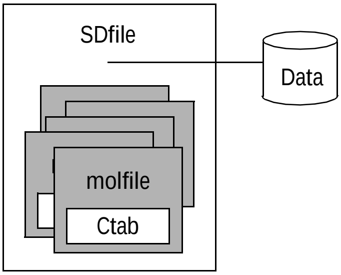
```
$MDL  REV  1 16OCT91  15:40
$MOL
$HDR

GSMACCS-II10169115402D 1   0.00353     0.00000     0

$END HDR
$CTAB
  9  9  0  0  0  0            4 V2000
    1.3337    0.7700    0.0000 C   0  0  0  0  0  0
       .                  .                 .
    1.3337   -0.7700    0.0000 C   0  0  0  0  0  0
    0.0000    3.0800    0.0000 R#  0  0  0  0  0  0
    2.6674    1.5400    0.0000 R#  0  0  0  0  0  0
   -2.6674    1.5400    0.0000 R#  0  0  0  0  0  0
  1  2  1  0  0  0
     .        .
  3  9  1  0  0  0
M  RGP  3   7   1   8   2   9   2
M  LOG  1   1   2   0
M  LOG  1   2   0   0   0
M  END
$END CTAB
$RGP
   1
$CTAB
  1  0  0  0  0  0            2 V2000
   12.2100   14.3903    0.0000 C   0  0  0  0  0  0
M  APO  1   1   1
M  END
$END CTAB
$END RGP
$RGP
   2
$CTAB
  2  1  0  0  0  0            2 V2000
   -1.4969    0.0508    0.0000 C   0  0  0  0  0  0
    0.0431    0.0508    0.0000 O   0  0  0  0  0  0
  1  2  2  0  0  0
M  APO  1   1   1
M  END
$END CTAB
$CTAB
  1  0  0  0  0  0            2 V2000
   12.2100   14.3903    0.0000 N   0  0  0  0  0  0
M  APO  1   1   1
M  END
$END CTAB
$END RGP
$END mol
```

The RGfile shown in Figure 4-1 corresponds to the following Rgroup query:



Chapter 5

SDfiles



An SDfile (structure-data file) contains the structural information and associated data items for one or more compounds. An example of an SDfile is shown in Figure 5-1. The format is:

```
                 [Molfile (see Chapter 3)]
                 [Data Header]
*c      *d    *l [Data]
                 [Blank line]
                 $$$$
```

where:

*l is repeated for each line of data
*d is repeated for each data item
*c is repeated for each compound

A *[Molfile]* block has the molfile format described in Chapter 3 or Chapter 10.

A *[Data Header]* (one line) precedes each item of data, starts with a *greater than* (>) sign, and contains at least one of the following:

- The field name enclosed in angle brackets. For example: `<melting.point>`
- The field number, DT*n* , where *n* represents the number assigned to the field in a MACCS database

Optional information for the data header includes:

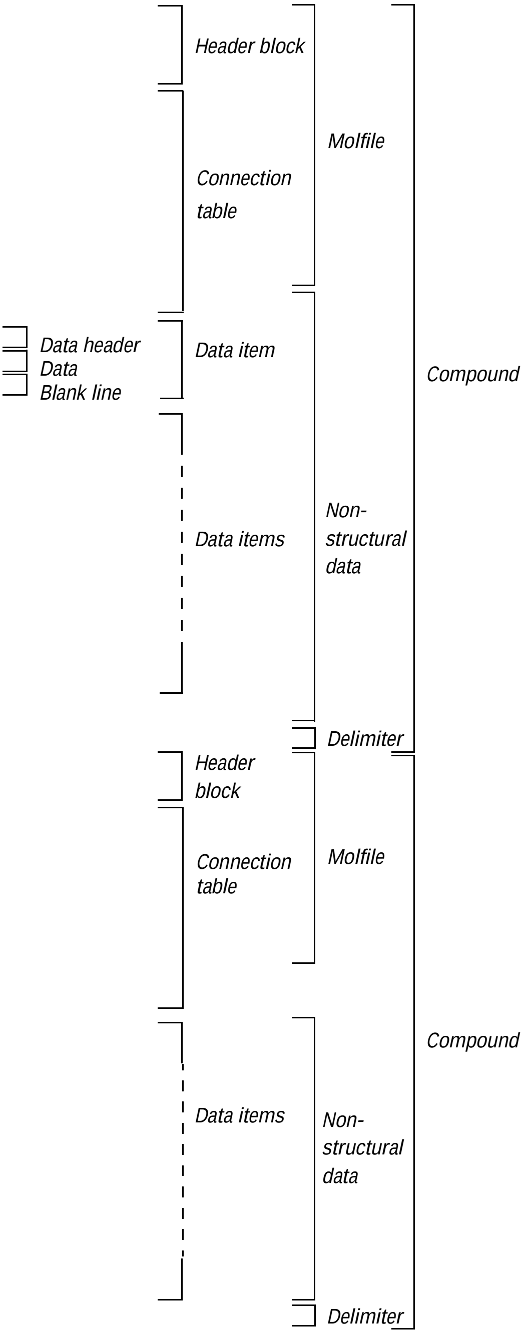
- The compound's external and internal registry numbers. External registry numbers must be enclosed in parentheses.
- Any combination of information

The following are examples of valid data headers:

```
> <MELTING.POINT>
> 55     (MD-08974)     <BOILING.POINT>   DT12
> DT12   55
> (MD-0894)   <BOILING.POINT>   FROM ARCHIVES
```

Figure 5-1 Example of an SDfile

```
1,2 CYCLO-C6 DI-COOH TRANS,L
  MACCS-II06039016292D 1   0.00339     0.00000    25

 12 12  0  0  1  0              1 V2000
   -0.0238   -0.7702    0.0000 C   0  0  1  0  0  0
           .                            .
    2.6974    0.7634    0.0000 O   0  0  0  0  0  0
  1  2  1  0  0  0
      .       .
  7 10  1  0  0  0
M  END
> 25  <MELTING.POINT>
179.0 - 183.0

> 25  <DESCRIPTION>
PW(W)

> 25  <ALTERNATE.NAMES>
1,2 CYCLOHEXANE-DICARBOXYLIC ACID TRANS,L
HEXAHYDROPHTHALIC ACID TRANS,L

> 25  <DATE>
01-10-1980

> 25  <CRC.NUMBER>
C-0710Dat

$$$$
2-METHYL FURAN
  MACCS-II06039016302D 1   0.00186     0.00000    29

  6  6  0  0  0  0              1 V2000
    0.5343    0.3006    0.0000 C   0  0  0  0  0  0
           .                            .
   -2.0038    0.2857    0.0000 C   0  0  0  0  0  0
  1  2  2  0  0  0
      .       .
  5  6  2  0  0  0
M  END
> 29  <DENSITY>
0.9132 - 20.0
> 29  <BOILING.POINT>
63.0 (737 MM)
79.0 (42 MM)
> 29  <ALTERNATE.NAMES>
SYLVAN
> 29  <DATE>
09-23-1980
> 29  <CRC.NUMBER>
F-0213
$$$$
```



A *[Data]* value may extend over multiple lines containing up to 200 characters each. A blank line terminates each data item.

A line containing four dollar signs ($$$$) terminates each complete data block describing a compound.

A datfile (data file) is effectively an SDfile with no *[Molfile]* descriptions or $$$$ delimiters. The *[Data Header]* in a datfile must include either an external or internal registry number in addition to a field name or number.

SDfile after a CFS search

After a conformationally flexible substructure (CFS) search, the following format information is appended by ISIS/Base PL to your SDfile after the connection table:

- Query information (`M $3D` appendix lines added to embedded molfile)
- CFS generated data (*DATA)
- MAPPED ATOMS and BONDS

This information describes, for example, how query atoms are mapped, the atom coordinates in models, and what is fitted during a CFS search.

Figure 5-2 Example of SDfile with appended CFS query information

```
   .
   .
   .
M  CHG  2  14  -1  16   1
M  $3D  5
M  $3D -9  3
M  $3D 13 18    6.3000    8.3000
M  $3D -9  3
M  $3D 18  9    3.1000    5.1000
M  $3D -9  3
M  $3D 18  4    2.4000    4.4000
M  $3D -9  3
M  $3D 13  9    2.8000    4.8000
M  $3D -9  3
M  $3D 13  4    3.1000    5.1000
M  END
> 31 <*DATA>
Method = Derivative

> 31 <MAPPED ATOMS AND BONDS>
(8 13 14 3 9 4 18) (12 13 7 8)

$$$$
```

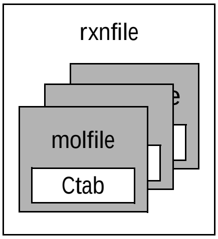
3D Query Fields

CFS-Generated Data

Mapping Atoms and Bonds

Chapter 6

Rxnfiles



Rxnfiles contain structural data for the reactants and products of a reaction. An example rxnfile for a simple reaction is shown in Figure 6-1. The format is:

```
          [Rxnfile Header]
          rrrppp
*r        $MOL
          [Molfile of reactant]
*p        $MOL
          [Molfile of product]
```

where:

*r is repeated for each reactant
*p is repeated for each product

Header Block

Line 1: $RXN in the first position on this line identifies the file as a reaction file.

Line 2: A line which is always blank.

Line 3: The program name and version (P), date/time (M/D/Y,H:m), and reaction registry number (R). This line has the format:

```
            PPPPPPPPMMDDYYHHmmRRRRRRRR
(FORTRAN:       A14 <--A12--->  I8   )
```

A blank line can be substituted for line 3.

Line 4: A line for comments. If no comment is entered, a blank line must be present.

Figure 6-1 Rxnfile for the acylation of benzene

```
$RXN

      REACCS81  1017911041    7439

  2  1
$MOL

  REACCS8110179110412D 1   0.00380     0.00000   315

  4  3  0  0  0  0  0  0  0  0  0
    0.3233   -0.2358    0.0000 C   0  0  0  0  0  0  0  0  0  1  0  0
   -1.0346   -0.9623    0.0000 C   0  0  0  0  0  0  0  0  0  2  0  0
    0.3233    1.4149    0.0000 O   0  0  0  0  0  0  0  0  0  3  0  0
    1.6431   -1.0308    0.0000 Cl  0  0  0  0  0  0  0  0  0  0  0  0
  1  2  1  0  0  0  2
  1  3  2  0  0  0  2
  1  4  1  0  0  0  4
$MOL

  REACCS8110179110412D 1   0.00371     0.00000     8

  6  6  0  0  0  0  0  0  0  0  0
    1.3335   -0.7689    0.0000 C   0  0  0  0  0  0  0  0  0  5  0  0
    1.3335    0.7689    0.0000 C   0  0  0  0  0  0  0  0  0  6  0  0
    0.0000   -1.5415    0.0000 C   0  0  0  0  0  0  0  0  0  7  0  0
    0.0000    1.5415    0.0000 C   0  0  0  0  0  0  0  0  0  8  0  0
   -1.3335   -0.7689    0.0000 C   0  0  0  0  0  0  0  0  0  9  0  0
   -1.3335    0.7689    0.0000 C   0  0  0  0  0  0  0  0  0 10  0  0
  1  2  1  0  0  0  2
  1  3  2  0  0  0  2
  2  4  2  0  0  0  2
  3  5  1  0  0  0  2
  4  6  1  0  0  0  2
  5  6  2  0  0  0  2
$MOL

  REACCS8110179110412D 1   0.00374     0.00000   255

  9  9  0  0  0  0  0  0  0  0  0
   -0.5311   -0.1384    0.0000 C   0  0  0  0  0  0  0  0  0  5  0  0
   -1.8626    0.6321    0.0000 C   0  0  0  0  0  0  0  0  0  6  0  0
   -0.5311   -1.6943    0.0000 C   0  0  0  0  0  0  0  0  0  7  0  0
    0.8191    0.6284    0.0000 C   0  0  0  0  0  0  0  0  0  1  0  0
   -3.2278   -0.1346    0.0000 C   0  0  0  0  0  0  0  0  0  8  0  0
   -1.8813   -2.4723    0.0000 C   0  0  0  0  0  0  0  0  0  9  0  0
    2.1282   -0.1085    0.0000 C   0  0  0  0  0  0  0  0  0  2  0  0
    0.8191    2.2292    0.0000 O   0  0  0  0  0  0  0  0  0  3  0  0
   -3.2278   -1.6831    0.0000 C   0  0  0  0  0  0  0  0  0 10  0  0
  1  2  1  0  0  0  2
       .        .
  6  9  2  0  0  0  2
```

Header block

#Reactants/ #Products

Molfile delimiter

Molfile for first reactant

Molfile delimiter

Molfile for second reactant

Molfile delimiter

Molfile for product

Reactants/Products

A line identifying the number of reactants and products, in that order. The format is:

```
rrrppp
```

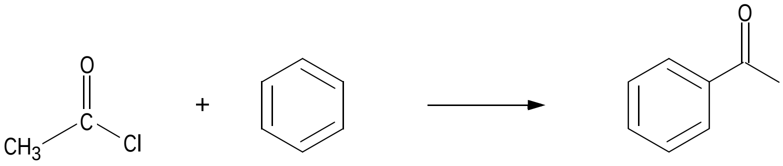
where the variables represent:

`rrr`	Number of reactants
`ppp`	Number of products

Molfile Blocks

A series of blocks, each starting with $MOL as a delimiter, giving the molfile for each reactant and product in turn. The molfile blocks are always in the same order as the molecules in the reaction; reactants first and products second.

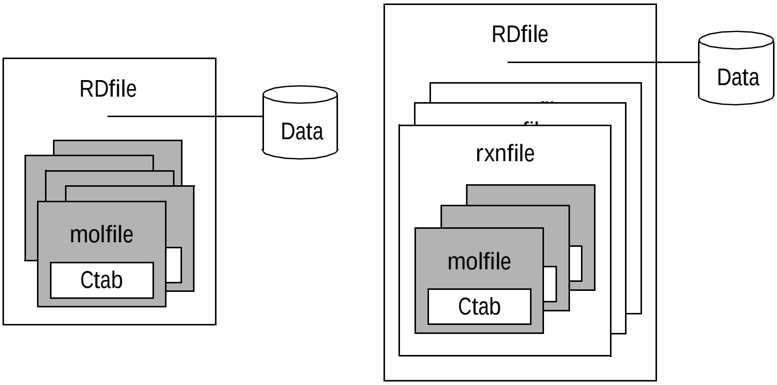
The rxnfile in Figure 6-1 corresponds to the following reaction:



Note: MACCS-II cannot read or write connection tables for reactions.

Chapter 7

RDfiles



An RDfile (reaction-data file) consists of a set of editable “records.” Each record defines a molecule or reaction, and its associated data. An example RDfile incorporating the rxnfile described in Chapter 6 is shown in Figure 7-1. The format for an RDfile is:

```
             [RDfile Header]
             [Molecule or Reaction Identifier]
*r        *d [Data-field Identifier]
             [Data]
```

where:

*d is repeated for each data item
*r is repeated for each reaction or molecule

Each logical line in an RDfile starts with a keyword in column 1 of a physical line. One or more blanks separate the first argument (if any) from the keyword. The blanks are ignored when the line is read. After the first argument, blanks are significant.

An argument longer than 80 characters breaks at column 80 and continues in column 1 of the next line. (The argument may continue on additional lines up to the physical limits on text length imposed by the database.)

The RDfile must not contain any blank lines except as part of embedded molfiles, rxnfiles, or data. An identifier separates records.

RDfile Header

Line 1: $RDFILE 1: The *[RDfile Header]* must occur at the beginning of the physical file and identifies the file as an RDfile. The version stamp "1" is intended for future expansion of the format.

Line 2: $DATM: Date/time (M/D/Y, H:m) stamp. This line is treated as a comment and ignored when the program is read.

Molecule and Reaction Identifiers

A *[Molecule or Reaction Identifier]* defines the start of each complete record in an RDfile. The form of a *molecule* identifier must be one of the following:

```
$MFMT [$MIREG internal-regno [$MEREG external-regno]] embedded molfile
$MIREG internal-regno
$MEREG external-regno
```

where:

- $MFMT defines a molecule by specifying its connection table as a molfile
- $MIREG *internal-regno* is the internal registry number (sequence number in the database) of the molecule
- $MEREG *external-regno* is the external registry number of the molecule (any uniquely identifying character string known to the database, for example, CAS number)
- Square brackets ([]) enclose optional parameters
- An embedded molfile (see Chapter 3) follows immediately after the $MFMT line

The forms of a *reaction* identifier closely parallel that of a molecule:

```
$RFMT [$RIREG internal-regno [$REREG external-regno]] embedded rxnfile
$PCRXN [$RIREG internal-regno [$REREG external-regno]] embedded CPSS rxnfile CP
$RIREG internal-regno
$REREG external-regno
```

where:

- $RFMT defines a reaction by specifying its descripton as a rxnfile and $PCRXN CP defines a reaction by specifying its descripton as a CPSS-style rxnfile
- $RIREG *internal-regno* is the internal registry number (sequence number in the database) of the reaction
- $REREG *external-regno* is the external registry number of the reaction (any uniquely identifying character string known to the database)
- Square brackets ([]) enclose optional parameters
- An embedded rxnfile (see Chapter 6) follows immediately after the $RFMT line, and an embedded CPSS-style rxnfile follows immediately after the $PCRXN CP line

Data-field Identifier

The *[Data-field Identifier]* specifies the name of a data field in the database. The format is:

```
$DTYPE field name
```

Data

Data associated with a field follows the field name on the next line and has the form:

```
$DATUM datum
```

The format of *datum* depends upon the data type of the field as defined in the database. For example: integer, real number, real range, text, molecule regno.

For fields whose data type is “molecule regno,” the *datum* must specify a molecule and, with the exception noted below, use one of the formats defined above for a molecular identifier. For example:

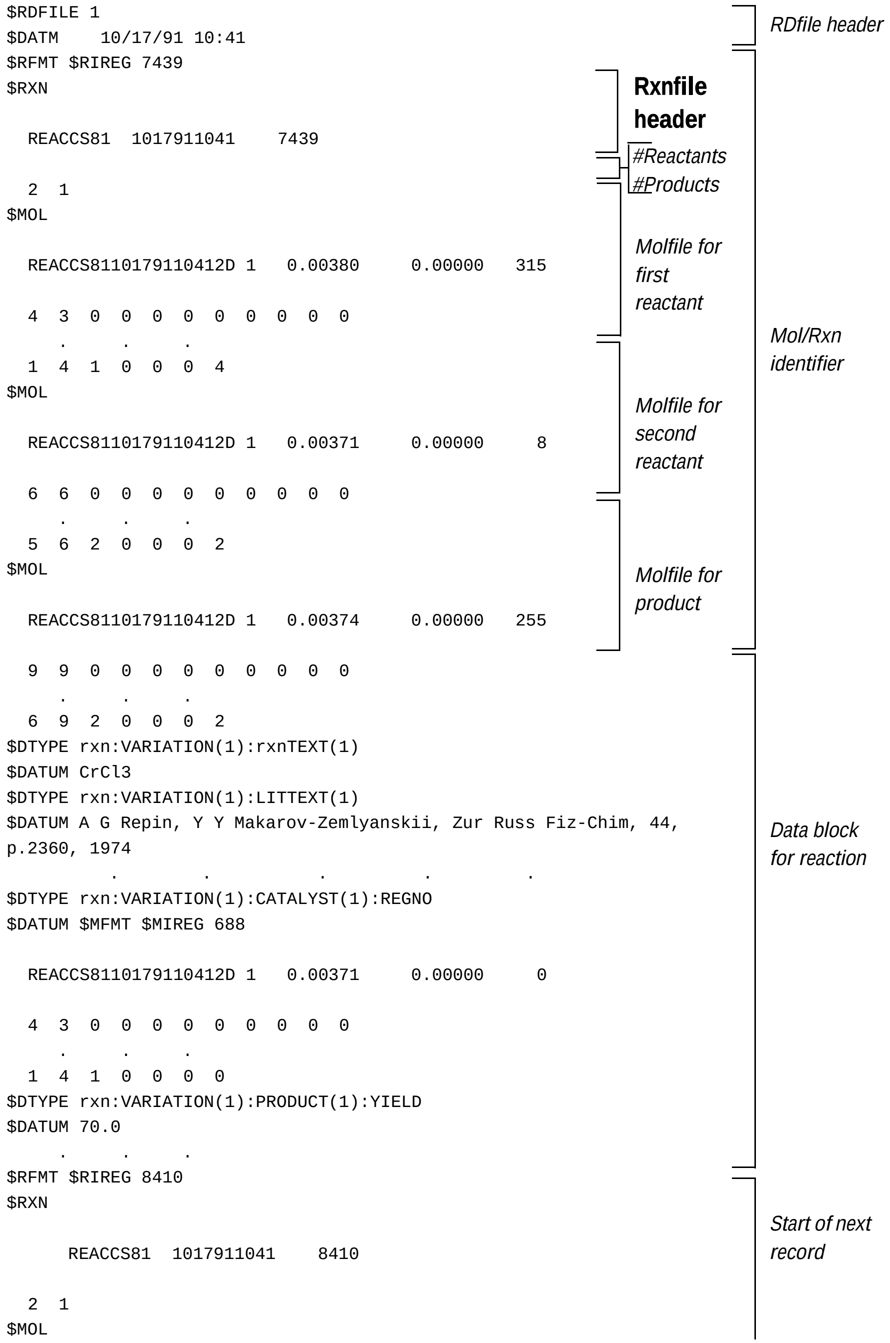
```
$DATUM $MFMT embedded molfile
$DATUM $MEREG external-regno
$DATUM $MIREG internal-regno
```

In addition, the following special format is accepted:

```
$DATUM molecule-identifier
```

Here, *molecule-identifier* acts in the same way as *external-regno* in that it can be any text string known to the database that uniquely identifies a molecule. (It is usually associated with a data field different from the *external-regno*.)

Figure 7-1 Example of a reaction RDfile



```
$RDFILE 1
$DATM    10/17/91 10:41
$RFMT $RIREG 7439
$RXN

  REACCS81  1017911041    7439

  2  1
$MOL

  REACCS8110179110412D 1   0.00380     0.00000   315

  4  3  0  0  0  0  0  0  0  0  0
     .     .     .
  1  4  1  0  0  0  4
$MOL

  REACCS8110179110412D 1   0.00371     0.00000     8

  6  6  0  0  0  0  0  0  0  0  0
     .     .     .
  5  6  2  0  0  0  2
$MOL

  REACCS8110179110412D 1   0.00374     0.00000   255

  9  9  0  0  0  0  0  0  0  0  0
     .     .     .
  6  9  2  0  0  0  2
$DTYPE rxn:VARIATION(1):rxnTEXT(1)
$DATUM CrCl3
$DTYPE rxn:VARIATION(1):LITTEXT(1)
$DATUM A G Repin, Y Y Makarov-Zemlyanskii, Zur Russ Fiz-Chim, 44,
p.2360, 1974
          .        .          .         .         .
$DTYPE rxn:VARIATION(1):CATALYST(1):REGNO
$DATUM $MFMT $MIREG 688

  REACCS8110179110412D 1   0.00371     0.00000     0

  4  3  0  0  0  0  0  0  0  0  0
     .     .     .
  1  4  1  0  0  0  0
$DTYPE rxn:VARIATION(1):PRODUCT(1):YIELD
$DATUM 70.0
     .     .     .
$RFMT $RIREG 8410
$RXN

      REACCS81  1017911041    8410

  2  1
$MOL
```

Atom Limit Enhancements

The formats presented in this chapter were added to support the chemical representation enhancements of ISIS 2.0 Desktop.

Phantom Extra Atom

The format for phantom extra atom information is as follows:

```
M  PXA aaaxxxxx.xxxxyyyyy.yyyyzzzzz.zzzz H   e...
```

where:

aaa	= Index of (real) atom for attachment
xyz	= Coordinates for the added atom
H	= Atom symbol
e...	= Additional text string (for example, the superatom label)

The FORTRAN format for the phantom extra atom entry is as follows:

```
(I4,4F10.4,1X,A3,1X,A)
```

The bond to the added phantom atom is added as a crossing bond to the outermost Sgroup that contains atom aaa. Note this appendix supplies coordinates and up to 35 characters of ‘label’ that can be used for the ISIS/Desktop superatom conversion mechanism. The ISIS/Desktop uses this appendix to register hydrogen-only superatoms, which are often used as superatom leaving groups on the desktop, but which cannot be directly registered into host database. The hydrogen-only leaving groups are converted to PXA appendices for registration, and converted back when ISIS/Desktop reads the structure.

The following are limitations on phantom extra atom:

- Superatom nesting cases
- No bonded phantom atom-phantom atom support

Superatom Attachment Point

The format for superatom attachment point is as follows:

```
M  SAP sssnn6 iii ooo cc
```

where:

`sss` = Index of superatom Sgroup

`nn6` = Number of `iii`,`ooo`,`c` entries on the line (6 maximum)

`iii` = Index of the attachment point atom

`ooo` = Index of atom in `sss` that leaves when `iii` is substituted

`cc` = 2 character attachment identifier (for example, ‘`H`’ or ‘`T`’ for head/tail). No validation of any kind is performed, and ‘ ’ is allowed. ISIS/Desktop uses the first character as the ID of the leaving group to attach if the bond between `ooo` and `iii` is deleted, and uses the second character to indicate the sequence polarity: `l` for left, `r` for right, and `x` for none (a crosslink).

The bond `iii-ooo` is either a sequence bond, a sequence crosslink bond, or a bond to a leaving group that terminates a sequence or caps a crosslink bond. In some cases, this bond may have been deleted by the user, probably to perform a substructure search. In this case, `ooo` will be 0. If the leaving group attached to `iii` consists of only a hydrogen, the leaving group will be replaced by a Phantom Extra Atom, as previously described. In this case, `iii` is set equal to `ooo` as a signal to ISIS/Desktop that a hydrogen-only leaving group must be reattached to `iii`.

The FORTRAN format for the superatom attachment point entry is as follows:

```
(I4,I4,1X,A2)
```

An attachment point entry is one `iii`,`ooo`,`cc` triad.

Multiple `M SAP` lines are permitted for each superatom Sgroup to the maximum of the atom attachment limit. The order of the attachment entries is significant because the first `iii,ooo,c` becomes the first connection made when drawing to the collapsed superatom, and so forth.

Superatom Class

The format for superatom class is as follows:

```
M  SCL sss d...
```

where:

sss	= Index of superatom Sgroup
d...	= Text string (for example, PEPTIDE, ...) 69 characters maximum

This appendix identifies the class of the superatom Sgroup. It distinguishes, for example, peptide groups from nucleotides.

Large REGNO

The format for the regno appendix is as follows:

```
M  REG r...
```

where:

r...	=Free format integer regno

This appendix supports overflow of the I6 regno field in the molfile header. If this appendix is present, the value of the regno in the molfile header is superceded.

Sgroup Bracket Style

The format for the Sgroup bracket style is as follows:

```
M  SBTnn8 sss ttt ...
```

where:

sss	= Index of Sgroup

`ttt` = Bracket display style: 0 = default, 1 = curved (parenthetic) brackets

This appendix supports altering the display style of the Sgroup brackets.

Chapter 9

Moving CTfiles On and Off the Clipboard in ISIS

Clipboard Objects

The two objects named here as SK and mSK are used to move MDL sketches on and off the clipboard in ISIS. The names and contents of these with respect to the PC (MS Windows), Macintosh, and SGI (Motif) platforms are summarized in Table 9-1 and described in the *ISIS Sketch File Formats* document. The additional object, CT, is also introduced. This contains structural information in CTfile format to facilitate structure exchanges between ISIS and non-MDL applications. The object, mSK, is not meaningful to platforms such as SGI, because Motif lacks a metafile format like the Macintosh or MS Windows metafile for storing drawing commands.

Table 9-1 ISIS clipboard objects-names and content

Clipboard Object	MS Windows Clipboard Format	Macintosh Scrap Type	SGI Motif Clip-board Format	Contents	Available in ISIS version
SK	MDLSK	swsD	MDL_SKETCH	Buffered MDL sketch file	1.0 and up
CT	MDLCT	swsC	MDL_MOL	Buffered MDL CTfile (molfile, RGfile or rxnfile)	1.01 and up
mSK	CF_METAFILEPICT	PICT		Picture with MDL sketch embedded	1.0 and up

ISIS will look for the objects listed in Table 9-1 in the order SK, CT, mSK and will take the first available for the image. The metafile, mSK, cannot be distinguished until after it is read from the clipboard, because the embedded file is not identified.

Note: CT has variable length lines. Each line is prefixed with one byte containing the length of the line. Thus, a blank line contains one byte of zero.

Hints on Creating a Reader/Writer For CT

Separate input/output routines from the CTfile interpreter.

Use open/read/close routines to read the contents of the buffer from the clipboard line by line.

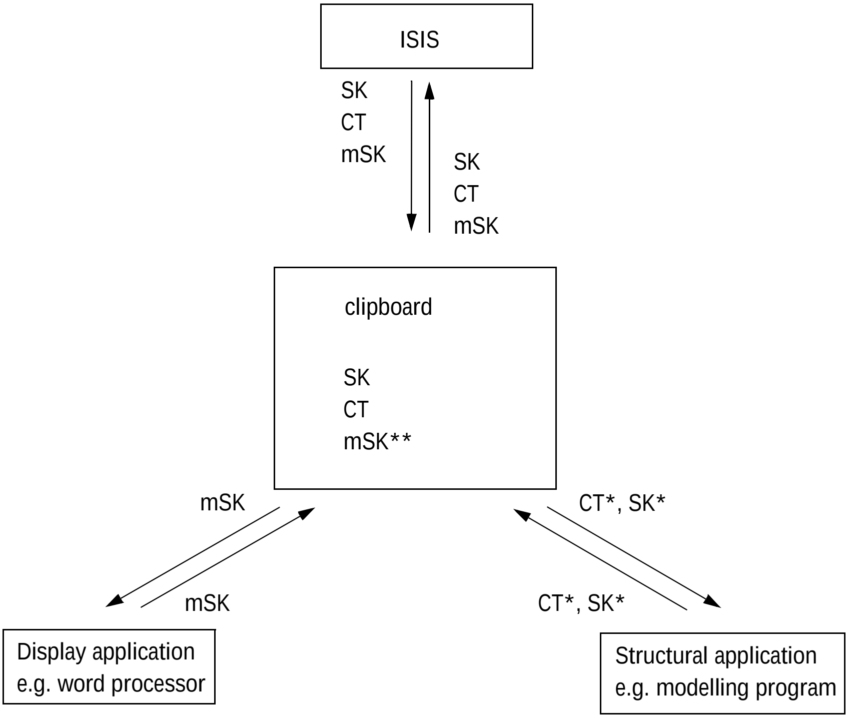
Copying *from* the Clipboard

Look for CT on the clipboard. If present and the first line contains:

- “$RXN”, the file is a rxnfile
- “$MDL”, the file is an RGfile
- Otherwise, the file is a molfile

Alternatively, you can develop your own procedure for reading a sketch file (SK* in Figure 9-1).

Figure 9-1 Transfer options



* Employing user-supplied file reader or writer

** Except SGI

Copying *to* the clipboard

Clear the clipboard of any existing data.

You may choose from among the following options recognizable by ISIS:

- Post a CT containing a buffered CTfile (rxnfile, RGfile or molfile) (with Version 1.01 or later of ISIS).
- Post an SK containing a buffered sketch file.
- Post your own rendering as a metafile or PICT image (PC and Macintosh, respectively) recognizable only by ISIS/Draw. However, this does not preserve the chemistry.

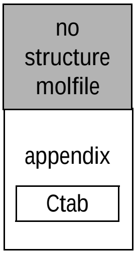
Sample Code For Copying or Pasting a CTfile in MS Windows

```
/* cutpaste.c */
extern HWND hwnd;            /* handle to application’s main window */
static int ctFormat;
/*--------------------------------------------------------------------*/
InitClipBoard()
{
    ctFormat = RegisterClipboardFormat(”MDLCT”);
} /*--------------------------------------------------------------------*/
CopyToClipboard(HANDLE ghCTBuffer)
/*ghCTBuffer is a global handle to a buffer containing the ASCII ctfile. Do not
delete it because it becomes the property of the clipboard after the
SetClipboardData() call. */
{
    if (OpenClipboard(hwnd)) {
       EmptyClipboard();
       SetClipboardData(ctFormat, ghCTBuffer);
       CloseClipboard();
    }
 }
/*--------------------------------------------------------------------*/ HANDLE
PasteFromClipboard()
{
    HANDLE ghCTBuffer = NULL;
   if (IsClipboardFormatAvailable(ctFormat)) {
       if (OpenClipboard(hwnd)) {
          ghCTBuffer = GetClipboardData(ctFormat);
          CloseClipboard();
       }
    }
/*ghCTBuffer is a global handle to a buffer containing the ASCII ctfile. It is
still the property of the clipboard so do not delete or alter it. */
    return(ghCTBuffer);
} /*--------------------------------------------------------------------*/
```

PART II
Extended File Formats

Chapter 10

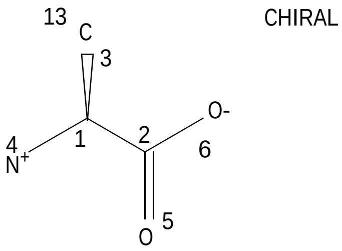
The Extended Molfile Format



The extended (V3000) molfile consists of a regular molfile “no structure” followed by a single molfile appendix that contains the body of the connection table (Ctab). Figure 10-1 shows both an alanine structure and the extended molfile corresponding to it. See Figure 2-1 for the V2000 version of this same structure.

Figure 10-1 Extended molfile organization illustrated using alanine

L-Alanine



```
L-Alanine
GSMACCS-II07129516502D 1   0.00366     0.00000     0
Figure 1, J. Chem. Inf. Comput. Sci., Vol 32, No. 3., 1992
  0  0  0     0  0            999 V3000
M  V30 BEGIN CTAB
M  V30 COUNTS 6 5 0 0 1
M  V30 BEGIN ATOM
M  V30 1 C -0.6622 0.5342 0 0 CFG=2
M  V30 2 C 0.6622 -0.3 0 0
M  V30 3 C -0.7207 2.0817 0 0 MASS=13
M  V30 4 N -1.8622 -0.3695 0 0 CHG=1
M  V30 5 O 0.622 -1.8037 0 0
M  V30 6 O 1.9464 0.4244 0 0 CHG=-1
M  V30 END ATOM
M  V30 BEGIN BOND
M  V30 1 1 1 2
M  V30 2 1 1 3 CFG=1
M  V30 3 1 1 4
M  V30 4 2 2 5
M  V30 5 1 2 6
M  V30 END BOND
M  V30 END CTAB
M  END
```

Header block

Comments line

Counts line

Atom block

Connection table Ctab

Bond block

Sgroup block

Rgroup block

3D block

Blocks not used in this Ctab

Note that the "no structure" is flagged with the "V3000" instead of the "V2000" version stamp.

There are two other changes to the header in addition to the version:

- The number of appendix lines is always written as 999, regardless of how many there actually are. (All current readers will disregard the count and stop at “M END”.)
- The “dimensional code” is maintained more explicitly. Thus “3D” really means 3D, although “2D” will be interpreted as 3D if any non-zero Z coordinates are found.

Unlike the V2000 molfile, the V3000 extended Rgroup molfile has the same header format as a non-Rgroup molfile.

Note: Do not create a molfile with a pre-V3000 Rgroup header (”$MDL”, and so forth) but with V3000 Ctab blocks. This is not allowed. A pre-V2000 Rgroup molfile can only have embedded molfiles that are also pre-V3000 versions, for example, the version is either “V2000” or “ ”.

Specifications For Atom and Bond Descriptions

The general syntax of an entry is:

```
M  V30 key posval posval ... [keyword=value] [keyword=value] ...
```

or

```
M  V30 BEGIN key [blockname]
M  V30 posval posval ... keyword=value keyword=value ...
...
M  V30 END key
```

Each line must begin with “M V30 ” with the two blank spaces after M and one blank space after 30. Following this is a list of zero or more required positional values (posval). Optional values may follow which use a ‘KEYWORD=value’ format. Items are separated by white space. There can also be white space preceding the first item. Trailing white space is ignored.

The value of a keyword can be a list containing two or more values:

```
KEYWORD=(N val1 val2 ... valN)
```

where N specifies the number of items that follow.

Values (`posval`, `value`, or `val1`, and so forth) can be strings. Strings that contain blank spaces or start with left parenthesis or double quote, must be surrounded by double quotes. A double quote may be entered literally by doubling it.

Each entry is one line of no more than 80 characters. To allow continuation when the 80-character line is too short, use a dash (-) as the last character. When read, the line is concatenated with the next line by removing the dash and stripping the initial “M V30 ” from the following line. For example:

```
M  V30 10 20 30 “abc-
M  V30   def”
```

is read as:

```
M  V30 10 20 30 “abc  def”
```

Generally, each section of the molfile is enclosed in a *block* that consists of lines such as:

```
M  V30 BEGIN key [blockname]
...
M  V30 END key
```

The ‘key’ value defines the kind of block, for example, CTAB, ATOM, or BOND. Depending upon the type of block, there may or may not be values on the BEGIN line.

Conventions

The new format conventions used in this chapter are as follows:

UPPERCASE — Literal text, to be entered as shown. Only the position of ”M V30 ” is significant; white space may be added anywhere else to improve readability. Note that *both* lower- and uppercase characters, or any combination of them, are acceptable for literals. They are shown here in uppercase only for readability.

lowercase — A token, which is defined elsewhere.

[ ] — An optional item. Do not include the brackets.

[ ]* — An optional item, where there may be zero, one, two, or more of the item.

| — Separates two or more options, only one of which is valid.

/ — Separates two or more items. Either or both may appear in any order.

{} Braces are used for grouping. They indicate indefinite or definite repeat.

The Extended Connection Table

The features of the extended connection table are described in this section.

CTAB block

A Ctab block defines the basic connection table, which is defined as:

```
M  V30 BEGIN CTAB [ctabname]
counts-line
atom-block
[bond-block]
[sgroup-block]
[3d-block]
[link-line]*
M  V30 END CTAB
```

The atom block, like the counts line, is required. The Sgroup block, 3D block, and link lines may occur in any order after the atom and bond blocks. The counts line, atom block, and bond block must appear in the order indicated.

Counts line

A counts line is required, and must be first. It specifies the number of atoms, bonds, 3D objects, and Sgroups. It also specifies whether or not the CHIRAL flag is set. Optionally, the counts line can specify molregno. This is only used when the regno exceeds 999999 (the limit of the format in the molfile header line). The format of the counts line is:

```
M  V30 COUNTS na nb nsg n3d chiral [REGNO=regno]
```

where:

```
na      = number of atoms
nb      = number of bonds
nsg     = number of Sgroups
n3d     = number of 3D constraints
chiral  = 1 if molecule is chiral, 0 if not
regno   = molecule or model regno
```

Atom block

An atom block specifies all node information for the connection table. It must precede the bond block. It has the following format:

```
M  V30 BEGIN ATOM
M  V30 index type x y z aamap -
M  V30 [CHG=val] [RAD=val] [CFG=val] [MASS=val] -
M  V30 [VAL=val] -
M  V30 [HCOUNT=val] [STBOX=val] [INVRET=val] [EXACHG=val] -
M  V30 [SUBST=val] [UNSAT=val] [RBCNT=val] -
M  V30 [ATTCHPT=val] -
M  V30 [RGROUPS=(nvals val [val ...])] -
M  V30 [ATTCHORD=(nvals nbr1 val1 [nbr2 val2 ...])] -
...
M  V30 END ATOM
```

The values are described in Table 10-1.

Table 10-1 Meaning of values in the atom block

Field	Meaning	Values	Notes
`index`	Atom index	Integer > 0	Identifies atoms. The actual value of the index does not matter as long as each index is unique to each atom. However, extremely large numbers used as indexes can cause the program to fail to allocate memory for the correspondence array.
`type`	Atom type	Type = reserved atom *or* atom *or* [NOT] '['atom, atom,...']'	A character string. If the string contains white space, it must be quoted. It can be a single atom or an atom list enclosed in square brackets with an optional preceding NOT.
		where reserved atom =	
		R# = Rgroup	
		A = "any" atom	
		Q = any atom but C or H	
		* = "star" atom	
		Atom = character string	For example, 'C' or 'Cl'
`x y z`	Atom coordinates	Angstroms	

Table 10-1 Meaning of values in the atom block (Continued)

Field	Meaning	Values	Notes
`aamap`	Atom-atom mapping	0 = no mapping	Reaction property
		> 0 = mapped atom	
`CHG`	Atom charge	Integer	Same range as V2000.
		0 = none (default)	
`RAD`	Atom radical	0 = none (default)	
		1 = singlet	
		2 = doublet	
		3 = triplet	
`CFG`	Stereo configuration	0 = none (default)	
		1 = odd parity	
		2 = even parity	
		3 = either parity	
`MASS`	Atomic weight	Integer > 0	Default = natural abundance
`VAL`	Valence	Integer > 0 *or*	Abnormal valence
		0 = none (default)	
		-1 = zero	
`HCOUNT`	Query hydrogen count	Integer > 0 *or*	Same maximum as V2000.
		0 = not specified (default)	
		-1 = zero	
`STBOX`	Stero box	0 = ignore the configuration of this double bond atom (default)	Both atoms of a double bond must be marked to search double bond stereochemistry
		1 = consider the stereo configuration of this double bond atom	
`INVRET`	Configuration inversion	0 = none (default)	Reaction property
		1 = configuration inverts	
		2 = configuration retained	
`EXACHG`	Exact change	0 = property not applied (default)	Reaction property
		1 = exact change as displayed in the reaction	

Table 10-1 Meaning of values in the atom block (Continued)

Field	Meaning	Values	Notes
`SUBST`	Query substitution count	Integer > 0 *or*	Same maximum as V2000.
		0 = not specified (default)	
		-1 = none	
`UNSAT`	Query unsaturation flag	0 = not specified (default)	
		1 = unsaturated	
`RBCNT`	Query ring bond count	Integer > 0 *or*	Same maximum as V2000.
		0 = not specified (default)	
		-1 = none	
`ATTCHPT`	Rgroup member attachment points	Attachment points on member:	When the Rgroup member atom has two attachment points, the atom with the lowest index number attaches to the first attachment point
		-1 = first and second site	
		1 = first site only	
		2 = second site only	
`RGROUPS`	`nvals` is the number of Rgroups that comprise this R# atom.	Integer > o	
	`val` is the Rgroup number.		
`ATTCHORD`	`nvals` is the number of values that follow on the `ATTCHORD` line	Integer > o	A list of atom neighbor index and atom neighbor value pairs that identify the attachment order information at the R# atom
	`nbr1` is atom neighbor index #1, `nbr2` is index #2...		
	`val1` is the attachment order for the `nbr1` attachment...		

Bond block

A bond block specifies all edge information for the connection table. It must precede the Sgroup or 3D blocks. Its format is:

```
M  V30 BEGIN BOND
M  V30 index type atom1 atom2 [CFG=val] [TOPO=val] [RXCTR=val] [STBOX=val]
...
M  V30 END BOND
```

where the values are described in Table 10-2.

Table 10-2 Meaning of values in the bond block

Field	Meaning	Values	Notes
`index`	Bond index	Integer > 0	The actual value of the index does not matter as long as all are unique. However, extremely large numbers used as indexes can cause the program to fail to allocate memory for the correspondence array.
`type`	Bond type	Integer:	Types 4 through 8 are for queries only.
		1 = single	
		2 = double	
		3 = triple	
		4 = aromatic	
		5 = single or double	
		6 = single or aromatic	
		7 = double or aromatic	
		8 = any	
`atom1,atom2`	Atom indexes	Integer > 0	Atom1 and Atom2 are bond end points.
`CFG`	Bond configuration	0 = none (default)	
		1 = up	
		2 = either	
		3 = down	
`TOPO`	Query property	0 = not specified (default)	
		1 = ring	

Table 10-2 Meaning of values in the bond block (Continued)

Field	Meaning	Values	Notes
		2 = chain	
RXCTR	Reacting center status	0 = unmarked (default) -1 = not a reacting center 1 = generic reacting center Additional: 2 = no change 4 = bond made or broken 8 = bond order changes 12 =(4 + 8) bond made or broken and changes 5 = (4 + 1), 9 = (8 + 1), and 13 =(12 + 1) are also possible	
STBOX	Stereo box	0 = ignore the configuration of this double bond (default) 1 = consider the stereo configuration of this double bond	A double bond must be marked to search double bond stereochemistry

Link atom line

There is one link atom line for each link atom in the Ctab. A link atom line has the format:

```
M  V30 LINKNODE minrep maxrep nbonds inatom outatom [inatom outatom...]
```

Table 10-3 Meaning of values in link lines

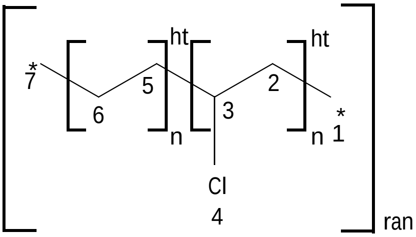
Field	Meaning	Values	Notes
`minrep`	Minimum number of group repetitions.	1	For future expansion. Not currently used.
`maxrep`	Maximum number of group repetitions.	Integer > 0	
`nbonds`	Number of directed bonds defining the group.	nbonds = # of pairs of inatom-outatom tuples	Number of tuples is usually two but may be one for link nodes with an attachment point.
`inatom`	Atom index of atom in the repeating group.	Integer > 0	
`outatom`	Atom index of atom bonded to `inatom`, but outside of repeating group.	Integer > 0	

Sgroup block

The Sgroup block contains general Sgroup information and information on each Sgroup structure as shown in Figure 10-2. For the V2000 version of this Sgroup structure and connection table, see Figure 2-2.

Figure 10-2 Connection table organization of an Sgroup structure

Polymer



```
Polymer
GSMACCS-II07129516502D 1   0.00374     0.00000     0
Figure 5, J. Chem. Inf. Comput. Sci., Vol 32, No. 3., 1992
  0  0  0     0  0            999 V3000
M  V30 BEGIN CTAB
M  V30 COUNTS 7 6 3 0 0
M  V30 BEGIN ATOM
M  V30 1 * 2.9463 0.3489 0 0
M  V30 2 C 1.6126 1.1189 0 0
M  V30 3 C 0.2789 0.3489 0 0 CFG=3
M  V30 4 Cl 0.2789 -1.1911 0 0
M  V30 5 C -1.0548 1.119 0 0
M  V30 6 C -2.3885 0.349 0 0
M  V30 7 * -3.9246 1.147 0 0
M  V30 END ATOM
M  V30 BEGIN BOND
M  V30 1 1 1 2
M  V30 2 1 2 3
M  V30 3 1 3 4
M  V30 4 1 5 6
M  V30 5 1 5 3
M  V30 6 1 7 6
M  V30 END BOND
M  V30 BEGIN SGROUP
M  V30 1 SRU 5 ATOMS=(2 5 6) XBONDS=(2 5 6) BRKXYZ=(9 -0.6103 1.2969 0 -0.6103 -
M  V30 0.171 0 0 0 0) BRKXYZ=(9 -3.1565 0.185 0 -3.1565 1.311 0 0 0 0) -
M  V30 CONNECT=HT
M  V30 2 SRU 6 ATOMS=(3 2 3 4) XBONDS=(2 1 5) BRKXYZ=(9 2.2794 1.2969 0 2.2794 -
M  V30 0.1709 0 0 0 0) BRKXYZ=(9 -0.1657 0.171 0 -0.1657 1.2969 0 0 0 0) -
M  V30 CONNECT=HT
M  V30 3 COP 7 ATOMS=(7 1 2 3 4 5 6 7) BRKXYZ=(9 3.6382 1.6391 0 3.6382 -
M  V30 -1.7685 0 0 0 0) BRKXYZ=(9 -4.707 -1.7685 0 -4.707 1.6391 0 0 0 0) -
M  V30 SUBTYPE=RAN
M  V30 END SGROUP
M  V30 END CTAB
M  END
```

Header block

Comments line

Counts line

Atom block

Ctab block

Bond block

Rgroup block

Blocks not used in this Ctab

3D block

Sgroup 1 Sgroup 2 Sgroup 3

Sgroup Properties

An Sgroup block defines all Sgroups in the molecule, including superatoms. The format is as follows:

```
M  V30 BEGIN SGROUP
[M  V30 DEFAULT [CLASS=class] -]
M  V30 index type extindex -
M  V30 [ATOMS=(natoms atom [atom ...])] -
M  V30 [XBONDS=(nxbonds xbond [xbond ...])] -
M  V30 [CBONDS=(ncbonds cbond [cbond ...])] -
M  V30 [PATOMS=(npatoms patom [patom ...])] -
M  V30 [SUBTYPE=subtype] [MULT=mult] -
M  V30 [CONNECT=connect] [PARENT=parent] [COMPNO=compno] -
M  V30 [XBHEAD=(nxbonds xbond [xbond ...])] -
M  V30 [XBCORR=(nxbpairs xb1 xb2 [xb1 xb2 ...])] -
M  V30 [LABEL=label] -
M  V30 [BRKXYZ=(9 bx1 by1 bz1 bx2 by2 bz2 bx3 by3 bz3])* -
M  V30 [ESTATE=estate] [CSTATE=(4 xbond cbvx cbvy cbvz)]* -
M  V30 [FIELDNAME=fieldname] [FIELDINFO=fieldinfo] -
M  V30 [FIELDDISP=fielddisp] -
M  V30 [QUERYTYPE=querytype] [QUERYOP=queryop] -
M  V30 [FIELDDATA=fielddata] ... -
M  V30 [CLASS=class] -
M  V30 [SAP=(3 aidx lvidx id)]* -
M  V30 [BRKTYP=bracketType] -
...
M  V30 END SGROUP
```

The DEFAULT field provides a way to specify default values for keyword options. The same keyword options and values as defined in Table 10-4.

Table 10-4 Meaning of values in the Sgroup block

Field	Meaning	Values	Notes
`index`	Sgroup index	integer > 0	The actual value of the index does not matter as long as all indexes are unique. However, extremely large numbers used as indexes can cause the program to fail to allocate memory for the correspondence array.
`type`	Sgroup type	String. Only first 3 letters are significant: SUPeratom MULtiple	

Table 10-4 Meaning of values in the Sgroup block (Continued)

Field	Meaning	Values	Notes
		SRU	
		MONomer	
		COPolymer	
		CROsslink	
		MODification	
		GRAft	
		COMponent	
		MIXture	
		FORmulation	
		DATa	
		ANY	
		GENeric	
`extindex`	External index value	Integer => 0:	Use 0 to autogenerate a number.
		If 0, positive integer assigned	This is the V2000 Sgroup label.
`ATOMS`	`natoms` is the number of atoms that define the Sgroup.	Integer > 0	
	`atom` is the atom index.	Integer > 0	
`XBONDS`	`nxbonds` is the number of crossing bonds.	Integer > 0	
	`xbond` is the crossing-bond index.	Integer > 0	
`CBONDS`	`ncbonds` is the number of containment bonds.	Integer > 0	Only used for Data Sgroups.
	`cbond` is the containment-bond index.	Integer > 0	

Table 10-4 Meaning of values in the Sgroup block (Continued)

Field	Meaning	Values	Notes
PATOMS	npatom is the number of paradigmatic repeating unit atoms. patom is the atom index of an atom in the paradigmatic repeating unit for a multiple group.	Integer > 0	This field is expected to become obsolete and is retained for compatibility with MACCS-II. The field is only used for multiple groups.
SUBTYPE	subtype is the Sgroup subtype.	String. Only the first 3 letters are significant: ALTernate RANdom BLOck	
MULT	mult is the multiple group multiplier.	Integer > 0	
CONNECT	connect is the connectivity.	String values are as follows: EU (default) HH HT	The default, if missing, is EU.The MDL V2000 writer never writes an EU entry.
PARENT	parent is the parent Sgroup index.	Integer > 0	
COMPNO	compno is the component order number.	Integer > 0	
XBHEAD	nxbonds is the number of crossing bonds that cross the “head” bracket.	Integer > 0	
	xbond is the crossing-bond index.	Integer > 0	If XBHEAD is missing, no bonds are paired as the head or tail of the repeating unit.

Table 10-4 Meaning of values in the Sgroup block (Continued)

Field	Meaning	Values	Notes
`XBCORR`	`nxbpairs`	2 x the number of pairs of crossing-bond correspondence, that is, the number of values in list.	
	`xb1 - xb2` is the pairs of crossing-bond correspondence, that is, `xb1` connects to `xb2`.	Integer > 0	
`LABEL`	`label` is the display label for this Sgroup.	String	For example, superatom name
`BRKXYZ`	`bx1 - bz3` are the double (X,Y,Z) display coordinates in each bracket.	Angstroms	By specifying 3 triples, the format allows a 3D display. However, only the first two (X, Y) coordinates are currently used. The Z value and last (X, Y) coordinates are currently ignored and should be set to zero.
`ESTATE`	`estate` is the expanded display state information for superatoms.	String E = expanded superatom or multiple group	Only superatoms and multiple groups (shortcuts) in an expanded internal state are supported. This field defines whether a superatom or multiple group is displayed as expanded or contracted. This field is expected to become obsolete.
`CSTATE`	`xbond` is the crossing bond of the expanded superatom.	Integer > 0	Display vector information for the contracted superatom.
	`cbvx - cvbz` is the vector to contracted superatom.	Angstroms	Only present for expanded superatoms. One CSTATE entry per crossing bond.
`FIELDNAME`	`fieldname` is the name of data field for Data Sgroup.	String	

Table 10-4 Meaning of values in the Sgroup block (Continued)

Field	Meaning	Values	Notes
`FIELDINFO`	`fieldinfo` is the program-specific field information.	Free-format string	Example: In MACCS-II this is: "<type> <units/format>"
`FIELDDISP`	`fielddisp` is the Data Sgroup field display information.	Free-format string	This string is interpreted by V3000 as identical to V2000 appendix for Data Sgroup display ('M SDD') except for the index value.
`QUERYTYPE`	`querytype` is the type of query or no query if missing.	String	
		' ' = not a query (default) 'MQ' = MACCS-II query 'IQ' = ISIS query '<p>Q' = <program> query	
`QUERYOP`	`queryop` is the query operator.	String.	Example: "=" or "LIKE" in ISIS
		ISIS: query operator MACCS-II: blank or missing	
`FIELDDATA`	`fielddata` is the query or field data.	Free-format string	Only one entry per query, but can be more than one for actual data. The order of the entries is important.
`CLASS`	`class` is the character string for superatom class.	String	Example: PEPTIDE
`SAP`	`aidx` is the index of attachment point or potential attachment point atom.	Integer > 0	
	`lvidx` is the index of leaving atom.	Allowed integers are:	

Table 10-4 Meaning of values in the Sgroup block (Continued)

Field	Meaning	Values	Notes
		0 = none or implied H 'aidx' = atom index number of attachment point atom # = atom index number of atom bonded to 'aidx'	
	`id` is the attachment identifier.	String (two chars in V2000)	There must be multiple entries if superatom has more than one attachment point. The order of the entries defines the order of the attachment points. Note that SAP entries may or may not include the actual attachment points, depending on the particular superatom and its representation on the ISIS/Desktop.
`BRKTYP`	`bracketType` is the displayed bracket style.	Allowed values for this string are: BRACKET (default) PAREN	This information supports Sgroup enhancements on the ISIS/Desktop.

Correspondence with existing V2000 appendices:

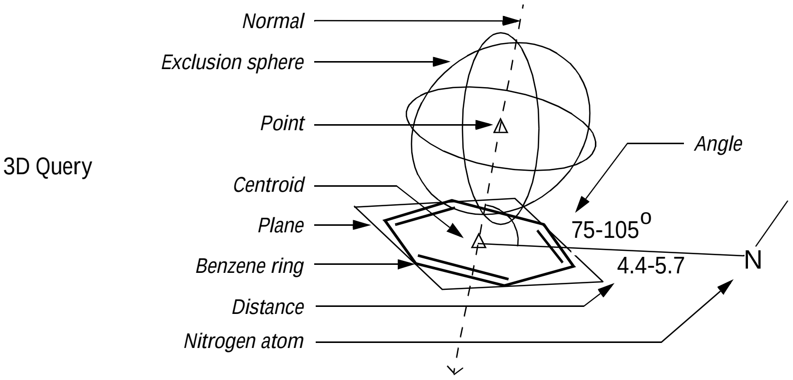
```
M  STY = type
M  SST = SUBTYPE
M  SLB = extindex
M  SCN = CONNECT
M  SDS = ESTATE
M  SAL = ATOMS
M  SBL = XBONDS or CBONDS
M  SPA = PATOMS
M  SMT = LABEL and MULT
M  CRS = XBHEAD,XBCORR
M  SDI = BRKXYZ
M  SBV = CSTATE
M  SDT = FIELDNAME, FIELDINFO, QUERYTYPE, QUERYOP
M  SDD = FIELDDISP
M  SCD = (not required)
M  SED = FIELDDATA
M  SPL = PARENT
```

```
M  SNC = COMPNO
M  SAP = SAP
M  SCL = CLASS
M  SBT = BRKTYP
```

3D block

The 3D block contains 3D information as shown in Figure 10-3. For the V2000 version of this 3D query and its connection table, see Figure 2-3.

Figure 10-3 Connection table organization of a 3D query



```
3D Query
  MACCS-II07129516503D 1   1.00000     0.00000     0
Figure 6, J. Chem. Inf. Comput. Sci., Vol 32, No. 3., 1992
  0  0  0     0  0            999 V3000
M  V30 BEGIN CTAB
M  V30 COUNTS 8 7 0 7 0
M  V30 BEGIN ATOM
M  V30 1 C 1.0252 0.2892 1.1122 0
M  V30 2 C -0.4562 0.6578 1.3156 0
M  V30 3 C -1.4813 0.3687 0.2033 0
M  V30 4 C -1.0252 -0.2892 -1.1122 0
M  V30 5 C 0.4562 -0.6578 -1.3156 0
M  V30 6 C 1.4813 -0.3687 -0.2033 0
M  V30 7 N 4.1401 -0.1989 1.3456 0
M  V30 8 C 4.6453 0.5081 1.7417 0
M  V30 END ATOM
M  V30 BEGIN BOND
M  V30 1 1 1 2
M  V30 2 2 2 3
M  V30 3 1 3 4
M  V30 4 2 4 5
M  V30 5 1 5 6
M  V30 6 2 6 1
M  V30 7 1 7 8
M  V30 END BOND
M  V30 BEGIN OBJ3D
M  V30 1 -7 6 ″″ 0 0 BASIS=(3 6 4 2)
M  V30 2 -5 13 ″″ 0 0 BASIS=(6 1 2 3 4 5 6)
M  V30 3 -8 7 ″″ 0 0 BASIS=(2 O3D.1 O3D.2)
M  V30 4 -3 6 ″″ -2 0 BASIS=(2 O3D.1 O3D.3) PNTDIR=1
M  V30 5 -16 12 ″″ 1.5 0 BASIS=(1 O3D.4) UNCONNOK=1
M  V30 6 -12 10 ″″ 75 105 BASIS=(3 O3D.4 O3D.1 7)
M  V30 7 -9 3 ″″ 4.4 5.7 BASIS=(2 7 O3D.1)
M  V30 END OBJ3D
M  V30 END CTAB
M  END
```

Header block

Comments line

Counts line

Atom block

Bond block

Blocks not used in this Ctab

Sgroup block

Rgroup block

3D objects block

Ctab block

A 3D block specifies information for all 3D objects in the connection table. It must follow the atom and bond blocks. As in V2000 molfiles, there can be only one fixed-atom constraint.

The format of the 3D block is as follows:

```
M  V30 BEGIN OBJ3D
M  V30 index typ color name value1 value2 -
M  V30 BASIS=(nbvals bval [bval ...]) -
M  V30 [ALLOW=(nvals val [val ...])] [PNTDIR=val] [ANGDIR=val] -
M  V30 [UNCONNOK=val] [DATA=strval] -
M  V30 [COMMENT=comment]
...
M  V30 END OBJ3D
```

Table 10-5 Meaning of values in the 3D block

Field	Meaning	Values	Notes
`index`	3D object index	Integer > 0	The actual value of the index does not matter as long as all indexes are unique. However, extremely large numbers used as indexes can cause the program to fail to allocate memory for the correspondence array.
`typ`	Object type	Integer < 0 for geometric constraints for data constraints Integer > 0 are field IDs	This format is the same as V2000.
`color`	Color value	Integer > 0	
`name`	Object name or, for data query, the field name.	String	
`value1`	Distance, radius, deviation, or minimum value.	Floating point, value1 = 0 if constraint has no floating values	
`value2`	Maximum value for range constraints.	Floating point, value2 = 0 if not a range constraint	
`BASIS`	`nbvals` is the number of objects in basis.	Integer > 0	

Table 10-5 Meaning of values in the 3D block (Continued)

Field	Meaning	Values	Notes
	`bval` is the atom number or 3D object index.	Integer *or* O3D.integer	For objects where order is important, for example, in an angle constructed from three points, the order must be the same as in V2000 molfiles.
`ALLOW`	`nvals` is the number of atoms allowed in an exclusion sphere.	Integer > 0	
	`val` is the atom number.	Integer > 0	
`PNTDIR`		0 = point has no direction	
		1 = point has direction	
`ANGDIR`		0 = dihedral angle has no direction	MACCS-II uses ‘Chiral’.
		1 = dihedral angle has direction	
`UNCONNOK`		0 = unconnected atoms are not OK	
		1 = unconnected atoms are OK	
`DATA`	`strval` is the data query string	String	
`COMMENT`	string comment	String. Normally uses the MACCS-II DASP, DISP, and BOX values	Same as V2000 molfile

The Extended Rgroup Query Molfile

A single molecule or Rgroup molecule connection table. The header is contained in the normal header location, that is, in the first three lines of the file. The body of the new molecule is contained in new appendixes, organized as follows:

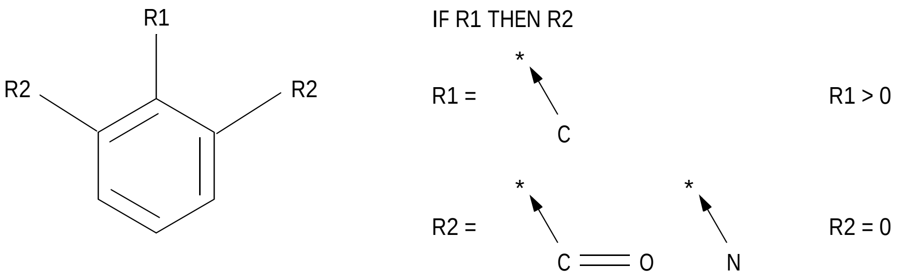
A molecule block consists of a main Ctab, plus optionally one or more Rgroup definitions.

```
ctab-block
[rgroup-block]*
```

Rgroup block

The Rgroup file shown in Figure 10-4 corresponds to the following Rgroup query. For the V2000 version of the Rgroup query and its connection table, see Figure 4-1.

Figure 10-4 Connection table organization of an Rgroup query (Continued on next page)



```
GSMACCS-II07139508292D 1   0.00353     0.00000     0

  0  0  0     0  0            999 V3000                                         Header block
M  V30 BEGIN CTAB
M  V30 COUNTS 9 9 0 0 0                                                          Counts line
M  V30 BEGIN ATOM
M  V30 1 C 1.3337 0.77 0 0
M  V30 2 C 0 1.54 0 0
M  V30 3 C -1.3337 0.77 0 0                                        Atom
M  V30 4 C -1.3337 -0.77 0 0                                       block
M  V30 5 C 0 -1.54 0 0                                             of root
M  V30 6 C 1.3337 -0.77 0 0
M  V30 7 R# 0 3.08 0 0 RGROUPS=(1 1)
M  V30 8 R# 2.6674 1.54 0 0 RGROUPS=(1 2)                                        Ctab for
M  V30 9 R# -2.6674 1.54 0 0 RGROUPS=(1 2)                                       Rgroup
M  V30 END ATOM                                                                  root
M  V30 BEGIN BOND
M  V30 1 1 1 2
M  V30 2 2 2 3
M  V30 3 1 3 4
M  V30 4 2 4 5                                                     Bond
M  V30 5 1 5 6                                                     block
M  V30 6 2 6 1                                                     of root
M  V30 7 1 1 8
M  V30 8 1 2 7
M  V30 9 1 3 9
M  V30 END BOND
M  V30 END CTAB
M  V30 BEGIN RGROUP 1
M  V30 RLOGIC 2 0 ""
M  V30 BEGIN CTAB
M  V30 COUNTS 1 0 0 0 0                                Counts line  Ctab for     Block for
M  V30 BEGIN ATOM                                                   Rgroup 1     Rgroup R1
M  V30 1 C 12.21 14.3903 0 0 ATTCHPT=1                 Atom block   Member 1
M  V30 END ATOM
M  V30 END CTAB
M  V30 END RGROUP
M  V30 BEGIN RGROUP 2
M  V30 RLOGIC 0 0 0
M  V30 BEGIN CTAB
M  V30 COUNTS 2 1 0 0 0                                Counts line  Ctab for
M  V30 BEGIN ATOM                                                   Rgroup 2
M  V30 1 C -1.4969 0.0508 0 0 ATTCHPT=1                             Member 1
M  V30 2 O 0.0431 0.0508 0 0                           Atom block
M  V30 END ATOM
M  V30 BEGIN BOND                                                                Block for
M  V30 1 2 1 2                                         Bond block                Rgroup R2
M  V30 END BOND
M  V30 END CTAB
M  V30 BEGIN CTAB
M  V30 COUNTS 1 0 0 0 0                                Counts line  Ctab for
M  V30 BEGIN ATOM                                                   Rgroup 2
M  V30 1 N 12.21 14.3903 0 0 ATTCHPT=1                 Atom block   Member 2
M  V30 END ATOM
M  V30 END CTAB
M  V30 END RGROUP
M  END
```

An Rgroup block defines one Rgroup. Each Ctab block specifies one member.

```
M  V30 BEGIN RGROUP rgroup-number
[rgroup-logic-line]
ctab-block
[ctab-block]*
M  V30 END RGROUP
```

Table 10-6 Meaning of values in the Rgroup block

Field	Meaning	Values	Notes
`rgroup-number`	Index of this rgroup	Integer > 0	

Rgroup logic lines

There is zero or one Rgroup logic line for each Rgroup in the molecule. If present, the Rgroup logic line specifies if-then logic between Rgroups, the convention about unfilled valence sites, and the Rgroup occurrence information. Its format is:

```
M  V30 RLOGIC thenR RestH Occur
```

Table 10-7 Meaning of values in Rgroup logic line

Field	Meaning	Values	Notes
`thenR`	Number of a “then” Rgroup	0 = none (default)	
`RestH`	Attachment(s) at Rgroup position	0 = off, that is, any molecule fragment at any unsatisfied Rgroup location (default) 1 = only hydrogen or a member of Rgroup is allowed	
`Occur`	String specifying number (range) of Rgroup occurrence sites that need to be satisfied.	String ‘> 0’ = default	Similar to MACCS-II and ISIS: [N[,[N[,...]]]]

DWG file format

An attempt to specify the DWG (R12) file format using the BFF grammar for binary files.

Acknowledgements

I would like to thank Reini Urban <rurban@sbox.tu-graz.ac.at> for his contributions.

Definition of the elementary elements

```
typedef word word :=
     byte : b1, byte : b2
     return (word)f | ((word)s << 8).
typedef longword longp :=
     byte : b1, byte : b2, byte : b3, byte : b4
     return (longword)b1 | ((longword)b2 << 8)
          | ((longword)b3 << 16) | ((longword)b4 << 24).
typedef longword longword :=
     byte : b1, byte : b2, byte : b3, byte : b4
     return (longword)b1 | ((longword)b2 << 8)
          | ((longword)b3 << 16) | ((longword)b4 << 24).
```

Definition of the whole file

```
root dwg_file :=
 [begin : end](
  char[12] : version,
  byte, word, word, word, byte,
  longp : p_entities,  longp : p_entend,
  longp : p_blocksec,  byte[4],  longp : p_bsend,  byte[4],
  tablepos : block_table,
  tablepos : layer_table,
  tablepos : style_table,
  tablepos : ltype_table,
  tablepos : view_table,
  header,  [cur : 0x3EF]byte*,
  tablepos : ucs_table,  [cur : 0x500]byte*,
  tablepos : vport_table,  byte[8],
  tablepos : appid_table,  byte[6],
  tablepos : dimstyle_table,  [cur : 0x69F]byte*,
  tablepos : p13_table,  bytes[38],
  [p_entities : p_entend]entities : ents,   byte[19],
  [block_table.start : ]blocks : block_table,
  [layer_table.start : ]layers : layer_table,
  [style_table.start : ]styles : style_table,
  [ltype_table.start : ]ltypes : ltype_table,
  [view_table.start : ]table : view_table,
  [ucs_table.start : ]table : ucs_table,
  [vport_table.start : vport_table.end]table : vport_table,
  [appid_table.start : ]appids : appid_table,
  [dimstyle_table.start : ]table : dimstyle_table,
  [p13_table.start : ]table : p13_table,
  [p_blocksec : p_bsend]entities : blocks,  bytes[36],
  longp = p_entities,  longp = p_entend,
  longp = blocksec,  longp = bsend,
  bytes[12],
  bytes[6],
  longp = block_table.start,   bytes[6],
  longp = layer_table.start,   bytes[6],
  longp = style_table.start,   bytes[6],
  longp = ltype_table.start,   bytes[6]
  longp = view_table.start,   bytes[6],
  longp = ucs_table.start,   bytes[6],
  longp = vport_table.start,   bytes[6],
  longp = appid_table.start,   bytes[6],
  longp = dimstyle_table.start,   bytes[6],
  longp = p13_table.start,   bytes[6],
  longp  bytes*,
 ).
```

A table position

```
tablepos :=
   word : size,
   long : nr,
   long : start,
```

The header

```
header :=
   word,
   point(TRUE) : inbase,
   point(TRUE) : extmin,
   point(TRUE) : extmax,
   point(FALSE) : limmin,
   point(FALSE) : limmax,
   double[4],
   byte[2],
   double[2],
   byte[56],
   double[3],
   byte[18],
   double .
```

The block table

```
blocks :=
  ( [size](
      byte : flag,
      char[32] : name,
      word : used,
      byte, word, byte, word,
      check_2
    )
  )[nr] : blocks_info,
  check_32.

check_2 := byte[2].
check_32 := byte[32].
```

The layer table

```
layers :=
  ( [size](
      byte : flag,
      char[32] : name,
      word : used,
      word : color,
      word : style,
      check_2
    )
  )[nr] : layer_info,
  check_32.
```

The style table

```
styles :=
  ( [size](
      byte : flag,
      char[32] : name,
      word, double[3], byte, double, char[128],
      check_2
    )
  )[nr] : style_info,
  check_32.
```

The line-type table

```
ltypes :=
  ( [size](
      byte : flag,
      char[32] : name,
      word,  char[48],  byte,
      byte,  double[13],
      check_2
    )
  )[nr] : ltype_info,
  check_32 .
```

The application identifier table

```
appids :=
  ( [size](
      byte : flag,
      char[32] : name,
      word,
      check_2
    )
  )[nr] : appid_info,
  check_32 .
```

The other tables

```
table :=
  ( [size](
      byte : flag,
      [size - 3]byte*,
      check_2
    )
  )[nr],
  check_32 .
```

The entities

(Experimental)

```
entities :=
  ( byte : kind,
    byte : flag,
    word : length,
    [length - 4](
      word : layer,
      word : opts,
      if (flag & 1) then byte : color else color = 0 fi,
      if (flag & 0x40) then byte : extra else extra = 0 fi,
      if (extra & 2) then xdata fi,
      if (flag & 2) then word : type fi,
      if (flag & 4 && kind > 2 && kind != 22) then double : z fi,
      if (flag & 8) then double : th fi,
      if (flag & 0x20) then handle fi,
      if (extra & 4) then word : paper fi,
      switch (kind)
      case 1:  /* LINE */
        point(!(flag & 4)) : l10,
        point(!(flag & 4)) : l11,
        if (opts & 1) then point(TRUE) : l210 fi,
        if (opts & 2) then double : l38 fi,
      case 2:  /* POINT */
        point(!(flag & 4)) : l10,
        if (opts & 1) then point(TRUE) : l210 fi,
        if (opts & 2) then double : l38 fi,
      case 3:  /* CIRCLE */
        point(FALSE) : l10,
        double : l40,
        if (opts & 1) then point(TRUE) : l210 fi,
        if (opts & 2) then double : l38 fi,
      case 4:  /* SHAPE */
        point(FALSE) : l10,
        word : l2,
        if (opts & 1) then point(TRUE) : l210 fi,
        if (opts & 2) then double : l38 fi,
      case 7: /* TEXT */
        point(FALSE) : l10,
        double : l40,
        string : l1,
        if (opts & 1) then double : l50 fi,
        if (opts & 2) then double : l41 fi,
        if (opts & 4) then double : l51 fi,            /*?*/
        if (opts & 8) then byte : l7 fi,
        if (opts & 0x10) then byte : l71 fi,
        if (opts & 0x20) then byte : l72 fi,
        if (opts & 0x40) then point(FALSE) : l11 fi,
        if (opts & 0x100) then byte : l73 fi,
      case 8:  /* ARC */
        point(FALSE) : l10,
        double : l40,
        double : l50,
        double : l51,
        if (opts & 1) then point(TRUE) : l210 fi,
        if (opts & 2) then double : l38 fi,
      case 9:   /* TRACE */
        point(FALSE) : l10,
        point(FALSE) : l11,
        point(FALSE) : l12,
        point(FALSE) : l13,
        if (opts & 1) then point(TRUE) : l210 fi,
        if (opts & 2) then double : l38 fi,
      case 11:   /* SOLID */
        point(FALSE) : l11,
        point(FALSE) : l12,
        point(FALSE) : l13,
        point(FALSE) : l14,
        if (opts & 1) then point(TRUE) : l210 fi,
        if (opts & 2) then double : l38 fi
      case 12:   /* BLOCK */
        point(FALSE) : l10,                /*?*/
        string : l1,                       /* if (opts & 1) then ? */
        if (opts & 2) then string : l3 fi
      case 13:  /* ENDBLK */
      case 14:   /* INSERT */
        word : l1,
        point(FALSE) : l10,
        if (opts & 1) then double : l41 fi,
        if (opts & 2) then double : l42 fi,
        if (opts & 4) then double : l43 fi,
        if (opts & 8) then double : l50 fi,
        if (opts & 0x10) then byte : l70 fi,                /*?*/
        if (opts & 0x20) then byte : l71 fi,                /*?*/
        if (opts & 0x40) then double : l44 fi,              /*?*/
        if (opts & 0x80) then double : l45 fi              /*?*/
      case 15:    /* ATTDEF */
        point(FALSE) : l10,
        double : l40,
        string : l1,
        string : l3,
        string : l2,
        byte : l70,
        if (opts & 1) then byte : l73 fi,          /*?*/
        if (opts & 2) then double : l50 fi,        /*?*/
        if (opts & 4) then double : l41 fi,
        if (opts & 8) then double : l42 fi,
        if (opts & 0x10) then byte : l7 fi,
        if (opts & 0x20) then byte : l71 fi,
        if (opts & 0x40) then byte : l72 fi,
        if (opts & 0x80) then point(FALSE) : l11 fi,   /*?*/
        if (opts & 0x100) then point(TRUE) : l210 fi,
        if (opts & 0x200) then double : l38 fi        /*?*/
      case 16:   /* ATTRIB */
        point(FALSE) : l10,
        double : l40,
        string : l1,
        string : l2,
        byte : l70,
        if (opts & 1) then byte : l73 fi,          /*?*/
        if (opts & 2) then double : l50 fi,        /*?*/
        if (opts & 4) then double : l41 fi,
        if (opts & 8) then double : l42 fi,
        if (opts & 0x10) then byte : l7 fi,
        if (opts & 0x20) then byte : l71 fi,
        if (opts & 0x40) then byte : l72 fi,
              if (opts & 0x80) then point(FALSE) : l11 fi,   /*?*/
              if (opts & 0x100) then point(TRUE) : l210 fi,
              if (opts & 0x200) then double : l38 fi        /*?*/
            case 17:   /* S/BEND */
              long
            case 19:   /* PLINE */
              if (opts & 1) then byte : l70 fi,
              if (opts & 2) then double : l40 fi,                   /*?*/
              if (opts & 4) then byte : l71 fi,         /*?*/
              if (opts & 8) then byte : l72 fi,         /*?*/
              if (opts & 0x10) then byte : l73 fi,         /*?*/
              if (opts & 0x20) then byte : l74 fi,         /*?*/
              if (opts & 0x40) then byte : l75 fi         /*?*/
            case 20:   /* VERTEX */
              point(FALSE) : l10,
              if (opts & 1) then double : l40 fi,           /*?*/
              if (opts & 2) then double : l41 fi,           /*?*/
              if (opts & 4) then byte : l70 fi,             /*?*/
              if (opts & 8) then double : l50 fi           /*?*/
            case 22:   /* 3DFACE */
              point(!(flag & 4)) : l10,
              point(!(flag & 4)) : l11,
              point(!(flag & 4)) : l12,
              point(!(flag & 4)) : l13
            case 23:   /* DIM */
              word : l1,
              point(TRUE) : l10,
              point(FALSE) : l11,   /*?*/
              if (opts & 2) then byte : l70 fi,
              if (opts & 1) then point(TRUE) : l12 fi,    /*?*/
              if (opts & 4) then string : l1 fi,
              if (opts & 8) then point(TRUE) : l13 fi,
              if (opts & 0x10) then point(TRUE) : l14 fi,
              if (opts & 0x20) then point(TRUE) : l15 fi,
              if (opts & 0x40) then point(TRUE) : l16 fi,
              if (opts & 0x80) then double : l40 fi,
              if (opts & 0x100) then double : l50 fi,
              if (opts & 0x200) then double : l51 fi,
              if (opts & 0x400) then double : l52 fi,
              if (opts & 0x800) then double : l53 fi
            case 24:   /* VPORT */
              point(TRUE) : l10,
              double : l40,
              double : l41,
              word : l68
            endswitch
            check_2
          )
        )* : entities.
```

Still need to define xdata and handle. (to be continued...)

Last updated: Tuesday, 09-Jan-96 20:21:36 MET

OpenOffice.org's Documentation of the

Microsoft® Excel File Format

Excel Versions 2, 3, 4, 5, 95, 97, 2000, XP

Author	Daniel Rentz <daniel.rentz@sun.com>
Source	PDF: http://sc.openoffice.org/excelfileformat.pdf XML: http://sc.openoffice.org/excelfileformat.sxw
Project started	2001-Jun-29
Last change	2001-Nov-24

Contents

1 Introduction 4

1.1 File Format Versions 4
1.2 Structure of a Worksheet File (BIFF2-BIFF4) 4
1.3 Structure of a Workbook File (BIFF5-BIFF8) 4
1.4 Structure of a Record 5
1.5 Byte Order 5

2 Basic Substructures 6

2.1 Byte Strings (BIFF2-BIFF7) 6
2.2 Unicode Strings (BIFF8) 6
2.3 RK Values 7
2.4 Error Codes 8
2.5 List of Cached Values 8
2.6 Encoded Document Names 9
2.7 Line Styles for Cell Borders (BIFF3-BIFF8) 10
2.8 Patterns for Cell Background Area (BIFF3-BIFF8) 10
2.9 Cell Attributes (BIFF2) 11

3 Formulas 12

3.1 Common Structure 12
3.2 Operators 13
3.3 Reference Classes 13
3.4 Encoding of Cell References in Tokens 14
3.5 Token Overview 14
3.6 Unary Operator Tokens 16
3.7 Binary Operator Tokens 16
3.8 Function Operator Tokens 18
3.9 Constant Operand Tokens 19
3.10 Operand Tokens 20

4 Worksheet/Workbook Structure 24

4.1 Worksheet Stream (BIFF2-BIFF4) 24
4.2 Workbook Stream (BIFF4) 24
4.3 Workbook Stream (BIFF5-BIFF8) 25
4.4 Shared String Table (BIFF8) 26
4.5 Internal and External References 26
4.6 Array Formulas, Shared Formulas 31
4.7 Multiple Operations (Table Operations) 31
4.8 AutoFilter 31
4.9 Scenarios 31
4.10 Web Queries (BIFF8) 31

5 Worksheet/Workbook Records 32

5.1 Overview, Ordered by Record IDs 32
5.2 Overview, Ordered by Record Names 34
5.3 BLANK 35
5.4 BOF – Begin of File 35
5.5 BOOLERR 36
5.6 CONTINUE 37
5.7 CRN 37
5.8 DCONREF – Data Consolidation Reference 38
5.9 DIMENSIONS 38

5.10 EOF – End of File 38
5.11 EXTERNCOUNT 38
5.12 EXTERNNAME 38
5.13 EXTERNSHEET 40
5.14 EXTSST – Extended SST 41
5.15 FONT 41
5.16 FORMAT 42
5.17 FORMULA 42
5.18 HLINK – Hyperlink 44
5.19 INTEGER 47
5.20 IXFE – Index to XF 47
5.21 LABEL 47
5.22 LABELSST 47
5.23 MULBLANK – Multiple BLANK 48
5.24 MULRK – Multiple RK 48
5.25 NAME 49
5.26 NUMBER 51
5.27 PALETTE 52
5.28 PASSWORD 52
5.29 RK 52
5.30 SCREENTIP 53
5.31 SHEETHDR 53
5.32 SHEETSOFFSET 54
5.33 SST – Shared String Table 54
5.34 STRING 54
5.35 SUPBOOK – External Workbook 54
5.36 XCT – CRN Count 56
5.37 XF – Extended Format 56

6 Drawing Objects, Escher Layer 61

7 Charts 62

8 PivotTables 63

9 Change Tracking 64

1 Introduction

1.1 File Format Versions

The Excel file format is named BIFF (Binary Interchange File Format). The following table shows which Excel version writes which file format.

Excel version	BIFF version	Document type	File format
Excel 2	BIFF2	Worksheet	Stream
Excel 3	BIFF3	Worksheet	Stream
Excel 4	BIFF4	Worksheet or workbook	Stream
Excel 5.0	BIFF5	Workbook	OLE2 storage
Excel 7.0 (Excel 95)	BIFF7	Workbook	OLE2 storage
Excel 97, 2000, XP	BIFF8	Workbook	OLE2 storage

The oldest file format BIFF2 has of course the most restrictions. From BIFF4 on it is possible to store a bundle of sheets, called a workbook. The current format BIFF8 contains major changes towards older BIFF versions, for instance the handling of Unicode strings.

1.2 Structure of a Worksheet File (BIFF2-BIFF4)

Files stored in the BIFF versions BIFF2 to BIFF4 contain all records for a sheet or a BIFF4 workbook in one simple stream. The records are arranged sequential, they are never embedded in other records.

1.3 Structure of a Workbook File (BIFF5-BIFF8)

An Excel workbook with several sheets (from BIFF5 on) is stored as an OLE2 compound file. It contains several streams for different types of data. The following table lists names of possible streams.

Stream name	Contents
Book	BIFF5/BIFF7 workbook stream (→4.3)
Workbook	BIFF8 workbook stream (→4.3)
<05$_H$>SummaryInformation	Document settings
<05$_H$>DocumentSummaryInformation	Document settings
User Names	User names in shared workbooks (→9)
Revision Log	Change tracking log stream (→9)

The names of the streams SummaryInformation and DocumentSummaryInformation contain a leading byte with the value 05$_H$.

It is possible to create substorages like subdirectories in a file system, for instance for the pivot table streams. These storages contain substreams itself.

Storage name	Contents
LNKxxxxxxxx	Storage for a linked OLE object (→6)
MBDxxxxxxxx	Storage for an embedded OLE object (→6)
_SX_DB_CUR	Pivot cache storage. The streams contain cached values for one or more PivotTables (→8).
_VBA_PROJECT_CUR	Visual BASIC project storage

In all streams the records are arranged sequential, they are never embedded in other records. Exception in BIFF8: The Escher object stream is splitted and embedded in several MSODRAWING records (→6).

1.4 Structure of a Record

In an Excel data stream the data is divided into several records. Each record contains specific data for the various features of Excel. The common structure of a record is described in the following table.

Offset	Size	Contents	
0	2	Identifier	} Record header
2	2	Size of the following data (sz)	
4	sz	Data	

The maximum size of the record data is limited. If the size of the record data exceeds the given limits, one or more CONTINUE (→5.6) records will be added. Inside of a CONTINUE record the data of the previous record continues as usual.

In the following descriptions only the record data without the headers is shown. All offsets are relative to the beginning of the record data and not to the entire record. The contents of most of the records differ from version to version. This will be described in separate tables. A few older records are replaced in newer BIFF versions. Excel does not write these old records anymore, but can still read them.

1.5 Byte Order

All data items containing more than one byte are stored using the Little-Endian method. That means the least significant byte is stored first and the most significant byte last. This is valid for all data types like 16-bit-integers, 32-bit-integers, floating-point values and Unicode characters. For instance the 16-bit-integer value 1234_H is converted into the byte sequence 34_H 12_H.

2 Basic Substructures

This chapter contains information about substructures which are part of several records, for instance strings or error codes.

2.1 Byte Strings (BIFF2-BIFF7)

All Excel file formats up to BIFF7 contain simple byte strings. The byte string consists of the length of the string followed by the character array. The length is stored either as 8-bit-integer or as 16-bit-integer, depending on the current record. The string is not zero-terminated.

Offset	Size	Contents
0	1 or 2	Length of the string (character count) (ln)
1 or 2	ln	Character array (8-bit-characters)

2.2 Unicode Strings (BIFF8)

From BIFF8 on, strings are stored in a new Unicode format which allows reading and writing 16-bit-characters. The following tables describe the standard format, but in many records the strings differ from this format. This will be mentioned separately. It is possible (but not required) to store Rich-Text formatting information and extended information for Far-East inside of an Unicode string. This results in four different ways to store a string. The string is not zero-terminated.

2.2.1 Contents of an Unicode string

The string consists of the character count (as usual an 8-bit-integer or a 16-bit-integer), option flags, the character array and optional formatting information. If the string is empty, sometimes the option flags field will not occur. This is mentioned at the respective place.

• Unicode string without additional information

Offset	Size	Contents
0	1 or 2	Length of the string (character count) (ln)
1 or 2	1	Option flags (see below): 00_H or 01_H
2 or 3	ln or 2·ln	Character array (8-bit-characters or 16-bit-characters)

• Unicode string with Rich-Text formatting information

<table>
<tr><th>Offset</th><th>Size</th><th>Contents</th></tr>
<tr><td>0</td><td>1 or 2</td><td>Length of the string (character count) (ln)</td></tr>
<tr><td>1 or 2</td><td>1</td><td>Option flags (see below): 08_H or 09_H</td></tr>
<tr><td>2 or 3</td><td>2</td><td>Number of Rich-Text formatting runs (rt)</td></tr>
<tr><td>4 or 5</td><td>ln or 2·ln</td><td>Character array (8-bit-characters or 16-bit-characters)</td></tr>
<tr><td>var.</td><td>4·rt</td><td>List of rt formatting runs. Each run contains two 16-bit indexes:
<table>
<tr><th>Offset</th><th>Size</th><th>Contents</th></tr>
<tr><td>0</td><td>2</td><td>First formatted character (zero-based)</td></tr>
<tr><td>2</td><td>2</td><td>Index to FONT record (→5.15)</td></tr>
</table>
</td></tr>
</table>

- **Unicode string with Far-East information**

Offset	Size	Contents
0	1 or 2	Length of the string (character count) (`ln`)
1 or 2	1	Option flags (see below): $\texttt{04}_H$ or $\texttt{05}_H$
2 or 3	4	Far-East data size (`sz`)
6 or 7	`ln` or 2·`ln`	Character array (8-bit-characters or 16-bit-characters)
var.	`sz`	Unknown extended data about phonetic, keyboard, etc.

- **Unicode string with Rich-Text and Far-East information**

Offset	Size	Contents
0	1 or 2	Length of the string (character count) (`ln`)
1 or 2	1	Option flags (see below): $\texttt{0C}_H$ or $\texttt{0D}_H$
2 or 3	2	Number of Rich-Text formatting runs (`rt`)
4 or 5	4	Far-East data size (`sz`)
8 or 9	`ln` or 2·`ln`	Character array (8-bit-characters or 16-bit-characters)
var.	4·`rt`	List of `rt` formatting runs. See above for details.
var.	`sz`	Unknown extended data about phonetic, keyboard, etc.

2.2.2 Option flags

Bit	Mask	Contents	
0	$\texttt{01}_H$	0 = 8-bit-characters	1 = 16-bit-characters
2	$\texttt{04}_H$	0 = Contains no Far-East info	1 = Contains Far-East info
3	$\texttt{08}_H$	0 = Contains no Rich-Text info	1 = Contains Rich-Text info

2.3 RK Values

An RK value is an encoded integer or floating-point value. RK values have a size of 4 bytes and are used to decrease file size for floating-point values.

Structure of an RK value (32-bit-value):

Bit	Mask	Contents	
0	$\texttt{00000001}_H$	0 = Value not changed	1 = Value multiplied by 100
1	$\texttt{00000002}_H$	0 = IEEE floating-point value	1 = Integer value
31-2	$\texttt{FFFFFFFC}_H$	Encoded value	

If bit 1 is set to 0, the encoded value represents the 30 most significant bits of an IEEE floating-point value. The 34 least significant bits must be set to zero. If bit 1 is set to 1, the encoded value represents a signed 30-bit-integer value.

If bit 0 is set to 1, the decoded value must be divided by 100 to get the final result.

Examples:

RK value	Decoded value	Result
$\texttt{3FF00000}_H$	Floating-point = 1	1
$\texttt{3FF00001}_H$	Floating-point = 1	0.01
$\texttt{004B5646}_H$	Integer = 1234321	1234321
$\texttt{004B5647}_H$	Integer = 1234321	12343.21

2.4 Error Codes

If the calculation of a formula results in an error or any other action fails, Excel sets a specific error code. These error codes are used for instance in cell records and formulas.

Error code	Error value	Description
00_H	#NULL!	Intersection of two cell ranges is empty
07_H	#DIV/0!	Division by zero
$0F_H$	#VALUE!	Wrong type of operand
17_H	#REF!	Illegal or deleted cell reference
$1D_H$	#NAME?	Wrong function or range name
24_H	#NUM!	Value range overflow
$2A_H$	#N/A!	Argument or function not available

2.5 List of Cached Values

The records CRN (→5.7) and EXTERNNAME (→5.12) and the formula token ptgArray (array constant, →3.10.1) require a list of constant values (floating-point values, strings, boolean values or error codes). These values are stored as a simple list. The number of values is stored before in the respective record or token.

• IEEE floating-point value

Offset	Size	Contents
0	1	01_H (identifier for a floating-point constant)
1	8	IEEE floating-point value

• String value

A string value, BIFF2-BIFF7:

Offset	Size	Contents
0	1	02_H (identifier for a string constant)
1	var.	Byte string, 8-bit string length (→2.1)

A string value, BIFF8:

Offset	Size	Contents
0	1	02_H (identifier for a string constant)
1	var.	Unicode string, 16-bit string length, option flags occur always (→2.2)

• Boolean value

Offset	Size	Contents
0	1	04_H (identifier for a boolean constant)
1	1	0 = FALSE, 1 = TRUE
2	7	Not used

• Error value

Offset	Size	Contents
0	1	10_H (identifier for an error constant)
1	1	Error code (→2.4)
2	7	Not used

2.6 Encoded Document Names

2.6.1 Encoded file names

The intention of encoding file names is to make them more platform independent. Encoded file names occur in the records EXTERNSHEET (BIFF2-BIFF7, →5.13) or SUPBOOK (BIFF8, →5.35) and DCONREF (→5.8).

The first character of the file name is used to determine the type of encoding. In Unicode strings (BIFF8) this could be a 16-bit-value.

First character	Meaning
00_H	Empty sheet name (nothing will follow)
01_H	Encoded file name
02_H	External reference to the own document (nothing will follow)
03_H	External reference to a sheet in the own document (BIFF5/BIFF7)
others	Not encoded. This is the first character of the file name.

Inside of the encoded file name there can occur several control characters.

Control character	Meaning
01_H	An MS-DOS drive letter will follow or „@“ for a local network path
02_H	Start path name on same drive as own document
03_H	End of subdirectory name
04_H	Start path name in parent directory of own document (may occur repeatedly)
06_H	Start path name in startup directory of Excel
09_H	Sheet in the same workbook (BIFF4)

Example: Own document is saved as „C:\path\own.xls“.

Formula	Encoded filename
=own.xls!A1	<02_H>
=Sheet2!A1	<01_H><09_H>Sheet2 (BIFF4 workbook)
=Sheet2!A1	<03_H>Sheet2 (BIFF5/BIFF7)
=[ext.xls]Sheet1!A1	<01_H>[ext.xls]Sheet1
='sub\[ext.xls]'Sheet1!A1	<01_H>sub<03_H>[ext.xls]Sheet1
='\[ext.xls]'Sheet1!A1	<01_H><02_H>[ext.xls]Sheet1
='\sub\[ext.xls]'Sheet1!A1	<01_H><02_H>sub<03_H>[ext.xls]Sheet1
='\sub\sub2\[ext.xls]'Sheet1!A1	<01_H><02_H>sub<03_H>sub2<03_H>[ext.xls]Sheet1
='D:\sub\[ext.xls]'Sheet1!A1	<01_H><01_H>Dsub<03_H>[ext.xls]Sheet1
='..\sub\[ext.xls]'Sheet1!A1	<01_H><04_H>sub<03_H>[ext.xls]Sheet1
='\\pc\sub\[ext.xls]'Sheet1!A1	<01_H><01_H>@pc<03_H>sub<03_H>[ext.xls]Sheet1

2.6.2 Encoded document names for DDE and OLE object links

A DDE link contains the name of the server application and the name of a document. An OLE object link contains a class name and a document name. In both cases the names are stored in one string, separated by the control character 03_H.

Example: A document contains a DDE link to the SO/OOo Calc document „example.sxc“ and an OLE object link to the bitmap file „example.bmp“.

Link	Encoded document name
DDE	soffice<03_H>example.sxc
OLE object	Package<03_H>example.bmp

2.7 Line Styles for Cell Borders (BIFF3-BIFF8)

These line styles are used to define cell borders. The styles `08`H to `0D`H are available in BIFF8 only.

Index	Style	Sample	Index	Style	Sample
`00`H	No line			The following for BIFF8 only:	
`01`H	Thin	────	`08`H	Medium dashed	╍╍╍╍
`02`H	Medium	━━━━	`09`H	Thin dash-dotted	-·-·-·-
`03`H	Dashed	╌╌╌╌	`0A`H	Medium dash-dotted	━·━·━·━
`04`H	Dotted	┈┈┈┈	`0B`H	Thin dash-dot-dotted	-··-··-
`05`H	Thick	▬▬▬▬	`0C`H	Medium dash-dot-dotted	━··━··━
`06`H	Double	════	`0D`H	Slanted medium dash-dotted	━⸝━⸝━⸝━
`07`H	Hair	········			

2.8 Patterns for Cell Background Area (BIFF3-BIFF8)

From BIFF3 on, the cell background area may contain a pattern. Foreground and background colors of the pattern are defined separately. In the following table black is used as foreground color and white as background color.

Index	Pattern	Sample	Index	Pattern	Sample
`00`H		No background			
`01`H	■	■	`0A`H	▩	▩
`02`H	▒	▒	`0B`H	▤	▤
`03`H	▓	▓	`0C`H	▥	▥
`04`H	░	░	`0D`H	▧	▧
`05`H	▤	▤	`0E`H	▨	▨
`06`H	▥	▥	`0F`H	▦	▦
`07`H	▧	▧	`10`H	▒	▒
`08`H	▨	▨	`11`H	░	░
`09`H	▩	▩	`12`H	░	░

2.9 Cell Attributes (BIFF2)

All cell records in BIFF2 contain a cell attribute field with a size of 3 bytes. They contain an index to an XF record (→5.37) and some repeated contents of the referenced XF record. The XF index field has a size of only 6 bits, so the index range is 0-63. If an index >62 is used, the XF index field always contains the vaue 63, and an IXFE record (→5.20) occurs in front of the cell record. It contains the correct index of the XF record.

Cell attributes field (3 bytes), BIFF2:

Offset	Size	Contents		
0	1	Cell protection and XF index:		
		Bit	**Mask**	**Contents**
		5-0	3F$_H$	Index to XF record (→5.37). The value 3F$_H$ (63) indicates a preceding IXFE record (→5.20).
		6	40$_H$	1 = Cell is locked
		7	80$_H$	1 = Formula is hidden
1	1	Indexes to FORMAT and FONT records:		
		Bit	**Mask**	**Contents**
		5-0	3F$_H$	Index to FORMAT record (→5.16)
		7-6	C0$_H$	Index to FONT record (→5.15)
2	1	Cell style:		
		Bit	**Mask**	**Contents**
		2-0	07$_H$	XF_HOR_ALIGN – Horizontal alignment (→5.37.1)
		3	08$_H$	1 = Cell has left black border
		4	10$_H$	1 = Cell has right black border
		5	20$_H$	1 = Cell has top black border
		6	40$_H$	1 = Cell has bottom black border
		7	80$_H$	1 = Cell has shaded background

3 Formulas

3.1 Common Structure

Formulas are stored as part of a record, for instance inside of a FORMULA record or a NAME record. The common format of a formula is as follows:

Formula in BIFF2:

Offset	Size	Contents
0	1	Size of the following formula data (RPN token array) (sz)
1	sz	Formula data (RPN token array)

Formula in BIFF3-BIFF8:

Offset	Size	Contents
0	2	Size of the following formula data (sz)
2	sz	Formula data (RPN token array)

The contents of a formula are stored in the Reverse-Polish Notation (RPN). This means, first occur all operands of an operation, followed by the respective operator. The operands and operators are called tokens. For instance the simple term 1+2 consists of 3 tokens. Written in RPN the formula is converted to the token list „1“, „2“, „+“. During parsing such an expression operands are pushed onto a stack. An operator pops the needed number of operands from stack, performs the operation and pushes the result back onto the stack.

Other examples for RPN token arrays:

Formula	Token array	Parsing result
2·4+5	2, 4, „·“, 5, „+“	The „·“ pops 4 and 2 and pushes 8, the „+“ pops 5 and 8 and pushes 40. That is the result.
2+4·5	2, 4, 5, „·“, „+“	The „·“ pops 5 and 4 and pushes 20, the „+“ pops 20 and 2 and pushes 22. That is the result.

A token can be a simple integer or floating point value, a string constant, a cell reference or cell range reference or an operator. A token is stored as follows:

Offset	Size	Contents
0	1	Token identifier
[1]	var.	(optional) Additional data for the token

Example of the formula for the term 1+2:

Offset	Size	Data	Name	Comment
0	2	0007_H	sz	Size of the following formula data
2	1	$1E_H$	ptgInt	} Integer value token
3	2	0001_H		
5	1	$1E_H$	ptgInt	} Integer value token
6	2	0002_H		
8	1	03_H	ptgAdd	Addition operator

In the following token descriptions, only the additional data following the token identifier is described.

3.2 Operators

There are 3 types of operators:

- Unary operators like the minus sign that negates a value. These operators pop the topmost operand from the stack.
- Binary operators like addition or multiplication. These operators pop the two topmost operands from the stack.
- Function operators represent the sheet functions of Excel. They operate on different numbers of topmost operands on the stack. Either the function expects a fixed number of operands (for instance SIN expects one operand) or a variable number of operands given in the function token (for instance SUM is able to process from 0 to 30 operands).

3.3 Reference Classes

Some of the tokens (especially function operators and operand tokens) exist in 3 different versions: reference class token, value class token and array class token. The token class depends on which type of data the involved operator expects. Sometimes only 1 or 2 token classes make sense.

- Reference class token: The reference itself, independent of the cell contents.
- Value class token: A dereferenced value.
- Array class token: A matrix reference to a cell range.

The structure of the 8-bit operand token identifier is described in the following table.

Bit	Mask	Contents
4-0	$1F_H$	Basic token identifier
6-5	60_H	01_2 = Reference class token (token range 20_H-$3F_H$) 10_2 = Value class token (token range 40_H-$5F_H$) 11_2 = Array class token (token range 60_H-$7F_H$)
7	80_H	0_2 (zero)

The class of an operand token is marked in its name: The names of value class tokens contain a trailing „V“ and the names of array class tokens a trailing „A“.

Examples for the different token classes:

- Reference class token: The formula `=ROW(A1)` returns 1, regardless of the content of A1. Cell reference token is ptgRef (24_H).
- Value class token: The formula `=A1+1` returns the value of the cell A1, increased by 1. Cell reference token is ptgRefV (44_H).
- Array class token: The formula `=MDETERM(A1:C3)` returns the determinant of the values inside of the matrix range A1:C3. Area reference token is ptgAreaA (65_H).

3.4 Encoding of Cell References in Tokens

3.4.1 Cell references in BIFF2-BIFF7

In the BIFF versions up to BIFF5, it is possible to use 16384 rows (2^{14}). A cell reference contains the row index as a 16-bit-value (zero-based, 0-16383), the column index as an 8-bit-value (zero-based, 0-255) and two flags. The flags specify whether the row or column index is absolute or relative.

Contents of the row index (16-bit-value), BIFF2-BIFF7:

Bit	Mask	Contents	
13-0	3FFF$_{H}$	Index to row (0-16383)	
14	4000$_{H}$	0 = Absolute column reference	1 = Relative column reference
15	8000$_{H}$	0 = Absolute row reference	1 = Relative row reference

Example: The reference B$6 has the absolute row index 5 and the relative column index 1. The value of the encoded row index is 4005$_{H}$ (row 6, column is relative). The value of the column index is 01$_{H}$ (column B).

3.4.2 Cell references in BIFF8

From BIFF8 on 65536 (2^{16}) rows are available. Therefore the column index field expands to a 16-bit-value and contains the relative flags.

Contents of the column index (16-bit-value), BIFF8:

Bit	Mask	Contents	
7-0	00FF$_{H}$	Index to column (0-255)	
14	4000$_{H}$	0 = Absolute column reference	1 = Relative column reference
15	8000$_{H}$	0 = Absolute row reference	1 = Relative row reference

Example: The reference B$6 has the absolute row index 5 and the relative column index 1. The value of the encoded column index is 4001$_{H}$ (column B, column is relative). The value of the row index is 0005$_{H}$ (row 6).

3.5 Token Overview

Following a list of all tokens, separated into several token classes and ordered by token identifier.

3.5.1 Unary operator tokens

Token ID	Token name	Description
12$_{H}$	ptgUplus	Unary plus
13$_{H}$	ptgUminus	Unary minus
14$_{H}$	ptgPercent	Percent sign

3.5.2 Binary operator tokens

Token ID	Token name	Description
03_H	ptgAdd	Addition
04_H	ptgSub	Subtraction
05_H	ptgMul	Multiplication
06_H	ptgDiv	Division
07_H	ptgPower	Exponentiation
08_H	ptgConcat	Concatenation
09_H	ptgLT	Less than
$0A_H$	ptgLE	Less than or equal
$0B_H$	ptgEQ	Equal
$0C_H$	ptgGE	Greater than or equal
$0D_H$	ptgGT	Greater than
$0E_H$	ptgNE	Not equal
$0F_H$	ptgIsect	Cell range intersection
10_H	ptgUnion	Cell range union
11_H	ptgRange	Cell range

3.5.3 Function operator tokens

Token ID	Token name	Description
21_H 41_H 61_H	ptgFunc	Function with fixed number of arguments
22_H 42_H 62_H	ptgFuncVar	Function with variable number of arguments

3.5.4 Constant operand tokens

Token ID	Token name	Description
16_H	ptgMissArg	Missing argument
17_H	ptgStr	String constant
$1C_H$	ptgErr	Error value
$1D_H$	ptgBool	Boolean value
$1E_H$	ptgInt	Integer value
$1F_H$	ptgNum	Floating-point number

3.5.5 Operand tokens

Token ID	Token name	Description
20_H 40_H 60_H	ptgArray	Array constant
23_H 43_H 63_H	ptgName	Internal defined name
24_H 44_H 64_H	ptgRef	2D cell reference
25_H 45_H 65_H	ptgArea	2D area reference
$2A_H$ $4A_H$ $6A_H$	ptgRefErr	Deleted 2D cell reference
$2B_H$ $4B_H$ $6B_H$	ptgAreaErr	Deleted 2D area reference
39_H 59_H 79_H	ptgNameX	External name
$3A_H$ $5A_H$ $7A_H$	ptgRef3d	3D cell reference
$3B_H$ $5B_H$ $7B_H$	ptgArea3d	3D area reference
$3C_H$ $5C_H$ $7C_H$	ptgRefErr3d	Deleted 3D cell reference
$3D_H$ $5D_H$ $7D_H$	ptgAreaErr3d	Deleted 3D area reference
2do: more		

3.6 Unary Operator Tokens

Unary operators perform an operation with the topmost operand from stack. The tokens do not contain any additional data.

3.6.1 ptgUplus (12_H)

Unary plus operator. This operator has no effect on the operand.

Example: `+1` returns 1.

3.6.2 ptgUminus (13_H)

Unary minus operator. Negates the operand.

Example: `-1` returns -1.

3.6.3 ptgPercent (14_H)

Percent sign. Divides the operand by 100.

Example: `1%` returns 0.01.

3.7 Binary Operator Tokens

Binary operators perform an operation with the two topmost operands from stack. The tokens do not contain any additional data.

3.7.1 ptgAdd (03_H)

Addition operator. Adds the operands.

Example: `3+2` returns 5.

3.7.2 ptgSub (04$_H$)

Subtraction operator. Subtracts the top operand from the second-to-top operand.
Example: `3-2` returns 1.

3.7.3 ptgMul (05$_H$)

Multiplication operator. Multiplicates the operands.
Example: `3*2` returns 6.

3.7.4 ptgDiv (06$_H$)

Division operator. Divides the second-to-top operand by the top operand.
Example: `3/2` returns 1.5.

3.7.5 ptgPower (07$_H$)

Exponentiation operator. Raises the second-to-top operand to the power of the top operand.
Example: `3^2` returns 9.

3.7.6 ptgConcat (08$_H$)

Concatenation operator. Appends the top operand to the second-to-top operand.
Example: `"ABC"&"DEF"` returns "ABCDEF“.

3.7.7 ptgLT (09$_H$)

Less than operator. Returns TRUE if the second-to-top operand is less than the top operand.
Example: `3<2` returns FALSE.

3.7.8 ptgLE (0A$_H$)

Less than or equal operator. Returns TRUE if the second-to-top operand is less than or equal to the top operand.
Example: `3<=2` returns FALSE.

3.7.9 ptgEQ (0B$_H$)

Equality operator. Returns TRUE if the operands are equal.
Example: `3=2` returns FALSE.

3.7.10 ptgGE (0C$_H$)

Greater than or equal operator. Returns TRUE if the second-to-top operand is greater than or equal to the top operand.
Example: `3>=2` returns TRUE.

3.7.11 ptgGT (0D$_H$)

Greater than operator. Returns TRUE if the second-to-top operand is greater than the top operand.
Example: `3>2` returns TRUE.

3.7.12 ptgNE (0E$_H$)

Inequality operator. Returns TRUE if the operands are not equal.

Example: `3<>2` returns TRUE.

3.7.13 ptgIsect (0F$_H$)

Intersection operator, represented by the space sign. Returns the intersected range of two ranges.

Example: `A1:B3 B2:C3` returns B2:B3.

3.7.14 ptgUnion (10$_H$)

Union operator, represented by the comma sign (for instance english Excel) or semicolon (for instance german Excel). Returns the union of two ranges.

Example: `(A1:A2,A2:A3)` will be handled as one parameter (useful for function parameters).

3.7.15 ptgRange (11$_H$)

Range operator, represented by the colon sign. Returns the rectangular range formed by two ranges. This token occurs for instance by using defined names.

Example: `namedcell:D5`.

3.8 Function Operator Tokens

3.8.1 ptgFunc (21$_H$), ptgFuncV (41$_H$), ptgFuncA (61$_H$)

This token contains the index to a function with fixed number of arguments.

Token ptgFunc, BIFF2-BIFF3:

Offset	Size	Contents
0	1	Index to a sheet function

Token ptgFunc, BIFF4-BIFF8:

Offset	Size	Contents
0	2	Index to a sheet function

3.8.2 ptgFuncVar (21$_H$), ptgFuncVarV (41$_H$), ptgFuncVarA (61$_H$)

This token contains the index to a function with variable number of arguments.

Token ptgFuncVar, BIFF2-BIFF3:

Offset	Size	Contents
0	1	Number of arguments
1	1	Index to a sheet function

Token ptgFuncVar, BIFF4-BIFF8:

Offset	Size	Contents
0	1	Number of arguments
1	2	Index to a sheet function

3.9 Constant Operand Tokens

3.9.1 ptgMissArg (16_H)

A missing argument in a function argument list is stored as a ptgMissArg token. This token does not contain any additional data.

Example: `SUM(1,,3)` – second argument is missing and represented by a ptgMissArg token.

3.9.2 ptgStr (17_H)

This token contains a string constant. The maximum length of the string is 253 characters in BIFF2 (due to the limitation of 255 bytes per formula) and 255 characters in BIFF3-BIFF7.

Token ptgStr, BIFF2-BIFF7:

Offset	Size	Contents
0	var.	Byte string, 8-bit string length (→2.1)

Token ptgStr, BIFF8:

Offset	Size	Contents
0	var.	Unicode string, 16-bit string length, option flags occur always (→2.2)

Example: `"ABC"`.

3.9.3 ptgErr ($1C_H$)

This token contains an error code.

Offset	Size	Contents
0	1	Error code (→2.4)

3.9.4 ptgBool ($1D_H$)

This token contains a boolean value (TRUE or FALSE).

Offset	Size	Contents
0	1	0 = FALSE, 1 = TRUE

3.9.5 ptgInt ($1E_H$)

This token contains an unsigned 16-bit-integer value in the range from 0 to 65535.

Offset	Size	Contents
0	2	Unsigned integer value

3.9.6 ptgNumber ($1F_H$)

This token contains an IEEE floating-point number.

Offset	Size	Contents
0	8	IEEE floating-point number

3.10 Operand Tokens

3.10.1 ptgArray (20_H), ptgArrayV (40_H), ptgArrayA (60_H)

This token contains an array constant. For instance the 2x1 matrix `{1;2}` is an array constant. The values of the array constant do not follow the token identifier but are stored behind the complete token array.

Token ptgArray, BIFF2-BIFF8:

Offset	Size	Contents
0	7	Not used

The constants of the array are stored row by row behind the formula in a list. The length of this list has not been added to the leading formula size field.

Array constant list, BIFF2-BIFF7:

Offset	Size	Contents
0	1	Number of columns (nc). The value 0 means 256 columns.
1	2	Number of rows (nr)
3	var.	List of nc·nr cached values (→2.5)

Array constant list, BIFF8:

Offset	Size	Contents
0	1	Number of columns decreased by 1 (nc)
1	2	Number of rows decreased by 1 (nr)
3	var.	List of (nc+1)·(nr+1) cached values (→2.5)

3.10.2 ptgName (23_H), ptgNameV (43_H), ptgNameA (63_H)

This token contains the one-based index to a NAME record (→5.25). In BIFF2-BIFF4 this could be the index to an EXTERNNAME record (→5.12) too. From BIFF5 on an external name is represented by the token ptgNameX (→3.10.7).

Token ptgName, BIFF2:

Offset	Size	Contents
0	2	One-based index to NAME record (→5.25) or EXTERNNAME record (→5.12)
2	5	Not used

Token ptgName, BIFF3-BIFF4:

Offset	Size	Contents
0	2	One-based index to NAME record (→5.25) or EXTERNNAME record (→5.12)
2	8	Not used

Token ptgName, BIFF5/BIFF7:

Offset	Size	Contents
0	2	One-based index to NAME record (→5.25)
2	12	Not used

Token ptgName, BIFF8:

Offset	Size	Contents
0	2	One-based index to NAME record (→5.25)
2	2	Not used

3.10.3 ptgRef (24_H), ptgRefV (44_H), ptgRefA (64_H)

This token contains the reference to a cell in the same sheet.
Token ptgRef, BIFF2-BIFF7:

Offset	Size	Contents
0	2	Index to row and relative flags (→3.4.1)
2	1	Index to column

Token ptgRef, BIFF8:

Offset	Size	Contents
0	2	Index to row
2	2	Index to column and relative flags (→3.4.2)

3.10.4 ptgArea (25_H), ptgAreaV (45_H), ptgAreaA (65_H)

This token contains the reference to a rectangular cell range in the same sheet.
Token ptgArea, BIFF2-BIFF7:

Offset	Size	Contents
0	2	Index to first row and relative flags (→3.4.1)
2	2	Index to last row and relative flags (→3.4.1)
4	1	Index to first column
5	1	Index to last column

Token ptgArea, BIFF8:

Offset	Size	Contents
0	2	Index to first row
2	2	Index to last row
4	2	Index to first column and relative flags (→3.4.2)
6	2	Index to last column and relative flags (→3.4.2)

3.10.5 ptgRefErr ($2A_H$), ptgRefErrV ($4A_H$), ptgRefErrA ($6A_H$)

This token contains the last reference to a deleted cell in the same sheet. The structure is equal to the token ptgRef (→3.10.3).

3.10.6 ptgAreaErr ($2B_H$), ptgAreaErrV ($4B_H$), ptgAreaErrA ($6B_H$)

This token contains the last reference to a deleted rectangular cell range in the same sheet. The structure is equal to the token ptgArea (→3.10.4).

3.10.7 ptgNameX (39_H), ptgNameXV (59_H), ptgNameXA (79_H) (BIFF5-BIFF8)

This token contains the index to a NAME or EXTERNNAME record. It occurs by using internal or external names, AddIn functions, DDE links or linked OLE objects. See →4.5.2 for details about references in BIFF5/BIFF7 and →4.5.3 for BIFF8.

Token ptgNameX, BIFF5/BIFF7:

Offset	Size	Contents
0	2	One-based index to EXTERNSHEET record (→5.13). A negative value indicates the own workbook. In this case a NAME record is indexed below. The absolute value indexes to the EXTERNSHEET record that contains the sheet name.
2	8	Not used
10	2	One-based index to NAME record (→5.25) or EXTERNNAME record (→5.12)
12	12	Not used

Token ptgNameX, BIFF8:

Offset	Size	Contents
0	2	Index to REF entry in EXTERNSHEET record (→5.13)
2	2	One-based index to NAME record (→5.25) or EXTERNNAME record (→5.12)
4	2	Not used

3.10.8 ptgRef3d ($3A_H$), ptgRef3dV ($5A_H$), ptgRef3dA ($7A_H$) (BIFF5-BIFF8)

This token contains a 3D reference or an external reference to a cell. See →4.5.2 for details about references in BIFF5/BIFF7 and →4.5.3 for BIFF8.

Token ptgRef3d, BIFF5/BIFF7:

Offset	Size	Contents
0	2	One-based index to EXTERNSHEET record (→5.13). A negative value indicates a 3D reference to the own workbook. The absolute value indexes to the EXTERNSHEET record that contains the first sheet name.
2	8	Not used
10	2	3D reference: Index of first referenced sheet; External reference: Not used
12	2	3D reference: Index of last referenced sheet; External reference: Not used
14	2	Index to row and relative flags (→3.4.1)
16	1	Index to column

Token ptgRef3d, BIFF8:

Offset	Size	Contents
0	2	Index to REF entry in EXTERNSHEET record (→5.13)
2	2	Index to row
4	2	Index to column and relative flags (→3.4.2)

3.10.9 ptgArea3d ($3B_H$), ptgArea3dV ($5B_H$), ptgArea3dA ($7B_H$) (BIFF5-BIFF8)

This token contains a 3D reference or an external reference to a rectangular cell range. See →4.5.2 for details about references in BIFF5/BIFF7 and →4.5.3 for BIFF8.

Token ptgArea3d, BIFF5/BIFF7:

Offset	Size	Contents
0	2	One-based index to EXTERNSHEET record (→5.13). A negative value indicates a 3D reference to the own workbook. The absolute value indexes to the EXTERNSHEET record that contains the first sheet name.
2	8	Not used
10	2	3D reference: Index of first referenced sheet; External reference: Not used
12	2	3D reference: Index of last referenced sheet; External reference: Not used
14	2	Index to first row and relative flags (→3.4.1)
16	2	Index to last row and relative flags (→3.4.1)
18	1	Index to first column
19	1	Index to last column

Token ptgArea3d, BIFF8:

Offset	Size	Contents
0	2	Index to REF entry in EXTERNSHEET record (→5.13)
2	2	Index to first row
4	2	Index to last row
6	2	Index to first column and relative flags (→3.4.2)
8	2	Index to last column and relative flags (→3.4.2)

3.10.10 ptgRefErr3d ($3C_H$), ptgRefErr3dV ($5C_H$), ptgRefErr3dA ($7C_H$) (BIFF5-BIFF8)

This token contains the last 3D reference or external reference to a deleted cell. The structure is equal to the token ptgRef3d (→3.10.8).

3.10.11 ptgAreaErr3d ($3D_H$), ptgAreaErr3dV ($5D_H$), ptgAreaErr3dA ($7D_H$) (BIFF5-BIFF8)

This token contains the last 3D reference or external reference to a deleted rectangular cell range. The structure is equal to the token ptgArea3d (→3.10.9).

4 Worksheet/Workbook Structure

In an Excel file, some complex features are splitted into several records. To keep these features consistent, the position and order of the records is very important. This chapter contains details about the correct combination of the records inside of the stream. The internal structure of the records is described in chapter 5.

4.1 Worksheet Stream (BIFF2-BIFF4)

The whole worksheet file consists of the worksheet stream. All records of the worksheet are enclosed by a leading BOF record (→5.4) and a trailing EOF record (→5.10). The sheet contents section contains all information about the worksheet, for instance sheet dimension, view settings, a font list, a list of defined names and external references, of course the contents and formats of all cells, row heights, column widths, drawing objects, chart objects, etc.

Common structure of a worksheet stream:

BOF	Type = worksheet
	Sheet contents
EOF	

4.2 Workbook Stream (BIFF4)

The whole BIFF4 workbook file consists of the workbook stream. It contains a workbook globals section and a list of the worksheets. The sheets are embedded into the outer pair of BOF/EOF records. The workbook globals section contains common information about the workbook, for instance text encoding, global view settings or a list of all sheet names. Additionally, in each workbook a SHEETSOFFSET record (→5.32) is present. The data of the sheets is stored in worksheet substreams. Each substream is preceded by a SHEETHDR record (→5.31) which contains the name of the sheet and the size of the following substream. The SHEETSOFFSET record mentioned above contains the stream position of the first SHEETHDR record. The substreams have the same structure as described in chapter 4.1. Note: In this context the term „substream“ is only a sequence of records and not a storage sub stream of OLE2 storages.

Common structure of a workbook stream with two sheets, BIFF4:

BOF	Type = workbook globals
	Workbook globals
SHEETSOFFSET	Position of the first SHEETHDR record
	Workbook globals
SHEETHDR	Sheet name = „Sheet1“, Byte length of following BOF/EOF record block
BOF	Type = worksheet
	Sheet contents
EOF	
SHEETHDR	Sheet name = „Sheet2“, Byte length of following BOF/EOF record block
BOF	Type = worksheet
	Sheet contents
EOF	
EOF	

4.3 Workbook Stream (BIFF5-BIFF8)

From BIFF5 on an Excel document is stored as an OLE2 storage. The workbook stream is located in the root directory of the storage. In BIFF5/BIFF7 it is named „Book“, in BIFF8 „Workbook“. The names are case-sensitive. In difference to the BIFF4 workbook stream, the worksheet substreams are appended to the workbook globals section, not embedded. The workbook global section and sheet contents section have similar contents as described for BIFF4 workbooks (→4.2).

Common structure of a workbook stream with two sheets, BIFF5-BIFF8:

BOF	Type = workbook globals
	Workbook globals
EOF	
BOF	Type = worksheet
	Sheet contents
EOF	
BOF	Type = worksheet
	Sheet contents
EOF	

4.4 Shared String Table (BIFF8)

A BIFF8 workbook collects all strings of all text cells in a global list, the shared string table (SST). This table is located in the workbook globals section in the record SST (→5.33). An SST record is followed by an EXTSST record (→5.14) which stores stream positions for a string hash table. Text cells are represented by LABELSST records (→5.22) which contain indexes to the shared string table. For reading Excel files only the SST record and the LABELSST records are important.

Example: A workbook contains anywhere the strings „AAA“, „BBB“ and „CCC“.

BOF	Type = workbook globals
	Workbook globals
SST	String 0 = „AAA“ String 1 = „BBB“ String 2 = „CCC“
EXTSST	
	Workbook globals
EOF	
BOF	Type = worksheet
	Sheet contents
LABELSST	String = 0
LABELSST	String = 2
	Sheet contents
LABELSST	String = 1
LABELSST	String = 0
	Sheet contents
EOF	

4.5 Internal and External References

This chapter describes all types of 3D and external references. In detail, this could be:

- a reference to a cell or a cell range of another sheet in the same workbook (3D reference),
- a reference to a cell or a cell range of a sheet in another workbook (external reference),
- a reference to a global or local defined name (internal name),
- a reference to a defined name in another workbook (external name),
- an AddIn function,
- a DDE link,
- an OLE object link.

For external references and external names a combination of XCT and CRN records occurs which store values of cells of the document. In the case the document cannot be found these values will be used to get the result of an external reference. An XCT record (→5.36) contains the number of following CRN records. A CRN record (→5.7) stores the contents of one cell or a sequence of cells of one row. Fragmentary cell ranges or cell ranges spanning over more than one row are splitted into several CRN records. 3D references do not use these records because the referenced cells are located in the own document.

4.5.1 References in BIFF2-BIFF4

2do

4.5.2 References in BIFF5/BIFF7

The document names and sheet names of references are stored in a list of EXTERNSHEET records. Each worksheet contains an EXTERNSHEET list with documents referenced from this sheet. Formulas in the sheet use indexes to the EXTERNSHEET list.

The XCT and CRN records occur behind the last EXTERNNAME record as far as they exist, otherwise directly behind the respective EXTERNSHEET record.

• External and 3D references

External and 3D references are represented in a formula by the tokens ptgRef3d (→3.10.8) or ptgArea3d (→3.10.9). These tokens contain an index to an EXTERNSHEET record located in the own worksheet and indexes to the first and last referenced sheet.

For 3D references, the tokens contain a negative EXTERNSHEET index, indicating a reference into the own workbook. The absolute value is the one-based index of the EXTERNSHEET record that contains the name of the first sheet. If the referenced sheets do not exist anymore, these tokens contain the sheet indexes `FFFF`$_H$ (deleted 3D reference).

Each external reference contains the one-based index to an EXTERNSHEET record. The sheet indexes of the tokens are not used.

Example: A document with 7 sheets (named from „Sheet1" to „Sheet7") contains the formulas
`=Sheet2!A1`,
`=SUM(Sheet4:Sheet6!A1:B3)`,
`=SUM([example.xls]ExtSheet1!A1:B2)` (contents: A1=1.11, B1=2.22, A2=3.33, B2=4.44),
`=[example.xls]ExtSheet3!A1` (contents: „ABCD") and
`=Sheet8!A1`.

EXTERNSHEET 1	Name = „Sheet2"
EXTERNSHEET 2	Name = „Sheet4"
EXTERNSHEET 3	Name = „Sheet6"
EXTERNSHEET 4	Name = „[example.xls]ExtSheet1"
XCT	Number of CRN = 2
CRN 0	Cell range = A1:B1, contents = 1.11, 2.22
CRN 1	Cell range = A2:B2, contents = 3.33, 4.44
EXTERNSHEET 5	Name = „[example.xls]ExtSheet3"
XCT	Number of CRN = 1
CRN 0	Cell range = A1, contents = „ABCD"
EXTERNSHEET 6	Name = „Sheet8"

• Internal names

2do

• External names

2do

• AddIn functions

2do

• DDE links, OLE object links

2do

4.5.3 References in BIFF8

The main data of all types of references is stored in a list inside of the workbook globals section. All formulas use only indexes to use specific references. In BIFF8 each referenced document is represented by a SUPBOOK record (→5.35). A SUPBOOK contains the name of the document and the names of the sheets of the document. After the last SUPBOOK occurs only one EXTERNSHEET record (→5.13). It contains a list with indexes to the SUPBOOKs for each used reference anywhere in the document. Formulas use indexes into this EXTERNSHEET list.

For the following examples an external document „example.xls“ is used. It contains 3 sheets named „ExtSheet1“, „ExtSheet2“ and „ExtSheet3“.

Example: A document contains (among other references) the two formulas
`=[example.xls]ExtSheet2!A1` and
`=[example.xls]ExtSheet1!A1`.

	Workbook globals
SUPBOOK 0	Any content
SUPBOOK 1	Document = „example.xls“ Sheet 0 = „ExtSheet1“ Sheet 1 = „ExtSheet2“ Sheet 2 = „ExtSheet3“
SUPBOOK 2	Any content
EXTERNSHEET	REF 0 = any reference REF 1 = {SUPBOOK = 1, sheet range = 1…1} REF 2 = any reference REF 3 = {SUPBOOK = 1, sheet range = 0…0} REF 4 = any reference
	Workbook globals

The first formula uses REF 1 in the EXTERNSHEET record. REF 1 refers to SUPBOOK 1 and sheet range 1…1. This means, the document „example.xls“ is used (document of SUPBOOK 1) and the name of the sheet is „ExtSheet2“ (sheet 1 of SUPBOOK 1). In the same way, the second formula uses REF 3 in the EXTERNSHEET record. All list entries inside of the EXTERNSHEET record are unique. For instance all formulas in the workbook referring to sheet „ExtSheet2“ of the document „example.xls“ use REF 1. All other SUPBOOKs and REFs are placeholders for other references in this example.

The XCT and CRN records occur behind the EXTERNNAME records as far as they exist, otherwise directly behind the respective SUPBOOK record.

• External and 3D references

The SUPBOOK for the own document has a special format: It contains only the number of all sheets and the value $\texttt{0401}_H$ instead of the sheet names. The sheet range indexes in the EXTERNSHEET record refer to the position of the sheets (zero-based). If a referenced sheet does not exist anymore, the sheet index $\texttt{FFFF}_H$ will occur (deleted 3D reference).

Example: A document with 7 sheets (named from „Sheet1“ to „Sheet7“) contains the formulas
`=Sheet2!A1`,
`=SUM(Sheet4:Sheet6!A1:B3)`,
`=SUM([example.xls]ExtSheet1!A1:B2)` (contents: A1=1.11, B1=2.22, A2=3.33, B2=4.44),
`=[example.xls]ExtSheet3!A1` (contents: „ABCD“) and
`=Sheet8!A1`.

SUPBOOK 0	Number of sheets: 7 `0401`$_H$ (own workbook)
SUPBOOK 1	Document = „example.xls“ Sheet 0 = „ExtSheet1“ Sheet 1 = „ExtSheet2“ Sheet 2 = „ExtSheet3“
XCT	Number of CRN = 2, sheet = 0 (ExtSheet1)
CRN 0	Cell range = A1:B1, contents = 1.11, 2.22
CRN 1	Cell range = A2:B2, contents = 3.33, 4.44
XCT	Number of CRN = 1, sheet = 2 (ExtSheet3)
CRN 0	Cell range = A1, contents = „ABCD“

EXTERNSHEET	REF 0 = {SUPBOOK = 0, sheet range = 1…1}
	REF 1 = {SUPBOOK = 0, sheet range = 3…5}
	REF 2 = {SUPBOOK = 1, sheet range = 0…0}
	REF 3 = {SUPBOOK = 1, sheet range = 1…1}
	REF 4 = {SUPBOOK = 0, sheet range = FFFF$_H$…FFFF$_H$}

Inside of the first formula the cell reference is represented by the token ptgRef3d (→3.10.8). The second formula contains the token ptgArea3d (→3.10.9).

• Internal names

All internal names are stored in a list of NAME records (→5.25) that follows the EXTERNSHEET record. There exist two types of internal names: global names which are valid in the whole workbook and local names which are attached to a specific sheet. For instance the local name „MyCell“ of the sheet „Sheet1“ can be used from everywhere in the workbook by entering `=Sheet1!MyCell`. Each NAME record contains the name itself and a one-based sheet index. The index zero indicates a global name. If a SUPBOOK contains local names, a special REF entry will be created in the EXTERNSHEET record. It contains the index to the SUPBOOK and the sheet range FFFE$_H$…FFFE$_H$.

Example for internal names: A document contains
- The global name „GlobalName“,
- The local names „Sheet1!Name“ and „Sheet2!Name“ and
- In Sheet1 the formulas `=GlobalName`, `=Name`, `=Sheet1!Name` and `=Sheet2!Name`.

SUPBOOK 0	Number of sheets: 3 `0401`$_H$ (own workbook)

EXTERNSHEET	REF 0 = {SUPBOOK = 0, sheet range = 0…0}
	REF 1 = {SUPBOOK = 0, sheet range = FFFE$_H$…FFFE$_H$}

NAME 1	Name = „GlobalName“, sheet = 0 (Global)
NAME 2	Name = „Name“, sheet = 1 (Sheet1)
NAME 3	Name = „Name“, sheet = 2 (Sheet2)

Inside of the formula a global name or a local name of the own sheet is represented by the token ptgName (→3.10.2) with an one-based index to the NAME record list. The first formula in the example above contains the token ptgNameV with index 1 and the second formula the same token with index 2.

Local names from other sheets are represented by the token ptgNameX (→3.10.7) with an index to the special REF entry of the EXTERNSHEET record and an index to the NAME record list. The third formula contains the token ptgNameX with the value {REF = 1, Name = 2} and the last formula the same token with the value {REF = 1, Name = 3}. Ref 1 refers to SUPBOOK 0 and Name 2 or Name 3 refer to the respective NAME records.

• External names

In Excel formulas can use names located in another workbook. In this case for each name an EXTERNNAME record (→5.12) after the SUPBOOK record occurs. The EXTERNNAME record contains the name itself and the one-based index to the sheet. Again the index zero indicates a global name. If a SUPBOOK contains external names, a special REF entry will be created in the EXTERNSHEET record. It contains the index to the SUPBOOK and the sheet range FFFE$_{\text{H}}$...FFFE$_{\text{H}}$.

Example: A document contains the formulas
`=example.xls!GlobalName` (location: ExtSheet1!B22; contents: 22),
`=[example.xls]ExtSheet3!Name` (location: ExtSheet3!C33; contents: „ABCD“) and
`=[example.xls]ExtSheet1!Name` (location: ExtSheet1!A11; contents: 11).

SUPBOOK 0	Document = „example.xls“ Sheet 0 = „ExtSheet1“ Sheet 1 = „ExtSheet2“ Sheet 2 = „ExtSheet3“
EXTERNNAME 1	Name = „GlobalName“, sheet = 0 (Global)
EXTERNNAME 2	Name = „Name“, sheet = 3 (ExtSheet3)
EXTERNNAME 3	Name = „Name“, sheet = 1 (ExtSheet1)
XCT	Number of CRN = 2, sheet = 0 (ExtSheet1)
CRN 0	Cell range = A11, contents = 11
CRN 1	Cell range = B22, contents = 22
XCT	Number of CRN = 1, sheet = 2 (ExtSheet3)
CRN 0	Cell range = C33, contents = „ABCD“
EXTERNSHEET	REF 1 = {SUPBOOK = 0, sheet range = FFFE$_{\text{H}}$...FFFE$_{\text{H}}$}

Inside of a formula an external name is represented by the token ptgNameX (→3.10.7). It contains the index to the special REF entry inside of the EXTERNSHEET record and the index to a EXTERNNAME record (one-based). The second formula in the example above contains the token ptgNameXV with the value {REF = 0, ExtName = 2}. REF 1 refers to SUPBOOK 0 and ExtName 2 refers to EXTERNNAME 2 (of SUPBOOK 0).

• AddIn functions

AddIn functions are stored similar to external names. If a formula uses an AddIn function, a special SUPBOOK containing only the value `3A01`$_{\text{H}}$ will occur. Behind of this SUPBOOK the names of all used AddIn functions are listed, each inside of an EXTERNNAME record. A special REF entry with the sheet range FFFE$_{\text{H}}$...FFFE$_{\text{H}}$ will be inserted into the EXTERNSHEET reference list.

Example: A document contains the formulas `=ISODD(1)` and `=ISEVEN(1)`.

SUPBOOK 0	`3A01`$_{\text{H}}$ (AddIn)
EXTERNNAME 1	Name = „ISODD“
EXTERNNAME 2	Name = „ISEVEN“
EXTERNSHEET	REF 0 = {SUPBOOK = 0, sheet range = FFFE$_{\text{H}}$...FFFE$_{\text{H}}$}

• DDE links, OLE object links

DDE links and OLE object links expect the name of the server application (DDE) or the class name (OLE) and the name of a source document. These items are encoded in a SUPBOOK record. The SUPBOOK is followed by EXTERNNAME records with additional data of the links. An EXTERNNAME record for a DDE links contains the item (data source range) and an EXTERNNAME record for an OLE object link contains the identifier of the object data storage.

Example: A document contains a DDE link to the range „Sheet1.A1:B2“ inside of the Calc document „example.sxc“ and an OLE object link to the bitmap file „example.bmp“.

SUPBOOK 0	Server application = „soffice“ Document = „example.sxc“

EXTERNNAME 1	Type = DDE link
	Item = „Sheet1.A1:B2“

SUPBOOK 1	Class name = „Package“ Document = „example.bmp“

EXTERNNAME 1	Type = OLE object link
	Storage = 00012345$_H$ (storage name = „LNK00012345“)

EXTERNSHEET	REF 0 = {SUPBOOK = 0, sheet range = FFFE$_H$...FFFE$_H$}
	REF 1 = {SUPBOOK = 1, sheet range = FFFE$_H$...FFFE$_H$}

Inside of a formula a DDE link is represented by the token ptgNameX (→3.10.7). An OLE object link contains a ptgNameX token inside of its OBJ record.

4.6 Array Formulas, Shared Formulas

2do

4.7 Multiple Operations (Table Operations)

2do

4.8 AutoFilter

2do

4.9 Scenarios

2do

4.10 Web Queries (BIFF8)

2do

5 Worksheet/Workbook Records

5.1 Overview, Ordered by Record IDs

Record ID	Record name	Occurs in BIFF versions					
		2	3	4	5	7	8
0000$_H$	DIMENSIONS	x					
0001$_H$	BLANK	x					
0002$_H$	INTEGER	x					
0003$_H$	NUMBER	x					
0004$_H$	LABEL	x					
0005$_H$	BOOLERR	x					
0006$_H$	FORMULA	x			x	x	x
0007$_H$	STRING	x					
0009$_H$	BOF	x					
000A$_H$	EOF	x	x	x	x	x	x
0013$_H$	PASSWORD	x	x	x	x	x	x
0016$_H$	EXTERNCOUNT	x	x	x	x	x	
0017$_H$	EXTERNSHEET	x	x	x	x	x	x
0018$_H$	NAME	x			x	x	x
001E$_H$	FORMAT	x	x				
0023$_H$	EXTERNNAME	x			x	x	x
0031$_H$	FONT	x			x	x	x
003C$_H$	CONTINUE	x	x	x	x	x	x
0043$_H$	XF	x					
0044$_H$	IXFE	x					
0051$_H$	DCONREF	x	x	x	x	x	x
0059$_H$	XCT		x	x	x	x	x
005A$_H$	CRN		x	x	x	x	x
008E$_H$	SHEETSOFFSET			x			
008F$_H$	SHEETHDR			x			
0092$_H$	PALETTE	x	x	x	x	x	x
00BD$_H$	MULRK				x	x	x
00BE$_H$	MULBLANK				x	x	x
00E0$_H$	XF				x	x	x
00FC$_H$	SST						x
00FD$_H$	LABELSST						x
00FF$_H$	EXTSST						x
01AE$_H$	SUPBOOK						x
01B8$_H$	HLINK						x
0200$_H$	DIMENSIONS		x	x	x	x	x
0201$_H$	BLANK		x	x	x	x	x
0203$_H$	NUMBER		x	x	x	x	x
0204$_H$	LABEL		x	x	x	x	

Record ID	Record name	Occurs in BIFF versions					
		2	3	4	5	7	8
0205$_{\mathrm{H}}$	BOOLERR		x	x	x	x	x
0206$_{\mathrm{H}}$	FORMULA		x				
0207$_{\mathrm{H}}$	STRING		x	x	x	x	x
0209$_{\mathrm{H}}$	BOF		x				
0218$_{\mathrm{H}}$	NAME		x	x			
0223$_{\mathrm{H}}$	EXTERNNAME		x	x			
0231$_{\mathrm{H}}$	FONT		x	x			
0243$_{\mathrm{H}}$	XF		x				
027E$_{\mathrm{H}}$	RK		x	x	x	x	x
0406$_{\mathrm{H}}$	FORMULA			x			
0409$_{\mathrm{H}}$	BOF			x			
041E$_{\mathrm{H}}$	FORMAT			x	x	x	x
0443$_{\mathrm{H}}$	XF			x			
0800$_{\mathrm{H}}$	SCREENTIP						x
0809$_{\mathrm{H}}$	BOF				x	x	x
2do: more							

5.2 Overview, Ordered by Record Names

Record ID	Record name	Occurs in BIFF versions					
		2	3	4	5	7	8
0001$_H$ 0201$_H$	BLANK	x	x	x	x	x	x
0*09$_H$	BOF	x	x	x	x	x	x
0005$_H$ 0205$_H$	BOOLERR	x	x	x	x	x	x
003C$_H$	CONTINUE	x	x	x	x	x	x
005A$_H$	CRN		x	x	x	x	x
0051$_H$	DCONREF	x	x	x	x	x	x
0000$_H$ 0200$_H$	DIMENSIONS	x	x	x	x	x	x
000A$_H$	EOF	x	x	x	x	x	x
0016$_H$	EXTERNCOUNT	x	x	x	x	x	
0023$_H$ 0223$_H$	EXTERNNAME	x	x	x	x	x	x
0017$_H$	EXTERNSHEET	x	x	x	x	x	x
00FF$_H$	EXTSST						x
0031$_H$ 0231$_H$	FONT	x	x	x	x	x	x
001E$_H$ 041E$_H$	FORMAT	x	x	x	x	x	x
0*06$_H$	FORMULA	x	x	x	x	x	x
01B8$_H$	HLINK						x
0002$_H$	INTEGER	x					
0044$_H$	IXFE	x					
0004$_H$ 0204$_H$	LABEL	x	x	x	x	x	
00FD$_H$	LABELSST						x
00BE$_H$	MULBLANK				x	x	x
00BD$_H$	MULRK				x	x	x
0018$_H$ 0218$_H$	NAME	x	x	x	x	x	x
0003$_H$ 0203$_H$	NUMBER	x	x	x	x	x	x
0092$_H$	PALETTE	x	x	x	x	x	x
0013$_H$	PASSWORD	x	x	x	x	x	x
027E$_H$	RK		x	x	x	x	x
0800$_H$	SCREENTIP						x
008F$_H$	SHEETHDR			x			
008E$_H$	SHEETSOFFSET			x			
00FC$_H$	SST						x
0007$_H$ 0207$_H$	STRING	x	x	x	x	x	x
01AE$_H$	SUPBOOK						x
0059$_H$	XCT		x	x	x	x	x
0*43$_H$ 00E0$_H$	XF	x	x	x	x	x	x
2do: more							

5.3 BLANK

BIFF2	BIFF3	BIFF4	BIFF5	BIFF7	BIFF8
`0001`$_H$	`0201`$_H$	`0201`$_H$	`0201`$_H$	`0201`$_H$	`0201`$_H$

This record represents an empty cell. It contains the cell address and formatting information.

Record BLANK, BIFF2:

Offset	Size	Contents
0	2	Index to row
2	2	Index to column
4	3	Cell attributes (→2.9)

Record BLANK, BIFF3-BIFF8:

Offset	Size	Contents
0	2	Index to row
2	2	Index to column
4	2	Index to XF record (→5.37)

5.4 BOF – Begin of File

BIFF2	BIFF3	BIFF4	BIFF5	BIFF7	BIFF8
`0009`$_H$	`0209`$_H$	`0409`$_H$	`0809`$_H$	`0809`$_H$	`0809`$_H$

The BOF record is the first record of a worksheet, the workbook globals section, a chart or a macro sheet.

Record BOF, BIFF2:

Offset	Size	Contents	
0	2	Version	
2	2	Type of the following data:	`0010`$_H$ = Worksheet `0020`$_H$ = Chart `0040`$_H$ = Macro sheet

Record BOF, BIFF3:

Offset	Size	Contents	
0	2	Version	
2	2	Type of the following data:	`0010`$_H$ = Worksheet `0020`$_H$ = Chart `0040`$_H$ = Macro sheet
4	2	Not used	

Record BOF, BIFF4:

Offset	Size	Contents	
0	2	Version	
2	2	Type of the following data:	`0010`$_H$ = Worksheet `0020`$_H$ = Chart `0040`$_H$ = Macro sheet `0100`$_H$ = Workbook globals
4	2	Not used	

Record BOF, BIFF5/BIFF7:

Offset	Size	Contents	
0	2	Version	
2	2	Type of the following data:	0005$_H$ = Workbook globals 0006$_H$ = Visual Basic module 0010$_H$ = Worksheet 0020$_H$ = Chart 0040$_H$ = BIFF4 Macro sheet 0100$_H$ = BIFF4 Workbook globals
4	2	Build identifier	
6	2	Build year	

Record BOF, BIFF8:

Offset	Size	Contents	
0	2	Version, contains 0600$_H$ for BIFF8	
2	2	Type of the following data:	0005$_H$ = Workbook globals 0006$_H$ = Visual Basic module 0010$_H$ = Worksheet 0020$_H$ = Chart 0040$_H$ = BIFF4 Macro sheet 0100$_H$ = BIFF4 Workbook globals
4	2	Build identifier	
6	2	Build year	
8	4	File history flags	
12	4	Lowest Excel version that can read all records in this file	

5.5 BOOLERR

BIFF2	BIFF3	BIFF4	BIFF5	BIFF7	BIFF8
0005$_H$	0205$_H$	0205$_H$	0205$_H$	0205$_H$	0205$_H$

This record represents a boolean or error value cell.
Record BOOLERR, BIFF2:

Offset	Size	Contents
0	2	Index to row
2	2	Index to column
4	3	Cell attributes (→2.9)
7	1	Boolean or error value, depending on the following byte
8	1	0 = Boolean value; 1 = Error code

Record BOOLERR, BIFF3-BIFF8:

Offset	Size	Contents
0	2	Index to row
2	2	Index to column
4	2	Index to XF record (→5.37)
6	1	Boolean or error value, depending on the following byte
7	1	0 = Boolean value; 1 = Error code

If the value field is a boolean value, it will contain 0 for FALSE and 1 for TRUE. See →2.4 for a list of error codes.

5.6 CONTINUE

BIFF2	BIFF3	BIFF4	BIFF5	BIFF7	BIFF8
003C$_H$	003C$_H$	003C$_H$	003C$_H$	003C$_H$	003C$_H$

Everytime the content of a record exceeds the given limits (see table), the record must be splitted. Several CONTINUE records containing the additional data are added after the parent record.

BIFF version	Maximum data size of a record
BIFF2-BIFF7	2080 bytes (2084 bytes including record header)
BIFF8	8224 bytes (8228 bytes including record header)

Record CONTINUE, BIFF2-BIFF8:

Offset	Size	Contents
0	var.	data continuation of the previous record

Unicode strings are splitted in a special way. At the beginning of each CONTINUE record the option flags byte is repeated. Only the character size flag will be set in this flags byte, the Rich-Text flag and the Far-East flag are set to zero.

Attention: In each CONTINUE record it is possible that the character size changes from 8-bit-characters to 16-bit-characters and vice versa. Never an Unicode string is splitted between character count field and option flags field or between option flags field and first character.

Example: The remaining size of a record may be 10 bytes (it has 8214 bytes of data). Now the string „ABCDEFGHØI“ has to be stored in this record. „Ø“ may be a special character with the character code 1234$_H$. Note: The records are shown with their headers to make the example clearer.

Offset	Size	Contents	Description
0	2		Any record identifier
2	2	2020$_H$ (8224)	Record data size
4	8214		Any data
8218	2	000A$_H$ (10)	Unicode string character count
8220	1	00$_H$	Unicode string option flags (8-bit-characters)
8221	7	41$_H$ 42$_H$ … 47$_H$	8-bit-character array „ABCDEFG“
8228	2	003C$_H$	Record identifier CONTINUE
8230	2	0007$_H$ (7)	Record data size
8232	1	01$_H$	Unicode string option flags (16-bit-characters)
8233	2	0048$_H$	16-bit-character „H“
8235	2	1234$_H$	16-bit-character „Ø“
8237	2	0049$_H$	16-bit-character „I“

5.7 CRN

BIFF2	BIFF3	BIFF4	BIFF5	BIFF7	BIFF8
---	005A$_H$	005A$_H$	005A$_H$	005A$_H$	005A$_H$

This record stores the contents of an external cell or cell range. An external cell range has one row only. If a cell range spans over more than one row, several CRN records will be created. See →4.5 for details about external references.

Record CRN, BIFF3-BIFF8:

Offset	Size	Contents
0	1	Index to last column inside of the referenced sheet (lc)
1	1	Index to first column inside of the referenced sheet (fc)
2	2	Index to row inside of the referenced sheet
4	var.	List of lc-fc+1 cached values (→2.5)

5.8 DCONREF – Data Consolidation Reference

BIFF2	BIFF3	BIFF4	BIFF5	BIFF7	BIFF8
0051$_H$	0051$_H$	0051$_H$	0051$_H$	0051$_H$	0051$_H$

2do

5.9 DIMENSIONS

BIFF2	BIFF3	BIFF4	BIFF5	BIFF7	BIFF8
0000$_H$	0200$_H$	0200$_H$	0200$_H$	0200$_H$	0200$_H$

2do

5.10 EOF – End of File

BIFF2	BIFF3	BIFF4	BIFF5	BIFF7	BIFF8
000A$_H$	000A$_H$	000A$_H$	000A$_H$	000A$_H$	000A$_H$

This record has no content. It indicates the end of a record block with leading BOF record (→5.4). This could be the end of the workbook globals, a worksheet, a chart, etc.

5.11 EXTERNCOUNT

BIFF2	BIFF3	BIFF4	BIFF5	BIFF7	BIFF8
0016$_H$	0016$_H$	0016$_H$	0016$_H$	0016$_H$	---

This record contains the number of following EXTERNSHEET records. In BIFF8 this record is omitted because there occurs only one EXTERNSHEET record. See →4.5.1 for details about external references in BIFF2-BIFF4 and →4.5.2 for BIFF5/BIFF7.

Record EXTERNCOUNT, BIFF2-BIFF7:

Offset	Size	Contents
0	2	Number of following EXTERNSHEET records (→5.13)

5.12 EXTERNNAME

BIFF2	BIFF3	BIFF4	BIFF5	BIFF8
0023$_H$	0223$_H$	0223$_H$	0023$_H$	0023$_H$

This record contains the name of an external defined name, the name of an AddIn function, a DDE link item or an OLE object storage name (BIFF8).

• EXTERNNAME in BIFF2-BIFF7

The meaning of the name is dependent on the leading EXTERNSHEET record (→5.13). See →4.5.1 for details about external references in BIFF2-BIFF4 and →4.5.2 for BIFF5/BIFF7.

Record EXTERNNAME, BIFF2-BIFF7:

Offset	Size	Contents
0	var.	External name (byte string, 8-bit string length, →2.1)

If the record contains an item of a DDE link, a list with cached values will be appended to the string. These values are used as results for the DDE link. They are saved row by row for a DDE link that spans over several cells. Note: Only the results of the DDE link (the contents of the referenced cells) are stored, not the results of the complete formulas.

Record EXTERNNAME for DDE items, BIFF2-BIFF7:

Offset	Size	Contents
0	var.	DDE item (byte string, 8-bit string length, →2.1)
var.	1	Number of columns (nc). The value 0 means 256 columns.
var.	2	Number of rows (nr)
var.	var.	List of nc·nr cached values (→2.5)

• EXTERNNAME in BIFF8

In BIFF8 the record contains option flags which describe the type of the external name. So, this record must follow the correct SUPBOOK record (→5.35) and must contain the correct flags. See →4.5.3 for details about external references in BIFF8.

Record EXTERNNAME for external names and AddIn functions, BIFF8:

Offset	Size	Contents
0	2	Option flags (see below)
2	2	One-based sheet index. The value 0 means all sheets or AddIn function.
4	2	Not used
6	var.	External name or AddIn function name (Unicode string, 8-bit string length, →2.2)
var.	var.	For external names only: formula data (RPN token array, →3)

Record EXTERNNAME for DDE links, BIFF8:

Offset	Size	Contents
0	2	Option flags (see below)
2	4	Not used
6	var.	DDE item (Unicode string, 8-bit string length, →2.2)
var.	1	Number of columns decreased by 1 (nc)
var.	2	Number of rows decreased by 1 (nr)
var.	var.	List of (nc+1)·(nr+1) cached values (→2.5)

Record EXTERNNAME for OLE object links, BIFF8:

Offset	Size	Contents
0	2	Option flags (see below)
2	4	Storage identifier
6	3	01_H 00_H 27_H

Option flags:

Bit	Mask	Contents	
0	0001_H	0 = No BuiltIn name	1 = BuiltIn name
1	0002_H	0 = Manual DDE/OLE link	1 = Automatic DDE/OLE link
4	0010_H	0 = External name or DDE link	1 = OLE object link
14-5	$7FE0_H$	For DDE links only: clipboard format of last successful update	

5.13 EXTERNSHEET

BIFF2	BIFF3	BIFF4	BIFF5	BIFF7	BIFF8
`0017`$_H$	`0017`$_H$	`0017`$_H$	`0017`$_H$	`0017`$_H$	`0017`$_H$

• EXTERNSHEET in BIFF2-BIFF7

In the file format versions up to BIFF7 this record stores the name of an external document and a sheet name inside of this document. See →4.5.1 for details about external references in BIFF2-BIFF4 and →4.5.2 for BIFF5/BIFF7.

Record EXTERNSHEET, BIFF2-BIFF7:

Offset	Size	Contents
0	var.	Encoded document and sheet name (→2.6). Byte string, 8-bit string length (→2.1).

Attention: The string length field is decreased by 1, if the EXTERNSHEET stores a reference to one of the own sheets (first character is `03`$_H$). Example: The formula `=Sheet2!A1` contains a reference to an EXTERNSHEET record with the string „<`03`$_H$>Sheet2“. The string consists of 7 characters but the string length field contains the value 6.

If a formula uses an AddIn function, a special EXTERNSHEET record will occur, followed by an EXTERNNAME record with the name of the function.

Record EXTERNSHEET for AddIn functions, BIFF2-BIFF7:

Offset	Size	Contents
0	2	`3401`$_H$ (`01`$_H$ `34`$_H$ = the byte string „#“)

• EXTERNSHEET in BIFF8

In BIFF8 the record stores a list with indexes to SUPBOOK records (list of REF structures). See →4.5.3 for details about external references in BIFF8.

Record EXTERNSHEET, BIFF8:

<table>
<tr><th>Offset</th><th>Size</th><th>Contents</th></tr>
<tr><td>0</td><td>2</td><td>Number of following REF structures (nm)</td></tr>
<tr><td>2</td><td>6·nm</td><td>List of nm REF structures. Each REF contains the following data:
<table>
<tr><th>Offset</th><th>Size</th><th>Contents</th></tr>
<tr><td>0</td><td>2</td><td>Index to SUPBOOK record</td></tr>
<tr><td>2</td><td>2</td><td>Index to first SUPBOOK sheet</td></tr>
<tr><td>4</td><td>2</td><td>Index to last SUPBOOK sheet</td></tr>
</table></td></tr>
</table>

5.14 EXTSST – Extended SST

BIFF2	BIFF3	BIFF4	BIFF5	BIFF7	BIFF8
---	---	---	---	---	00FF$_{H}$

This record occurs in conjunction with the SST record (→5.33). It contains a hash table with stream offsets to the SST record to optimize string search operations. Excel does not shorten this record if strings are deleted from the shared string table, so the last part might contain invalid data. The stream indexes in this record divide the SST into hash buckets containing a constant number of strings. See →4.4 for more information about shared string tables.

Record EXTSST, BIFF8:

<table>
<tr><th>Offset</th><th>Size</th><th colspan="3">Contents</th></tr>
<tr><td>0</td><td>2</td><td colspan="3">Number of strings in a hash bucket, this number is ⩾8</td></tr>
<tr><td>2</td><td>var.</td><td colspan="3">List of OFFSET structures. Each OFFSET contains the following data:</td></tr>
<tr><td></td><td></td><th>Offset</th><th>Size</th><th>Contents</th></tr>
<tr><td></td><td></td><td>0</td><td>4</td><td>Absolute stream position of first string of this bucket</td></tr>
<tr><td></td><td></td><td>4</td><td>2</td><td>Position of first string of this bucket inside of current record, including record header. This counter restarts at zero inside of CONTINUE records.</td></tr>
<tr><td></td><td></td><td>6</td><td>2</td><td>Not used</td></tr>
</table>

5.15 FONT

BIFF2	BIFF3	BIFF4	BIFF5	BIFF7	BIFF8
0031$_{H}$	0231$_{H}$	0231$_{H}$	0031$_{H}$	0031$_{H}$	0031$_{H}$

This record contains information about an used font, including character formatting.

Some of the elements occur unchanged in every BIFF version. These elements are described in the following tables using a specific name for each element. In the description of the record structure the names are used to reference to these tables.

5.15.1 FONT substructures

- **FONT_SCRIPT – Subscript or superscript (2 bytes), BIFF5-BIFF8**

Value	Contents
0000$_{H}$	None
0001$_{H}$	Superscript
0002$_{H}$	Subscript

- **FONT_UNDERLINE – Underline type (1 byte), BIFF5-BIFF8**

Value	Contents
00$_{H}$	None
01$_{H}$	Single
02$_{H}$	Double
03$_{H}$	Single accounting
04$_{H}$	Double accounting

5.15.2 FONT record contents

Record FONT, BIFF2:

Offset	Size	Contents		
0	2	Height of the font (in 1/20 of a point)		
2	2	Option flags:		
		Bit	**Mask**	**Contents**
		0	0001_{H}	1 = Characters are bold
		1	0002_{H}	1 = Characters are italic
		2	0004_{H}	1 = Characters are underlined
		3	0008_{H}	1 = Characters are struck out
4	var.	Font name (byte string, 8-bit string length, →2.1)		

Record FONT, BIFF3-BIFF4:

Offset	Size	Contents		
0	2	Height of the font (in 1/20 of a point)		
2	2	Option flags:		
		Bit	**Mask**	**Contents**
		0	0001_{H}	1 = Characters are bold
		1	0002_{H}	1 = Characters are italic
		2	0004_{H}	1 = Characters are underlined
		3	0008_{H}	1 = Characters are struck out
4	2	Index into PALETTE record (→5.27)		
6	var.	Font name (byte string, 8-bit string length, →2.1)		

Record FONT, BIFF5-BIFF8:

Offset	Size	Contents		
0	2	Height of the font (in 1/20 of a point)		
2	2	Option flags:		
		Bit	**Mask**	**Contents**
		1	0002_{H}	1 = Characters are italic
		3	0008_{H}	1 = Characters are struck out
4	2	Index into PALETTE record (→5.27)		
6	2	Boldness (100-1000). Standard values are 0190_{H} (400) for normal text and $02BC_{H}$ (700) for bold text.		
8	2	FONT_SCRIPT – Subscript or superscript (see above)		
10	1	FONT_UNDERLINE – Underline type (see above)		
11	1	Font family...		
12	1	Character set...		
13	1	Not used		
14	var.	Font name:	BIFF5/BIFF7: Byte string, 8-bit string length (→2.1) BIFF8: Unicode string, 8-bit string length (→2.2)	

5.16 FORMAT

BIFF2	BIFF3	BIFF4	BIFF5	BIFF7	BIFF8
$001E_{H}$	$001E_{H}$	$041E_{H}$	$041E_{H}$	$041E_{H}$	$041E_{H}$

2do

5.17 FORMULA

BIFF2	BIFF3	BIFF4	BIFF5	BIFF7	BIFF8
0006$_H$	0206$_H$	0406$_H$	0006$_H$	0006$_H$	0006$_H$

This record contains the token array and the result of a formula cell.

• Record contents

Record FORMULA, BIFF2:

Offset	Size	Contents
0	2	Index to row
2	2	Index to column
4	3	Cell attributes (→2.9)
7	8	Result of the formula (IEEE floating-point value)
15	1	0 = Do not recalculate, 1 = Recalculate always
16	var.	Formula data (RPN token array, →3)

Record FORMULA, BIFF3-BIFF4:

<table>
<tr><th>Offset</th><th>Size</th><th colspan="3">Contents</th></tr>
<tr><td>0</td><td>2</td><td colspan="3">Index to row</td></tr>
<tr><td>2</td><td>2</td><td colspan="3">Index to column</td></tr>
<tr><td>4</td><td>2</td><td colspan="3">Index to XF record (→5.37)</td></tr>
<tr><td>6</td><td>8</td><td colspan="3">Result of the formula. See below for details.</td></tr>
<tr><td>14</td><td>2</td><td colspan="3">Option flags:</td></tr>
<tr><td></td><td></td><th>Bit</th><th>Mask</th><th>Contents</th></tr>
<tr><td></td><td></td><td>0</td><td>0001$_H$</td><td>1 = Recalculate always</td></tr>
<tr><td></td><td></td><td>1</td><td>0002$_H$</td><td>1 = Calculate on open</td></tr>
<tr><td>16</td><td>var.</td><td colspan="3">Formula data (RPN token array, →3)</td></tr>
</table>

Record FORMULA, BIFF5-BIFF8:

<table>
<tr><th>Offset</th><th>Size</th><th colspan="3">Contents</th></tr>
<tr><td>0</td><td>2</td><td colspan="3">Index to row</td></tr>
<tr><td>2</td><td>2</td><td colspan="3">Index to column</td></tr>
<tr><td>4</td><td>2</td><td colspan="3">Index to XF record (→5.37)</td></tr>
<tr><td>6</td><td>8</td><td colspan="3">Result of the formula. See below for details.</td></tr>
<tr><td>14</td><td>2</td><td colspan="3">Option flags:</td></tr>
<tr><td></td><td></td><th>Bit</th><th>Mask</th><th>Contents</th></tr>
<tr><td></td><td></td><td>0</td><td>0001$_H$</td><td>1 = Recalculate always</td></tr>
<tr><td></td><td></td><td>1</td><td>0002$_H$</td><td>1 = Calculate on open</td></tr>
<tr><td></td><td></td><td>3</td><td>0008$_H$</td><td>1 = Part of a shared formula</td></tr>
<tr><td>16</td><td>4</td><td colspan="3">Not used</td></tr>
<tr><td>20</td><td>var.</td><td colspan="3">Formula data (RPN token array, →3)</td></tr>
</table>

• Result of the formula

Dependent on the type of value the formula returns, the result field has the following format:

Result is a numeric value:

Offset	Size	Contents
0	8	IEEE floating-point value

Result is a string (the string itself follows in a STRING record, →5.34):

Offset	Size	Contents
0	1	00$_H$ (identifier for a string value)
1	5	Not used
6	2	FFFF$_H$

Result is a boolean value:

Offset	Size	Contents
0	1	01$_H$ (identifier for a boolean value)
1	1	Not used
2	1	0 = FALSE, 1 = TRUE
3	3	Not used
6	2	FFFF$_H$

Result is an error value:

Offset	Size	Contents
0	1	02$_H$ (identifier for an error value)
1	1	Not used
2	1	Error code (→2.4)
3	3	Not used
6	2	FFFF$_H$

5.18 HLINK – Hyperlink

BIFF2	BIFF3	BIFF4	BIFF5	BIFF7	BIFF8
---	---	---	---	---	01B8$_H$

In Excel, every cell can contain only one hyperlink. Therefore, the HLINK record refers to one cell address or a cell range where all cells contain the same hyperlink. Every hyperlink can contain a text mark and a description that is shown in the sheet instead of the real link. Text marks are appended behind a link, separated by the „#“ sign. Examples for text marks are www.xyz.org#table1 or C:\example.xls#Sheet1!A1.

Inside of this record strings are stored in several formats. Sometimes occurs the character count, otherwise the character array size (in 16-bit-character arrays the character count is half of the array size). Furthermore some strings are zero-terminated, others not. They are stored either as 16-bit-character arrays or as 8-bit-character arrays, independent of the characters.

5.18.1 Common record contents

Each HLINK record starts with the same data items and continues with special data related to the current type of hyperlink.

Record HLINK, BIFF8:

Offset	Size	Contents
0	2	Index to first row
2	2	Index to last row
4	2	Index to first column
6	2	Index to last column
8	20	Unknown byte sequence: D0$_H$ C9$_H$ EA$_H$ 79$_H$ F9$_H$ BA$_H$ CE$_H$ 11$_H$ 8C$_H$ 82$_H$ 00$_H$ AA$_H$ 00$_H$ 4B$_H$ A9$_H$ 0B$_H$ 02$_H$ 00$_H$ 00$_H$ 00$_H$
28	4	Option flags (see below)
[32]	4	(optional, see option flags) Character count of description text, including trailing zero word (dl)
[36]	2·dl	(optional, see option flags) Character array of description text, no Unicode string header, always 16-bit-characters, zero-terminated
		Special data (→5.18.2 and following)
[var.]	4	(optional, see option flags) Character count of the text mark, including trailing zero word (tl)
[var.]	2·tl	(optional, see option flags) Character array of the text mark without „#“ sign, no Unicode string header, always 16-bit-characters, zero-terminated

The special data parts in the following are described with relative offsets (starting again by zero). The real offset inside of the record data (without header) is either 32 (without description) or 36+2·dl (with description).

• Option flags

The option flags specify the following content of the record.

Bit	Mask	Contents	
0	00000001$_H$	0 = No link extant	1 = File link or URL
1	00000002$_H$	0 = Relative file path	1 = Absolute path or URL
2 and 4	00000014$_H$	0 = No description	1 (both bits) = Description
3	00000008$_H$	0 = No text mark	1 = Text mark
8	00000100$_H$	0 = File link or URL	1 = Network path

5.18.2 Hyperlink to a common URL

These data fields occur for links which are not local files or files in the local network. The lower 9 bits of the option flags field must be `0.000x.xx11`$_2$ (x means optional, depending on hyperlink content). The byte sequence should be used to distinguish an URL from a file link.

Offset	Size	Contents
0	16	Unknown byte sequence, used as URL identifier: E0$_H$ C9$_H$ EA$_H$ 79$_H$ F9$_H$ BA$_H$ CE$_H$ 11$_H$ 8C$_H$ 82$_H$ 00$_H$ AA$_H$ 00$_H$ 4B$_H$ A9$_H$ 0B$_H$
16	4	Size of character array of the URL, including trailing zero word (us). There are us/2-1 characters in the following string.
20	us	Character array of the URL, no Unicode string header, always 16-bit-characters, zero-terminated

5.18.3 Hyperlink to a local file

These data fields are for links to files on local drives. The path of the file can be complete with drive letter (absolute) or relative to the location of the workbook. The lower 9 bits of the option flags field must be $\texttt{0.000x.xxx1}_2$. The byte sequence should be used to distinguish an URL from a file link.

Offset	Size	Contents
0	16	Unknown byte sequence, used as file link identifier: 03_H 03_H 00_H 00_H 00_H 00_H 00_H 00_H $C0_H$ 00_H 00_H 00_H 00_H 00_H 00_H 46_H
16	2	Directory up-level count. Each leading „..\“ in the file link is deleted and inceases this counter.
18	4	Character count of the shortened file path and name, including trailing zero byte (sl)
22	sl	Character array of the shortened file path and name in 8.3-DOS-format. This field can be filled with a long file name too. No Unicode string header, always 8-bit-characters, zero-terminated.
22+sl	24	Unknown byte sequence: FF_H FF_H AD_H DE_H 00_H 00_H 00_H 00_H 00_H 00_H 00_H 00_H 00_H 00_H 00_H 00_H 00_H 00_H 00_H 00_H 00_H 00_H 00_H 00_H
46+sl	4	Size of the following file link field including string length field and additional data field (sz). If sz is zero, nothing will follow (except a text mark).
[50+sl]	4	(optional) Size of character array of the extended file path and name (xl). There are xl/2 characters in the following string.
[54+sl]	2	(optional) Unknown byte sequence: 03_H 00_H
[56+sl]	xl	(optional) Character array of the extended file path and name (xl), no Unicode string header, always 16-bit-characters, not zero-terminated

5.18.4 Hyperlink to a file located in the local network

These data fields are for links to files located in the local network. The lower 9 bits of the option flags field must be $\texttt{1.000x.xx11}_2$.

Offset	Size	Contents
0	4	Character count of the network path and file name, including trailing zero word (fl)
4	2·fl	Character array of the network path and file name, no Unicode string header, always 16-bit-characters, zero-terminated.

5.18.5 Hyperlink to a place in the current workbook

In this case only the text mark field is present (optional with description). Example: The URL „#Sheet2!B1:C2“ refers to the given range in the current workbook. The lower 9 bits of the option flags field must be $\texttt{0.000x.1x00}_2$.

5.19 INTEGER

BIFF2	BIFF3	BIFF4	BIFF5	BIFF7	BIFF8
0002_H	---	---	---	---	---

This record represents a cell that contains an unsigned 16-bit-integer value. If a value cannot be stored as a 16-bit-integer, a NUMBER record (→5.26) must be written. From BIFF3 on this record is replaced by the RK record (→5.29).

Record INTEGER, BIFF2:

Offset	Size	Contents
0	2	Index to row
2	2	Index to column
4	3	Cell attributes (→2.9)
7	2	Unsigned 16-bit-integer value

5.20 IXFE – Index to XF

BIFF2	BIFF3	BIFF4	BIFF5	BIFF7	BIFF8
0044_H	---	---	---	---	---

This record occurs in front of every cell record (for instance BLANK, INTEGER, NUMBER, LABEL, FORMULA) that references to an XF record (→5.37) with an index greater than 62. The XF index field of the cell record consists only of 6 bits. The maximum value 63 is used to indicate a preceding IXFE record containing the real XF index. See →2.9 for more details.

Record IXFE, BIFF2:

Offset	Size	Contents
0	2	Index to XF record (→5.37)

5.21 LABEL

BIFF2	BIFF3	BIFF4	BIFF5	BIFF7	BIFF8
0004_H	0204_H	0204_H	0204_H	0204_H	---

This record represents a cell that contains a string. In BIFF8 it is replaced by the LABELSST record (→5.22).

Record LABEL, BIFF2:

Offset	Size	Contents
0	2	Index to row
2	2	Index to column
4	3	Cell attributes (→2.9)
7	var.	Byte string, 8-bit string length (→2.1)

Record LABEL, BIFF3-BIFF7:

Offset	Size	Contents
0	2	Index to row
2	2	Index to column
4	2	Index to XF record (→5.37)
6	var.	Byte string, 16-bit string length (→2.1)

5.22 LABELSST

BIFF2	BIFF3	BIFF4	BIFF5	BIFF7	BIFF8
---	---	---	---	---	00FD$_{\text{H}}$

This record represents a cell that contains a string. It replaces the LABEL record (→5.21) used in BIFF2-BIFF7. See →4.4 for more information about shared string tables.

Record LABELSST, BIFF8:

Offset	Size	Contents
0	2	Index to row
2	2	Index to column
4	2	Index to XF record (→5.37)
6	4	Index into SST record (→5.33)

5.23 MULBLANK – Multiple BLANK

BIFF2	BIFF3	BIFF4	BIFF5	BIFF7	BIFF8
---	---	---	00BE$_{\text{H}}$	00BE$_{\text{H}}$	00BE$_{\text{H}}$

This record represents a cell range of empty cells. All cells are located in the same row.

Record MULBLANK, BIFF5-BIFF8:

Offset	Size	Contents
0	2	Index to row
2	2	Index to first column (fc)
4	2·(lc-fc+1)	Array of lc-fc+1 16-bit-indexes to XF records (→5.37)
var.	2	Index to last column (lc)

5.24 MULRK – Multiple RK

BIFF2	BIFF3	BIFF4	BIFF5	BIFF7	BIFF8
---	---	---	00BD$_{\text{H}}$	00BD$_{\text{H}}$	00BD$_{\text{H}}$

This record represents a cell range containing RK value cells. All cells are located in the same row.

Record MULRK, BIFF5-BIFF8:

<table>
<tr><th>Offset</th><th>Size</th><th>Contents</th></tr>
<tr><td>0</td><td>2</td><td>Index to row</td></tr>
<tr><td>2</td><td>2</td><td>Index to first column (fc)</td></tr>
<tr><td>4</td><td>6·(lc-fc+1)</td><td>Array of lc-fc+1 XF/RK structures. Each XF/RK contains:
<table>
<tr><th>Offset</th><th>Size</th><th>Contents</th></tr>
<tr><td>0</td><td>2</td><td>Index to XF record (→5.37)</td></tr>
<tr><td>2</td><td>4</td><td>RK value (→2.3)</td></tr>
</table></td></tr>
<tr><td>var.</td><td>2</td><td>Index to last column (lc)</td></tr>
</table>

5.25 NAME

BIFF2	BIFF3	BIFF4	BIFF5	BIFF7	BIFF8
0018$_H$	0218$_H$	0218$_H$	0018$_H$	0018$_H$	0018$_H$

This record contains the name and the token array of an internal defined name.

Record NAME, BIFF2:

Offset	Size	Contents		
0	1	Option flags:		
		Bit	Mask	Contents
		1	02$_H$	1 = Function macro or command macro
		2	04$_H$	1 = Complex function (array formula or user defined)
1	1	If name is function macro or command macro (see option flags above): 01$_H$ = Function macro, 02$_H$ = Command macro		
2	1	Keyboard shortcut		
3	1	Length of the name (character count) (ln)		
4	1	Size of the formula data (RPN token array) (sz)		
5	ln	Character array of the name		
5+ln	sz	Formula data (RPN token array without size field, →3)		
5+ln+sz	1	Duplicate of the formula data size field (sz)		

Record NAME, BIFF3-BIFF4:

Offset	Size	Contents		
0	2	Option flags:		
		Bit	Mask	Contents
		0	0001$_H$	1 = Name is hidden
		1	0002$_H$	1 = Name is a function
		2	0004$_H$	1 = Name is a command
		3	0008$_H$	1 = Function macro or command macro
		4	0010$_H$	1 = Complex function (array formula or user defined)
		5	0020$_H$	1 = Built-in name (see table below)
		11-6	0FC0$_H$	BIFF3: Not used; BIFF4: Index to function group
2	1	Keyboard shortcut		
3	1	Length of the name (character count) (ln)		
4	2	Size of the formula data (RPN token array) (sz)		
6	ln	Character array of the name		
6+ln	sz	Formula data (RPN token array without size field, →3)		

Record NAME, BIFF5/BIFF7:

<table>
<tr><th>Offset</th><th>Size</th><th>Contents</th></tr>
<tr><td>0</td><td>2</td><td>Option flags:
<table>
<tr><th>Bit</th><th>Mask</th><th>Contents</th></tr>
<tr><td>11-0</td><td>0FFF$_H$</td><td>Equal to BIFF4 (see table above)</td></tr>
<tr><td>12</td><td>1000$_H$</td><td>1 = Name contains binary data</td></tr>
</table></td></tr>
<tr><td>2</td><td>1</td><td>Keyboard shortcut</td></tr>
<tr><td>3</td><td>1</td><td>Length of the name (character count) (ln)</td></tr>
<tr><td>4</td><td>2</td><td>Size of the formula data (RPN token array) (sz)</td></tr>
<tr><td>6</td><td>2</td><td>Unused</td></tr>
<tr><td>8</td><td>2</td><td>0 = Global name, otherwise index to sheet (one-based)</td></tr>
<tr><td>10</td><td>1</td><td>Length of menu text (character count) (lm)</td></tr>
<tr><td>11</td><td>1</td><td>Length of description text (character count) (ld)</td></tr>
<tr><td>12</td><td>1</td><td>Length of help topic text (character count) (lh)</td></tr>
<tr><td>13</td><td>1</td><td>Length of status bar text (character count) (ls)</td></tr>
<tr><td>14</td><td>ln</td><td>Character array of the name</td></tr>
<tr><td>14+ln</td><td>sz</td><td>Formula data (RPN token array without size field, →3)</td></tr>
<tr><td>14+ln+sz</td><td>lm</td><td>Character array of menu text</td></tr>
<tr><td>var.</td><td>ld</td><td>Character array of description text</td></tr>
<tr><td>var.</td><td>lh</td><td>Character array of help topic text</td></tr>
<tr><td>var.</td><td>ls</td><td>Character array of status bar text</td></tr>
</table>

Record NAME, BIFF8:

Offset	Size	Contents
0	2	Option flags: Equal to BIFF5/BIFF7 (see table above)
2	1	Keyboard shortcut
3	1	Length of the name (character count)
4	2	Size of the formula data (RPN token array) (sz)
6	2	Unused
8	2	0 = Global name, otherwise index to sheet (one-based)
10	1	Length of menu text (character count)
11	1	Length of description text (character count)
12	1	Length of help topic text (character count)
13	1	Length of status bar text (character count)
14	var.	Name (Unicode string without length field, →2.2)
var.	sz	Formula data (RPN token array without size field, →3)
var.	var.	Menu text (Unicode string without length field, →2.2)
var.	var.	Description text (Unicode string without length field, →2.2)
var.	var.	Help topic text (Unicode string without length field, →2.2)
var.	var.	Status bar text (Unicode string without length field, →2.2)

- **Built-in names**

From BIFF3 on only an index to a built-in names is stored. If bit 5 of the option flags field is set, the name string contains only one character with this index.

Built-in index	Built-In name
00_H	Consolidate_Area
01_H	Auto_Open
02_H	Auto_Close
03_H	Extract
04_H	Database
05_H	Criteria
06_H	Print_Area
07_H	Pint_Titles
08_H	Recorder
09_H	Data_Form
$0A_H$	Auto_Activate
$0B_H$	Auto_Deactivate
$0C_H$	Sheet_Title
$0D_H$	BIFF3-BIFF4: Not used; BIFF5-BIFF8: Autofilter

5.26 NUMBER

BIFF2	BIFF3	BIFF4	BIFF5	BIFF7	BIFF8
0003_H	0203_H	0203_H	0203_H	0203_H	0203_H

This record represents a cell that contains a floating-point value.

Record NUMBER, BIFF2:

Offset	Size	Contents
0	2	Index to row
2	2	Index to column
4	3	Cell attributes (→2.9)
7	8	IEEE floating-point value

Record NUMBER, BIFF3-BIFF8:

Offset	Size	Contents
0	2	Index to row
2	2	Index to column
4	2	Index to XF record (→5.37)
6	8	IEEE floating-point value

5.27 PALETTE

BIFF2	BIFF3	BIFF4	BIFF5	BIFF7	BIFF8
---	0092$_{H}$	0092$_{H}$	0092$_{H}$	0092$_{H}$	0092$_{H}$

This record contains the definition of all colors available for cell and object formatting.

Record PALETTE, BIFF3-BIFF8:

Offset	Size	Contents		
0	2	Number of following colors (nm)		
2	4·nm	List of nm colors. Each color contains:		
		Offset	**Size**	**Contents**
		0	1	Red component of the color
		1	1	Green component of the color
		2	1	Blue component of the color
		3	1	Not used

5.28 PASSWORD

BIFF2	BIFF3	BIFF4	BIFF5	BIFF7	BIFF8
0013$_{H}$	0013$_{H}$	0013$_{H}$	0013$_{H}$	0013$_{H}$	0013$_{H}$

This record stores a 16-bit hash value for a sheet or workbook protection password.

Offset	Size	Contents
0	2	16-bit hash value of the password

This is the algorithm to create the hash value from a given password:

- The ASCII values of all characters are rotated left with a number of digits depending on the character position (first character is rotated left 1 bit, second character 2 bits, and so on). There is a space of 15 bits available for rotation (bit 15 jumps to bit 0, bit 16 jumps to bit 1 and so on).
- All rotated characters are combined using XOR operation.
- The number of characters is added using XOR operation.
- The constant CE4B$_{H}$ is added using XOR operation.

Example: The password is „abcdefghij“ (10 characters).

Character	ASCII	Shifted	Rotated
a	61$_{H}$	000000C2$_{H}$	00C2$_{H}$
b	62$_{H}$	00000188$_{H}$	0188$_{H}$
c	63$_{H}$	00000318$_{H}$	0318$_{H}$
d	64$_{H}$	00000640$_{H}$	0640$_{H}$
e	65$_{H}$	00000CA0$_{H}$	0CA0$_{H}$
f	66$_{H}$	00001980$_{H}$	1980$_{H}$
g	67$_{H}$	00003380$_{H}$	3380$_{H}$
h	68$_{H}$	00006800$_{H}$	6800$_{H}$
i	69$_{H}$	0000D200$_{H}$	5201$_{H}$
j	6A$_{H}$	0001A800$_{H}$	2803$_{H}$

All the rotated values and the number of characters 000A$_{H}$ and the constant CE4B$_{H}$ result in the hash value FEF1$_{H}$.

5.29 RK

BIFF2	BIFF3	BIFF4	BIFF5	BIFF7	BIFF8
---	027E$_H$	027E$_H$	027E$_H$	027E$_H$	027E$_H$

This record represents a cell that contains an RK value (encoded integer or floating-point value). If a floating-point value cannot be encoded to an RK value, a NUMBER record (→5.26) must be written. This record replaces the record INTEGER (→5.19) written in BIFF2.

Record RK, BIFF3-BIFF8:

Offset	Size	Contents
0	2	Index to row
2	2	Index to column
4	2	Index to XF record (→5.37)
6	4	RK value (→2.3)

5.30 SCREENTIP

BIFF2	BIFF3	BIFF4	BIFF5	BIFF7	BIFF8
---	---	---	---	---	0800$_H$

This record contains the cell range and text for a screen tip. It occurs in conjunction with the HLINK record for hyperlinks (→5.18).

Record SCREENTIP, BIFF8:

Offset	Size	Contents
0	2	0800$_H$ (repeated record ID)
2	2	Index to first row
4	2	Index to last row
6	2	Index to first column
8	2	Index to last column
10	var.	Character array of the screen tip, no Unicode string header, always 16-bit-characters, zero-terminated

5.31 SHEETHDR

BIFF2	BIFF3	BIFF4	BIFF5	BIFF7	BIFF8
---	---	008F$_H$	---	---	---

This record occurs only in BIFF4 workbook files. It precedes a substream for a sheet (delimited by a BOF and a EOF record) and contains the byte length of the substream and the sheet name. Adding the substream length to the stream position of the following BOF record gives the position of the next SHEETHDR record. See →4.2 for details about the BIFF4 workbook stream.

Record SHEETHDR, BIFF4:

Offset	Size	Contents
0	4	Byte length of the following sheet substream
4	var.	Name of the sheet (byte string, 8-bit string length, →2.1)

5.32 SHEETSOFFSET

BIFF2	BIFF3	BIFF4	BIFF5	BIFF7	BIFF8
---	---	$008E_H$	---	---	---

This record occurs only in BIFF4 workbook files. It is located in the workbook globals section and contains the stream position of the first SHEETHDR record (→5.31). See →4.2 for details about the BIFF4 workbook stream.

Record SHEETSOFFSET, BIFF4:

Offset	Size	Contents
0	4	Stream position of the first SHEETHDR record (→5.31)

5.33 SST – Shared String Table

BIFF2	BIFF3	BIFF4	BIFF5	BIFF7	BIFF8
---	---	---	---	---	$00FC_H$

This record contains a list of all strings used anywhere in the workbook. Each string occurs only one time. The workbook uses indexes into the list to reference to strings. See →4.4 for more information.

Record SST, BIFF8:

Offset	Size	Contents
0	4	Total number of strings in the workbook (see below)
4	4	Number of following strings (nm)
8	var.	List of nm Unicode strings, 16-bit string length (→2.2)

The first field of the SST record counts the total occurrence of strings in the workbook. For instance, the string „AAA“ is used 3 times and the string „BBB“ is used 2 times. The first field contains 5 and the second field contains 2, followed by the two strings.

5.34 STRING

BIFF2	BIFF3	BIFF4	BIFF5	BIFF7	BIFF8
0007_H	0207_H	0207_H	0207_H	0207_H	0207_H

This record stores the result of a string formula. It occurs directly after a string formula (→5.17).

Record STRING, BIFF2:

Offset	Size	Contents
0	var.	Byte string, 8-bit string length (→2.1)

Record STRING, BIFF3-BIFF7:

Offset	Size	Contents
0	var.	Byte string, 16-bit string length (→2.1)

In BIFF8 files the whole record is omitted, if the result is an empty string.

Record STRING, BIFF8:

Offset	Size	Contents
0	var.	Unicode string with at least 1 character, 16-bit string length (→2.2)

5.35 SUPBOOK – External Workbook

BIFF2	BIFF3	BIFF4	BIFF5	BIFF7	BIFF8
---	---	---	---	---	01AE$_H$

This record mainly stores the name of an external document and a list of sheet names inside of this document. Furthermore it is used to store names of documents for DDE and OLE object links or to indicate an internal 3D reference or an AddIn function. See →4.5.3 for details about external references in BIFF8.

5.35.1 External references

A SUPBOOK record for external references stores the name of the document and a list of sheet names.

Record SUPBOOK for external references, BIFF8:

Offset	Size	Contents
0	2	Number of sheet names (nm)
2	var.	Encoded document name without sheet name (→2.6.1). Unicode string, 16-bit string length (→2.2).
var.	var.	List of nm sheet names (Unicode strings with 16-bit string length, →2.2)

5.35.2 Internal references

In each file occurs a SUPBOOK that is used for internal 3D references. It stores the number of sheets of the own document.

Record SUPBOOK for 3D references, BIFF8:

Offset	Size	Contents
0	2	Number of sheets in this document
2	2	0401$_H$

5.35.3 AddIn functions

AddIn function names are stored in EXTERNNAME records following this SUPBOOK record.

Record SUPBOOK for AddIn functions, BIFF8:

Offset	Size	Contents
0	2	0001$_H$
2	2	3A01$_H$

5.35.4 DDE links, OLE object links

The SUPBOOK record of a DDE link or an OLE object link contains the name of the server application (DDE) or the class name (OLE) and the name of a source document. These names are encoded in one string.

Record SUPBOOK for DDE links and OLE object links, BIFF8:

Offset	Size	Contents
0	2	0000$_H$
2	var.	Encoded source document name (→2.6.2). Unicode string, 16-bit string length (→2.2).

5.36 XCT – CRN Count

BIFF2	BIFF3	BIFF4	BIFF5	BIFF7	BIFF8
---	0059$_H$	0059$_H$	0059$_H$	0059$_H$	0059$_H$

This record stores the number of immediately following CRN records. These records are used to store the cell contents of external references. See →4.5 for details about of external references.

Record XCT, BIFF3-BIFF7:

Offset	Size	Contents
0	2	Number of following CRN records

Record XCT, BIFF8:

Offset	Size	Contents
0	2	Number of following CRN records
2	2	Index to sheet table of the involved SUPBOOK record (→5.35)

5.37 XF – Extended Format

BIFF2	BIFF3	BIFF4	BIFF5	BIFF7	BIFF8
0043$_H$	0243$_H$	0443$_H$	00E0$_H$	00E0$_H$	00E0$_H$

This record contains formatting information for cells, rows, columns or styles.

From BIFF3 on, some of the elements occur unchanged in every BIFF version. These elements are described in the following using a specific name for each element. In the description of the record structure the names are used to reference to these tables.

5.37.1 XF substructures

• XF_TYPE_PROT – XF type and cell protection (3 bits), BIFF3-BIFF8

These 3 bits are part of a specific data byte.

Bit	Mask	Contents
0	01$_H$	1 = Cell is locked
1	02$_H$	1 = Formula is hidden
2	04$_H$	0 = Cell XF; 1 = Style XF

• XF_USED_ATTRIB – Attributes used from parent style XF (1 byte), BIFF3-BIFF8

In this byte, each bit describes the validity of a specific attribute. In cell XFs an unset bit means the attribute of the parent style XF is used, a set bit means the attribute of this XF is used. In style XFs an unset bit means the attribute setting is valid, a set bit means the attribute should be ignored.

Bit	Mask	Contents
2	04$_H$	Flag for number format
3	08$_H$	Flag for font
4	10$_H$	Flag for alignment, text wrap and rotation
5	20$_H$	Flag for border lines
6	40$_H$	Flag for background area style
7	80$_H$	Flag for cell protection (cell locked and formula hidden)

• XF_HOR_ALIGN – Horizontal alignment (3 bits), BIFF2-BIFF8

The horizontal alignment consists of 3 bits and is part of a specific data byte.

Value	Horizontal alignment
00_H	General
01_H	Left
02_H	Centered
03_H	Right
04_H	Filled
05_H	Justified (BIFF3-BIFF8)
06_H	Centered across selection (BIFF3-BIFF8)

• XF_VERT_ALIGN – Vertical alignment (2 bits), BIFF4-BIFF8

The vertical alignment consists of 2 bits and is part of a specific data byte. Vertical alignment is not available in BIFF2 and BIFF3.

Value	Vertical alignment
00_H	Left
01_H	Centered
02_H	Right
03_H	Justified (BIFF5-BIFF8)

• XF_ORIENTATION – Text orientation (2 bits), BIFF4-BIFF7

In the BIFF versions BIFF4-BIFF7, text can be rotated in steps of 90-degrees or stacked. The orientation mode consists of 2 bits and is part of a specific data byte. In BIFF8 a rotation angle occurs instead of these flags.

Value	Text orientation
00_H	Not rotated
01_H	Letters are stacked top-to-bottom, but not rotated
02_H	Text is rotated 90 degrees counterclockwise
03_H	Text is rotated 90 degrees clockwise

• XF_ROTATION – Text rotation angle (1 byte), BIFF8

Value	Text rotation
0	Not rotated
1-90	1 deg. - 90 deg. counterclockwise
91-180	1 deg. - 90 deg. clockwise
255	Letters are stacked top-to-bottom, but not rotated

• XF_BORDER_34 – Cell border style (4 bytes), BIFF3-BIFF4

Cell borders contain a line style and a line color for each line of the border.

Bit	Mask	Contents
2-0	00000007$_H$	Top line style (→2.7)
7-3	000000F8$_H$	Index into PALETTE record for top line color (→5.27)
10-8	00000700$_H$	Left line style
15-11	0000F800$_H$	Index into PALETTE record for left line color
18-16	00070000$_H$	Bottom line style
23-19	00F80000$_H$	Index into PALETTE record for bottom line color
26-24	07000000$_H$	Right line style
31-27	F8000000$_H$	Index into PALETTE record for right line color

• XF_AREA_34 – Cell background area style (2 bytes), BIFF3-BIFF4

A cell background area style contains an area pattern and a foreground and background color.

Bit	Mask	Contents
5-0	003F$_H$	Fill pattern (→2.8)
10-6	07C0$_H$	Index into PALETTE record for pattern foreground (→5.27)
15-11	F800$_H$	Index into PALETTE record for pattern background

5.37.2 XF record contents

Record XF, BIFF2:

Offset	Size	Contents		
0	1	Index to FONT record (→5.15)		
1	1	Not used		
2	1	**Bit**	**Mask**	**Contents**
		5-0	3F$_H$	Index to FORMAT record (→5.16)
		6	40$_H$	1 = Cell is locked
		7	80$_H$	1 = Formula is hidden
3	1	**Bit**	**Mask**	**Contents**
		2-0	07$_H$	XF_HOR_ALIGN – Horizontal alignment (see above)
		3	08$_H$	1 = Cell has left black border
		4	10$_H$	1 = Cell has right black border
		5	20$_H$	1 = Cell has top black border
		6	40$_H$	1 = Cell has bottom black border
		7	80$_H$	1 = Cell has shaded background

Record XF, BIFF3:

Offset	Size	Contents		
0	1	Index to FONT record (→5.15)		
1	1	Index to FORMAT record (→5.16)		
2	1	XF_TYPE_PROT – XF type and cell protection (see above)		
3	1	XF_USED_ATTRIB – Used attributes (see above)		
4	2	**Bit**	**Mask**	**Contents**
		2-0	0007$_H$	XF_HOR_ALIGN – Horizontal alignment (see above)
		3	0008$_H$	1 = Text is wrapped at right border
		15-4	FFF0$_H$	Index to parent style XF (always FFF$_H$ in style XFs)
6	2	XF_AREA_34 – Cell background area (see above)		
8	4	XF_BORDER_34 – Cell border lines (see above)		

Record XF, BIFF4:

Offset	Size	Contents		
0	1	Index to FONT record (→5.15)		
1	1	Index to FORMAT record (→5.16)		
2	2	**Bit**	**Mask**	**Contents**
		2-0	0007$_H$	XF_TYPE_PROT – XF type, cell protection (see above)
		15-4	FFF0$_H$	Index to parent style XF (always FFF$_H$ in style XFs)
4	1	**Bit**	**Mask**	**Contents**
		2-0	07$_H$	XF_HOR_ALIGN – Horizontal alignment (see above)
		3	08$_H$	1 = Text is wrapped at right border
		5-4	30$_H$	XF_VERT_ALIGN – Vertical alignment (see above)
		7-6	C0$_H$	XF_ORIENTATION – Text orientation (see above)
5	1	XF_USED_ATTRIB – Used attributes (see above)		
6	2	XF_AREA_34 – Cell background area (see above)		
8	4	XF_BORDER_34 – Cell border lines (see above)		

Record XF, BIFF5/BIFF7:

Offset	Size	Contents		
0	2	Index to FONT record (→5.15)		
2	2	Index to FORMAT record (→5.16)		
4	2	**Bit**	**Mask**	**Contents**
		2-0	0007$_H$	XF_TYPE_PROT – XF type, cell protection (see above)
		15-4	FFF0$_H$	Index to parent style XF (always FFF$_H$ in style XFs)
6	1	**Bit**	**Mask**	**Contents**
		2-0	07$_H$	XF_HOR_ALIGN – Horizontal alignment (see above)
		3	08$_H$	1 = Text is wrapped at right border
		5-4	30$_H$	XF_VERT_ALIGN – Vertical alignment (see above)
7	1	**Bit**	**Mask**	**Contents**
		1-0	03$_H$	XF_ORIENTATION – Text orientation (see above)
		7-2	FC$_H$	XF_USED_ATTRIB – Used attributes (see above)
8	4	Cell border lines and background area:		
		Bit	**Mask**	**Contents**
		6-0	0000007F$_H$	Index into PALETTE for pattern foreground
		13-7	00003F8$_H$	Index into PALETTE for pattern background
		21-16	003F0000$_H$	Fill pattern (→2.8)
		24-22	01C00000$_H$	Bottom line style (→2.7)
		31-25	FE000000$_H$	Index into PALETTE for bottom line color
12	4	**Bit**	**Mask**	**Contents**
		2-0	00000007$_H$	Top line style (→2.7)
		5-3	00000038$_H$	Left line style
		8-6	000001C0$_H$	Right line style
		15-9	0000FE00$_H$	Index into PALETTE for top line color
		22-16	007F0000$_H$	Index into PALETTE for left line color
		29-23	3F800000$_H$	Index into PALETTE for right line color

Record XF, BIFF8:

Offset	Size	Bit	Mask	Contents
0	2			Index to FONT record (→5.15)
2	2			Index to FORMAT record (→5.16)
4	2	2-0	0007$_H$	XF_TYPE_PROT – XF type, cell protection (see above)
		15-4	FFF0$_H$	Index to parent style XF (always FFF$_H$ in style XFs)
6	1	2-0	07$_H$	XF_HOR_ALIGN – Horizontal alignment (see above)
		3	08$_H$	1 = Text is wrapped at right border
		5-4	30$_H$	XF_VERT_ALIGN – Vertical alignment (see above)
7	1			XF_ROTATION: Text rotation angle (see above)
8	1	3-0	0F$_H$	Indent level
		4	10$_H$	1 = Shrink content to fit into cell
		5	20$_H$	1 = Cell is part of a merged range
9	1	7-2	FC$_H$	XF_USED_ATTRIB – Used attributes (see above)
10	4			Cell border lines and background area:
		3-0	0000000F$_H$	Left line style (→2.7)
		7-4	000000F0$_H$	Right line style
		11-8	00000F00$_H$	Top line style
		15-12	0000F000$_H$	Bottom line style
		22-16	007F0000$_H$	Index into PALETTE for left line color
		29-23	3F800000$_H$	Index into PALETTE for right line color
		30	40000000$_H$	1 = Diagonal line from top left to right bottom
		31	80000000$_H$	1 = Diagonal line from bottom left to right top
14	4	6-0	0000007F$_H$	Index into PALETTE for top line color
		13-7	00003F80$_H$	Index into PALETTE for bottom line color
		20-14	001FC000$_H$	Index into PALETTE for diagonal line color
		24-21	01E00000$_H$	Diagonal line style (→2.7)
		31-26	FC000000$_H$	Fill pattern (→2.8)
18	2	6-0	007F$_H$	Index into PALETTE for pattern foreground
		13-7	3F80$_H$	Index into PALETTE for pattern background

6 Drawing Objects, Escher Layer

2do

7 Charts

2do

8 PivotTables

2do

9 Change Tracking

2do

.EXE Executable-File Header Format (3.1)

An executable (.EXE) file for the Microsoft Windows operating system contains a combination of code and data or a combination of code, data, and resources. The executable file also contains two headers: an MS-DOS header and a Windows header. The next two sections describe these headers; the third section describes the code and data contained in a Windows executable file.

MS-DOS Header

The MS-DOS (old-style) executable-file header contains four distinct parts: a collection of header information (such as the signature word, the file size, and so on), a reserved section, a pointer to a Windows header (if one exists), and a stub program.

The following illustration shows the MS-DOS executable-file header:

If the word value at offset 18h is 40h or greater, the word value at 3Ch is typically an offset to a Windows header. Applications must verify this for each executable-file header being tested, because a few applications have a different header style.

MS-DOS uses the stub program to display a message if Windows has not been loaded when the user attempts to run a program.

For more information about the MS-DOS executable-file header, see the Microsoft MS-DOS Programmer's Reference (Redmond, Washington: Microsoft Press, 1991).

Windows Header

The Windows (new-style) executable-file header contains information that the loader requires for segmented executable files.

This information includes the linker version number, data specified by the linker, data specified by the resource compiler, tables of segment data, tables of resource data, and so on. The following illustration shows the Windows executable-file header:

The following sections describe the entries in the Windows executable-file header.

Information Block

The information block in the Windows header contains the linker version number, the lengths of various tables that further describe the executable file, the offsets from the beginning of the header to the beginning of these tables, the heap and stack sizes, and so on. The following list summarizes the contents of the header information block (the locations are relative to the beginning of the block):

Location   Description

00h   Specifies the signature word. The low byte contains "N" (4Eh) and the high byte contains "E" (45h).
02h   Specifies the linker version number.
03h   Specifies the linker revision number.
04h   Specifies the offset to the entry table (relative to the beginning of the header).
06h   Specifies the length of the entry table, in bytes.
08h   Reserved.
0Ch   Specifies flags that describe the contents of the executable file. This value can be one or more of the following bits:

Bit   Meaning

0   The linker sets this bit if the executable-file format is SINGLEDATA. An executable file with this format contains one data segment. This bit is set if the file is a dynamic-link library (DLL).
1   The linker sets this bit if the executable-file format is MULTIPLEDATA. An executable file with this format contains multiple data segments. This bit is set if the file is a Windows application.

If neither bit 0 nor bit 1 is set, the executable-file format is NOAUTODATA. An executable file with this format does not contain an automatic data segment.

2   Reserved.

3 Reserved.
8 Reserved.
9 Reserved.
11 If this bit is set, the first segment in the executable file contains code that loads the application.
13 If this bit is set, the linker detects errors at link time but still creates an executable file.

14 Reserved.
15 If this bit is set, the executable file is a library module.

If bit 15 is set, the CS:IP registers point to an initialization procedure called with the value in the AX register equal to the module handle. The initialization procedure must execute a far return to the caller. If the procedure is successful, the value in AX is nonzero. Otherwise, the value in AX is zero.
The value in the DS register is set to the library's data segment if SINGLEDATA is set. Otherwise, DS is set to the data segment of the application that loads the library.

0Eh Specifies the automatic data segment number. (0Eh is zero if the SINGLEDATA and MULTIPLEDATA bits are cleared.)
10h Specifies the initial size, in bytes, of the local heap. This value is zero if there is no local allocation.
12h Specifies the initial size, in bytes, of the stack. This value is zero if the SS register value does not equal the DS register value.
14h Specifies the segment:offset value of CS:IP.
18h Specifies the segment:offset value of SS:SP.

The value specified in SS is an index to the module's segment table. The first entry in the segment table corresponds to segment number 1.
If SS addresses the automatic data segment and SP is zero, SP is set to the address obtained by adding the size of the automatic data segment to the size of the stack.

1Ch Specifies the number of entries in the segment table.
1Eh Specifies the number of entries in the module-reference table.
20h Specifies the number of bytes in the nonresident-name table.
22h Specifies a relative offset from the beginning of the Windows header to the beginning of the segment table.
24h Specifies a relative offset from the beginning of the Windows header to the beginning of the resource table.
26h Specifies a relative offset from the beginning of the Windows header to the beginning of the resident-name table.
28h Specifies a relative offset from the beginning of the Windows header to the beginning of the module-reference table.
2Ah Specifies a relative offset from the beginning of the Windows header to the beginning of the imported-name table.
2Ch Specifies a relative offset from the beginning of the file to the beginning of the nonresident-name table.

30h Specifies the number of movable entry points.
32h Specifies a shift count that is used to align the logical sector. This count is log2 of the segment sector size. It is typically 4, although the default count is 9. (This value corresponds to the /alignment [/a] linker switch. When the linker command line contains /a:16, the shift count is 4. When the linker command line contains /a:512, the shift count is 9.)
34h Specifies the number of resource segments.
36h Specifies the target operating system, depending on which bits are set:

Bit	Meaning
0	Operating system format is unknown.
1	Reserved.
2	Operating system is Microsoft Windows.
3	Reserved.
4	Reserved.

37h Specifies additional information about the executable file. It can be one or more of the following values:

Bit Meaning

1 If this bit is set, the executable file contains a Windows 2.x application that runs in version 3.x protected mode.
2 If this bit is set, the executable file contains a Windows 2.x application that supports proportional fonts.
3 If this bit is set, the executable file contains a fast-load area.

38h Specifies the offset, in sectors, to the beginning of the fast-load area. (Only Windows uses this value.)
3Ah Specifies the length, in sectors, of the fast-load area. (Only Windows uses this value.)
3Ch Reserved.
3Eh Specifies the expected version number for Windows. (Only Windows uses this value.)

Segment Table

The segment table contains information that describes each segment in an executable file. This information includes the segment length, segment type, and segment-relocation data. The following list summarizes the values found in the segment table (the locations are relative to the beginning of each entry):

Location Description

00h Specifies the offset, in sectors, to the segment data (relative to the beginning of the file). A value of zero means no data exists.
02h Specifies the length, in bytes, of the segment, in the file. A value of zero indicates that the segment length is 64K, unless the selector offset is also zero.
04h Specifies flags that describe the contents of the executable file. This value can be one or more of the following:

Bit Meaning

0 If this bit is set, the segment is a data segment. Otherwise, the segment is a code segment.
1 If this bit is set, the loader has allocated memory for the segment.
2 If this bit is set, the segment is loaded.
3 Reserved.
4 If this bit is set, the segment type is MOVABLE. Otherwise, the segment type is FIXED.
5 If this bit is set, the segment type is PURE or SHAREABLE. Otherwise, the segment type is IMPURE or NONSHAREABLE.
6 If this bit is set, the segment type is PRELOAD. Otherwise, the segment type is LOADONCALL.
7 If this bit is set and the segment is a code segment, the segment type is EXECUTEONLY. If this bit is set and the segment is a data segment, the segment type is READONLY.

8 If this bit is set, the segment contains relocation data.
9 Reserved.
10 Reserved.
11 Reserved.
12 If this bit is set, the segment is discardable.
13 Reserved.
14 Reserved.
15 Reserved.

06h Specifies the minimum allocation size of the segment, in bytes. A value of zero indicates that the minimum allocation size is 64K.

Resource Table

The resource table describes and identifies the location of each resource in the executable file. The table has the following form:

```
WORD     rscAlignShift;
TYPEINFO rscTypes[];
```

```
WORD     rscEndTypes;
BYTE     rscResourceNames[];
BYTE     rscEndNames;
```

Following are the members in the resource table:

rscAlignShift   Specifies the alignment shift count for resource data. When the shift count is used as an exponent of 2, the resulting value specifies the factor, in bytes, for computing the location of a resource in the executable file.

rscTypes   Specifies an array of TYPEINFO structures containing information about resource types. There must be one TYPEINFO structure for each type of resource in the executable file.

rscEndTypes   Specifies the end of the resource type definitions. This member must be zero.

rscResourceNames   Specifies the names (if any) associated with the resources in this table. Each name is stored as consecutive bytes; the first byte specifies the number of characters in the name.

rscEndNames   Specifies the end of the resource names and the end of the resource table. This member must be zero.

Type Information

The TYPEINFO structure has the following form:

```
typedef struct _TYPEINFO {
    WORD        rtTypeID;
    WORD        rtResourceCount;
    DWORD       rtReserved;
    NAMEINFO    rtNameInfo[];
} TYPEINFO;
```

Following are the members in the TYPEINFO structure:

rtTypeID   Specifies the type identifier of the resource. This integer value is either a resource-type value or an offset to a resource-type name. If the high bit in this member is set (0x8000), the value is one of the following resource-type values:

Value	Resource type
RT_ACCELERATOR	Accelerator table
RT_BITMAP	Bitmap
RT_CURSOR	Cursor
RT_DIALOG	Dialog box
RT_FONT	Font component
RT_FONTDIR	Font directory
RT_GROUP_CURSOR	Cursor directory
RT_GROUP_ICON	Icon directory
RT_ICON	Icon
RT_MENU	Menu
RT_RCDATA	Resource data
RT_STRING	String table

If the high bit of the value in this member is not set, the value represents an offset, in bytes relative to the beginning of the resource table, to a name in the rscResourceNames member.

rtResourceCount Specifies the number of resources of this type in the executable file.

rtReserved   Reserved.

rtNameInfo   Specifies an array of NAMEINFO structures containing information about individual resources. The rtResourceCount member specifies the number of structures in the array.

Name Information

The NAMEINFO structure has the following form:

```
typedef struct _NAMEINFO {
    WORD rnOffset;
    WORD rnLength;
    WORD rnFlags;
    WORD rnID;
    WORD rnHandle;
    WORD rnUsage;
} NAMEINFO;
```

Following are the members in the NAMEINFO structure:

rnOffset Specifies an offset to the contents of the resource data (relative to the beginning of the file). The offset is in terms of alignment units specified by the rscAlignShift member at the beginning of the resource table.
rnLength Specifies the resource length, in bytes.
rnFlags Specifies whether the resource is fixed, preloaded, or shareable. This member can be one or more of the following values:

Value	Meaning
0x0010	Resource is movable (MOVEABLE). Otherwise, it is fixed.
0x0020	Resource can be shared (PURE).
0x0040	Resource is preloaded (PRELOAD). Otherwise, it is loaded on demand.

rnID Specifies or points to the resource identifier. If the identifier is an integer, the high bit is set (8000h). Otherwise, it is an offset to a resource string, relative to the beginning of the resource table.
rnHandle Reserved.
rnUsage Reserved.

Resident-Name Table

The resident-name table contains strings that identify exported functions in the executable file. As the name implies, these strings are resident in system memory and are never discarded. The resident-name strings are case-sensitive and are not null-terminated. The following list summarizes the values found in the resident-name table (the locations are relative to the beginning of each entry):

Location	Description
00h	Specifies the length of a string. If there are no more strings in the table, this value is zero.
01h - xxh	Specifies the resident-name text. This string is case-sensitive and is not null-terminated.
xxh + 01h	Specifies an ordinal number that identifies the string. This number is an index into the entry table.

The first string in the resident-name table is the module name.

Module-Reference Table

The module-reference table contains offsets for module names stored in the imported-name table. Each entry in this table is 2 bytes long.

Imported-Name Table

The imported-name table contains the names of modules that the executable file imports. Each entry contains two parts: a single byte that specifies the length of the string and the string itself. The strings in this table are not null-terminated.

Entry Table

The entry table contains bundles of entry points from the executable file (the linker generates each bundle). The numbering system for these ordinal values is 1-based--that is, the ordinal value corresponding to the first entry point is 1.

The linker generates the densest possible bundles under the restriction that it cannot reorder the entry points. This restriction is

necessary because other executable files may refer to entry points within a given bundle by their ordinal values.
The entry-table data is organized by bundle, each of which begins with a 2-byte header. The first byte of the header specifies the number of entries in the bundle (a value of 00h designates the end of the table). The second byte specifies whether the corresponding segment is movable or fixed. If the value in this byte is 0FFh, the segment is movable. If the value in this byte is 0FEh, the entry does not refer to a segment but refers, instead, to a constant defined within the module. If the value in this byte is neither 0FFh nor 0FEh, it is a segment index.

For movable segments, each entry consists of 6 bytes and has the following form:

Location   Description

00h   Specifies a byte value. This value can be a combination of the following bits:

Bit(s) Meaning

0   If this bit is set, the entry is exported.
1   If this bit is set, the segment uses a global (shared) data segment.
3-7   If the executable file contains code that performs ring transitions, these bits specify the number of words that compose the stack. At the time of the ring transition, these words must be copied from one ring to the other.

01h   Specifies an int 3fh instruction.
03h   Specifies the segment number.
04h   Specifies the segment offset.

For fixed segments, each entry consists of 3 bytes and has the following form:

Location   Description

00h   Specifies a byte value. This value can be a combination of the following bits:

Bit(s) Meaning

0   If this bit is set, the entry is exported.
1   If this bit is set, the entry uses a global (shared) data segment. (This may be set only for SINGLEDATA library modules.)
3-7   If the executable file contains code that performs ring transitions, these bits specify the number of words that compose the stack. At the time of the ring transition, these words must be copied from one ring to the other.

01h   Specifies an offset.

Nonresident-Name Table

The nonresident-name table contains strings that identify exported functions in the executable file. As the name implies, these strings are not always resident in system memory and are discardable. The nonresident-name strings are case-sensitive; they are not null-terminated. The following list summarizes the values found in the nonresident-name table (the specified locations are relative to the beginning of each entry):

Location   Description

00h   Specifies the length, in bytes, of a string. If this byte is 00h, there are no more strings in the table.
01h - xxh   Specifies the nonresident-name text. This string is case-sensitive and is not null-terminated.
xx + 01h   Specifies an ordinal number that is an index to the entry table.

The first name that appears in the nonresident-name table is the module description string (which was specified in the module-definition file).

Code Segments and Relocation Data

Code and data segments follow the Windows header. Some of the code segments may contain calls to functions in other segments and may, therefore, require relocation data to resolve those references. This relocation data is stored in a relocation table that appears immediately after the code or data in the segment. The first 2 bytes in this table specify the number of relocation items the table contains. A relocation item is a collection of bytes specifying the following information:

Address type (segment only, offset only, segment and offset)

Relocation type (internal reference, imported ordinal, imported name)

Segment number or ordinal identifier (for internal references)

Reference-table index or function ordinal number (for imported ordinals)

Reference-table index or name-table offset (for imported names)

Each relocation item contains 8 bytes of data, the first byte of which specifies one of the following relocation-address types:

Value	Meaning
0	Low byte at the specified offset
2	16-bit selector
3	32-bit pointer
5	16-bit offset
11	48-bit pointer
13	32-bit offset

The second byte specifies one of the following relocation types:

Value	Meaning
0	Internal reference
1	Imported ordinal
2	Imported name
3	OSFIXUP

The third and fourth bytes specify the offset of the relocation item within the segment.

If the relocation type is imported ordinal, the fifth and sixth bytes specify an index to a module's reference table and the seventh and eighth bytes specify a function ordinal value.

If the relocation type is imported name, the fifth and sixth bytes specify an index to a module's reference table and the seventh and eighth bytes specify an offset to an imported-name table.

If the relocation type is internal reference and the segment is fixed, the fifth byte specifies the segment number, the sixth byte is zero, and the seventh and eighth bytes specify an offset to the segment. If the relocation type is internal reference and the segment is movable, the fifth byte specifies 0FFh, the sixth byte is zero; and the seventh and eighth bytes specify an ordinal value found in the segment's entry table.

.EXE - DOS EXE File Structure

Offset	Size	Description
00	word	"MZ" - Link file .EXE signature (Mark Zbikowski?)
02	word	length of image mod 512
04	word	size of file in 512 byte pages
06	word	number of relocation items following header
08	word	size of header in 16 byte paragraphs, used to locate the beginning of the load module
0A	word	min # of paragraphs needed to run program
0C	word	max # of paragraphs the program would like
0E	word	offset in load module of stack segment (in paras)
10	word	initial SP value to be loaded
12	word	negative checksum of pgm used while by EXEC loads pgm
14	word	program entry point, (initial IP value)
16	word	offset in load module of the code segment (in paras)
18	word	offset in .EXE file of first relocation item
1A	word	overlay number (0 for root program)

- relocation table and the program load module follow the header
- relocation entries are 32 bit values representing the offset into the load module needing patched
- once the relocatable item is found, the CS register is added to the value found at the calculated offset

Registers at load time of the EXE file are as follows:

AX:	contains number of characters in command tail, or 0
BX:CX	32 bit value indicating the load module memory size
DX	zero
SS:SP	set to stack segment if defined else, SS = CS and SP=FFFFh or top of memory.
DS	set to segment address of EXE header
ES	set to segment address of EXE header
CS:IP	far address of program entry point, (label on "END" statement of program)

GGT
GERBER Cutter data...... some call it ISO Standard when they create there code and don't want to call it Gerber in order not to make any false impression and to boost GGT's sales......

Gerber is extremely reluctant to cooperate - unless you sell their equipment and you have a non-disclosure agreement.

This format was invented in the late 60s !

Quote:

TABLE 2-1. SUMMARY OF INPUT DATA CODES

Input Code	Function
A	Knife up (same as M15)
B	Knife down (same as M14)
D1	Pen down
D2	Pen up
D4	Light source (same as Q)
E	Flick notch (same as M68)
F	Set feed rate
GO4	Set origin point
G7O	Select 3.3 English data
G71	Select 5.1 Metric data format
G91	Identifies GERBER cutter data (4.2 format)
H	File identifier
L	Begin slowdown (same as M25)
M0	EOF (end of file)
M00	Program stop
M01	Optional stop
M14	Knife down (same as B)
M15	Knife up (same as A)
M17	Maximum advance
M18	Inhibit next overcut
M19	Ignore overcut and advance
M20	Message stop (displayed on OCT)
M25	Run part at reduced velocity (same as L)
M26	Restore normal velocity (same as O)
M30	Rewind data file (return)
M31	Labeler data fellows
M40	Enable automatic sharpen
M41	Disable automatic sharpen
M42	Sharpen
M43	Drill (same as R)
M44	auxiliary drill
M46	Lift and plunge corner
M47	Turn off knife intelligence
M48	Turn on knife intelligence
M51	Null knife intelligence
M60	Run part at 95 % velocity
M61	Run part at 90 % velocity
M62	Run part at 85 % velocity
M63	Run part at 80 % velocity
M64	Run part at 75 % velocity
M65	Run part at 70 % velocity
M66	Run part at 65 % velocity
M67	Run part at 60 % velocity
M68	Special notch (same as E)
M69	Conveyor bite
M70	Origin
N	Sequence number of piece
O	Resume normal speed (same as M26)
Q	Establish light as tool (same as D4)
R	Drill (same as M43)
X	Precedes X coordinate area
Y	Precedes Y coordinate area
Z	Bite size identifier
/	Block delete
*	EOB (end of block)

2.6.1 X,Y COORDINATE DATA BLOCK

a. Coordinate data should be 4.2 or 3.3 format expressed in inches, or for metric systems in a 5.1 format in millimeters. Decimal points are assumed according to the data format.

b. Leading zeros should be omitted.

c. The X,Y data must be in absolute coordinates.

d. The negative sign must be include when required. Data with no sign is assumed to be positive.

2.6.2 LABEL DATA BLOCK

The label data can be any printable character except the End of Block character (*) up to 36 characters in length.

2.6.3 END OF BLOCK

The first M31 block is read as position coordinates. The second M31 block is processed along with the X,Y position coordinates. This block of data establishes the angle on which the label is placed.

A second rotational format allows for two blocks of label data. One is to be used in the normal cut mode and the second while in the inverted or "mirror" mode. The data should appear as follows:

```
        *XnnnnYnnnnM31*Normal Label*MirrorLabel*XnnnnYnnnnM31*
```

2.7 M, G, AND D COMMANDS

The following rules apply to the use of the M, G and D commands.

1. A block may contain only one M, G or D command.

2. Leading zeros may be omitted from M, G and D commands.

3. M, G and D commands are modal.

Unquote:

Experience as we go along.........

C100

Gerber Cutter controller:
Info. according to Dr J. Helmig of GGT Brussels
All Cutters work with the C100 Cutter Controller. Older Cutters which still work with the C90 Controller, can use a MID-unit which translates the C100-Data into the right Format.

The C100 runs on an AT under DOS 3.3 and has a 1.2 Mb Floppy. The System can be online connected via any DOS-Compatible Network ! It cannot be addressed via serial lines, since Gerber is using COM1 and COM2 for the Plotter Control.

The C100 controls automatically the knife intelligence. No additional NC-code is needed. For speed-control it is better to cut small pieces first and a bit slower. This is done with code M25, which should be defined after the N-code. i.e. *N10*M25* . As soon as the Controller reads the next N-code, the cutter goes to full speed again.

Lift and plunge of the knife has to be programmed.
i.e.

```
X100Y100*                       MOVE WITH KNIFE UP
M14                             PLUNGE KNIFE
X100Y200*X200Y200*X200Y100*     CUT POSITIONS
X100Y100*                       CUT POSITIONS
M15*                            LIFT KNIFE
```

The Controller is smart enough to know by which angel and when to lift and plunge. If the operator wants to influence this intelligence on critical corners in order to improve the cut-quality, than this is possible with the M46-code - before reaching the corner. i.e.

```
X100Y100*                       MOVE WITH KNIFE UP
M14                             PLUNGE KNIFE
X100Y150*                       KNIFE DOWN
M46                             LIFT&PLUNGE ON NEXT CORNER
X100Y200*                       THIS IS THE CORNER
X200Y200*
```

FURTHER QUESTIONS ???? lets have them !

Table size is different by all Gerber Cutters and should be an option.

- usable table length
- usable table width
- static table or conveyor

The resolution is depending on the format:
Format 4.2 = 1/100 Inch (Standard Format)
Format 3.3 = 1/1000 Inch defined with: N1*G70*
Format 5.1 = 1/10 Millimeter defined with: N1*G71*

Serial Ports:
The C100 has a Quad-Board with 4 serial ports. COM4 with Adress:0x2A0 IRQ=12 is still available. No driver is available to shuffle data into the system. That's why Gerber has no online connection !!! Smart programmer's ?

Further Questions:

GGT
US-06084-0769 Tolland/Connect.
24, Industrial Park Road
West P.O.Box 769
Tel 001 203 871 8082
Mr John Schnetzer

GGT
US-06101-0305 Hartford/Connect.
P.O.Box 305
Tel 001 203.644.1551

G I F (tm)

Graphics Interchange Format (tm)

A standard defining a mechanism
for the storage and transmission
of raster-based graphics information
June 15, 1987

an H&R Block Company
5000 Arlington Centre Blvd.
Columbus, Ohio 43220
(614) 457-8600

Graphics Interchange Format (GIF) Specification

Table of Contents

INTRODUCTION . . . . page 3
GENERAL FILE FORMAT . . . . page 3
GIF SIGNATURE . . . . page 4
SCREEN DESCRIPTOR . . . . page 4
GLOBAL COLOR MAP . . . . page 5
IMAGE DESCRIPTOR . . . . page 6
LOCAL COLOR MAP . . . . page 7
RASTER DATA . . . . page 7
GIF TERMINATOR . . . . page 8
GIF EXTENSION BLOCKS . . . . page 8
APPENDIX A - GLOSSARY . . . . page 9
APPENDIX B - INTERACTIVE SEQUENCES . . . . . . page 10
APPENDIX C - IMAGE PACKAGING & COMPRESSION . . page 12
APPENDIX D - MULTIPLE IMAGE PROCESSING . . . . page 15

INTRODUCTION

'GIF' (tm) is CompuServe's standard for defining generalized color raster images. This 'Graphics Interchange Format' (tm) allows high-quality, high-resolution graphics to be displayed on a variety of graphics hardware and is intended as an exchange and display mechanism for graphics images. The image format described in this document is designed to support current and future image technology and will in addition serve as a basis for future CompuServe graphics products.

The main focus of this document is to provide the technical information necessary for a programmer to implement GIF encoders and decoders. As such, some assumptions are made as to terminology relavent to graphics and programming in general.

The first section of this document describes the GIF data format and its components and applies to all GIF decoders, either as standalone programs or as part of a communications package. Appendix B is a section relavent to decoders that are part of a communications software package and describes the protocol requirements for entering and exiting GIF mode, and responding to host interrogations. A glossary in Appendix A defines some of the terminology used in this document. Appendix C gives a detailed explanation of how the graphics image itself is packaged as a series of data bytes.

Graphics Interchange Format Data Definition

GENERAL FILE FORMAT

```
+-----------------------+
| +-------------------+ |
| |   GIF Signature   | |
| +-------------------+ |
| +-------------------+ |
| | Screen Descriptor | |
| +-------------------+ |
| +-------------------+ |
| | Global Color Map  | |
| +-------------------+ |
. . .               . . .
| +-------------------+ |    ---+
| |  Image Descriptor | |       |
| +-------------------+ |       |
| +-------------------+ |       |
| |  Local Color Map  | |       |-   Repeated 1 to n times
| +-------------------+ |       |
| +-------------------+ |       |
| |    Raster Data    | |       |
| +-------------------+ |    ---+
. . .               . . .
|-    GIF Terminator   -|
+-----------------------+
```

GIF SIGNATURE

The following GIF Signature identifies the data following as a valid GIF image stream. It consists of the following six characters:

G I F 8 7 a

The last three characters '87a' may be viewed as a version number for this particular GIF definition and will be used in general as a reference in documents regarding GIF that address any version dependencies.

SCREEN DESCRIPTOR

The Screen Descriptor describes the overall parameters for all GIF images following. It defines the overall dimensions of the image space or logical screen required, the existance of color mapping information, background screen color, and color depth information. This information is stored in a series of 8-bit bytes as described below.

```
              bits
         7 6 5 4 3 2 1 0  Byte #
        +---------------+
        |               |  1
        +-Screen Width -+      Raster width in pixels (LSB first)
        |               |  2
        +---------------+
        |               |  3
        +-Screen Height-+      Raster height in pixels (LSB first)
        |               |  4
        +-+-----+-+-----+      M = 1, Global color map follows Descriptor
        |M|  cr |0|pixel|  5   cr+1 = # bits of color resolution
        +-+-----+-+-----+      pixel+1 = # bits/pixel in image
        |   background  |  6   background=Color index of screen background
        +---------------+          (color is defined from the Global color
        |0 0 0 0 0 0 0 0|  7        map or default map if none specified)
        +---------------+
```

The logical screen width and height can both be larger than the physical display. How images larger than the physical display are handled is implementation dependent and can take advantage of hardware characteristics (e.g. Macintosh scrolling windows). Otherwise images can be clipped to the edges of the display.

The value of 'pixel' also defines the maximum number of colors within an image. The range of values for 'pixel' is 0 to 7 which represents 1 to 8 bits. This translates to a range of 2 (B & W) to 256 colors. Bit 3 of word 5 is reserved for future definition and must be zero.

GLOBAL COLOR MAP

The Global Color Map is optional but recommended for images where accurate color rendition is desired. The existence of this color map is indicated in the 'M' field of byte 5 of the Screen Descriptor. A color map can also be associated with each image in a GIF file as described later. However this global map will normally be used because of hardware restrictions in equipment available today. In the individual Image Descriptors the 'M' flag will normally be zero. If the Global Color Map is present, it's definition immediately follows the Screen Descriptor. The number of color map entries following a Screen Descriptor is equal to 2**(# bits per pixel), where each entry consists of three byte values representing the relative intensities of red, green and blue respectively. The structure of the Color Map block is:

```
              bits
         7 6 5 4 3 2 1 0  Byte #
        +---------------+
        | red intensity |  1    Red value for color index 0
        +---------------+
        |green intensity|  2    Green value for color index 0
```

```
     +---------------+
     | blue intensity|  3    Blue value for color index 0
     +---------------+
     | red intensity |  4    Red value for color index 1
     +---------------+
     |green intensity|  5    Green value for color index 1
     +---------------+
     | blue intensity|  6    Blue value for color index 1
     +---------------+
     :               :       (Continues for remaining colors)
```

Each image pixel value received will be displayed according to its closest match with an available color of the display based on this color map. The color components represent a fractional intensity value from none (0) to full (255). White would be represented as (255,255,255), black as (0,0,0) and medium yellow as (180,180,0). For display, if the device supports fewer than 8 bits per color component, the higher order bits of each component are used. In the creation of a GIF color map entry with hardware supporting fewer than 8 bits per component, the component values for the hardware should be converted to the 8-bit format with the following calculation:

```
     <map_value> = <component_value>*255/(2**<nbits> -1)
```

This assures accurate translation of colors for all displays. In the cases of creating GIF images from hardware without color palette capability, a fixed palette should be created based on the available display colors for that hardware. If no Global Color Map is indicated, a default color map is generated internally which maps each possible incoming color index to the same hardware color index modulo <n> where <n> is the number of available hardware colors.

IMAGE DESCRIPTOR

The Image Descriptor defines the actual placement and extents of the following image within the space defined in the Screen Descriptor. Also defined are flags to indicate the presence of a local color lookup map, and to define the pixel display sequence. Each Image Descriptor is introduced by an image separator character. The role of the Image Separator is simply to provide a synchronization character to introduce an Image Descriptor. This is desirable if a GIF file happens to contain more than one image. This character is defined as 0x2C hex or ',' (comma). When this character is encountered between images, the Image Descriptor will follow immediately.

Any characters encountered between the end of a previous image and the image separator character are to be ignored. This allows future GIF enhancements to be present in newer image formats and yet ignored safely by older software decoders.

```
           bits
      7 6 5 4 3 2 1 0  Byte #
     +---------------+
     |0 0 1 0 1 1 0 0|  1    ',' - Image separator character
     +---------------+
     |               |  2    Start of image in pixels from the
     +-  Image Left -+       left side of the screen (LSB first)
     |               |  3
     +---------------+
     |               |  4
     +-  Image Top  -+       Start of image in pixels from the
     |               |  5    top of the screen (LSB first)
     +---------------+
     |               |  6
     +- Image Width -+       Width of the image in pixels (LSB first)
     |               |  7
     +---------------+
     |               |  8
     +- Image Height-+       Height of the image in pixels (LSB first)
     |               |  9
     +-+-+-+-+-+-----+       M=0 - Use global color map, ignore 'pixel'
     |M|I|0|0|0|pixel| 10    M=1 - Local color map follows, use 'pixel'
     +-+-+-+-+-+-----+       I=0 - Image formatted in Sequential order
                             I=1 - Image formatted in Interlaced order
                             pixel+1 - # bits per pixel for this image
```

The specifications for the image position and size must be confined to the dimensions defined by the Screen Descriptor. On the other hand it is not necessary that the image fill the entire screen defined.

LOCAL COLOR MAP

A Local Color Map is optional and defined here for future use. If the 'M' bit of byte 10 of the Image Descriptor is set, then a color map follows the Image Descriptor that applies only to the following image. At the end of the image, the color map will revert to that defined after the Screen Descriptor. Note that the 'pixel' field of byte 10 of the Image Descriptor is used only if a Local Color Map is indicated. This defines the parameters not only for the image pixel size, but determines the number of color map entries that follow. The bits per pixel value will also revert to the value specified in the Screen Descriptor when processing of the image is complete.

RASTER DATA

The format of the actual image is defined as the series of pixel color index values that make up the image. The pixels are stored left to right sequentially for an image row. By default each image row is written sequentially, top to bottom. In the case that the Interlace or 'I' bit is set in byte 10 of the Image Descriptor then the row order of the image display follows a four-pass process in which the image is filled in by widely spaced rows. The first pass writes every 8th row, starting with the top row of the image window. The second pass writes every 8th row starting at the fifth row from the top. The third pass writes every 4th row starting at the third row from the top. The fourth pass completes the image, writing every other row, starting at the second row from the top. A graphic description of this process follows:

```
Image
Row  Pass 1  Pass 2  Pass 3  Pass 4          Result
---------------------------------------------------
  0  **1a**                                  **1a**
  1                          **4a**          **4a**
  2                  **3a**                  **3a**
  3                          **4b**          **4b**
  4          **2a**                          **2a**
  5                          **4c**          **4c**
  6                  **3b**                  **3b**
  7                          **4d**          **4d**
  8  **1b**                                  **1b**
  9                          **4e**          **4e**
 10                  **3c**                  **3c**
 11                          **4f**          **4f**
 12          **2b**                          **2b**
. . .
```

The image pixel values are processed as a series of color indices which map into the existing color map. The resulting color value from the map is what is actually displayed. This series of pixel indices, the number of which is equal to image-width*image-height pixels, are passed to the GIF image data stream one value per pixel, compressed and packaged according to a version of the LZW compression algorithm as defined in Appendix C.

GIF TERMINATOR

In order to provide a synchronization for the termination of a GIF image file, a GIF decoder will process the end of GIF mode when the character 0x3B hex or ';' is found after an image has been processed. By convention the decoding software will pause and wait for an action indicating that the user is ready to continue. This may be a carriage return entered at the keyboard or a mouse click. For interactive applications this user action must be passed on to the host as a carriage return character so that the host application can continue. The decoding software will then typically leave graphics mode and resume any previous process.

GIF EXTENSION BLOCKS

To provide for orderly extension of the GIF definition, a mechanism for defining the packaging of extensions within a GIF data stream is necessary. Specific GIF extensions are to be defined and documented by CompuServe in order to provide a controlled enhancement path.

GIF Extension Blocks are packaged in a manner similar to that used by the raw data though not compressed. The basic structure is:

```
 7 6 5 4 3 2 1 0  Byte #
+---------------+
|0 0 1 0 0 0 0 1|  1       '!' - GIF Extension Block Introducer
```

```
        +---------------+
        | function code |  2       Extension function code (0 to 255)
        +---------------+    ---+
        |  byte count   |       |
        +---------------+       |
        :               :       +-- Repeated as many times as necessary
        |func data bytes|       |
        :               :       |
        +---------------+    ---+
        . . .       . . .
        +---------------+
        |0 0 0 0 0 0 0 0|       zero byte count (terminates block)
        +---------------+
```

A GIF Extension Block may immediately preceed any Image Descriptor or occur before the GIF Terminator.

All GIF decoders must be able to recognize the existence of GIF Extension Blocks and read past them if unable to process the function code. This ensures that older decoders will be able to process extended GIF image files in the future, though without the additional functionality.

GLOSSARY

Pixel - The smallest picture element of a graphics image. This usually corresponds to a single dot on a graphics screen. Image resolution is typically given in units of pixels. For example a fairly standard graphics screen format is one 320 pixels across and 200 pixels high. Each pixel can appear as one of several colors depending on the capabilities of the graphics hardware.

Raster - A horizontal row of pixels representing one line of an image. A typical method of working with images since most hardware is oriented to work most efficiently in this manner.

LSB - Least Significant Byte. Refers to a convention for two byte numeric values in which the less significant byte of the value preceeds the more significant byte. This convention is typical on many microcomputers.

Color Map - The list of definitions of each color used in a GIF image. These desired colors are converted to available colors through a table which is derived by assigning an incoming color index (from the image) to an output color index (of the hardware). While the color map definitons are specified in a GIF image, the output pixel colors will vary based on the hardware used and its ability to match the defined color.

Interlace - The method of displaying a GIF image in which multiple passes are made, outputting raster lines spaced apart to provide a way of visualizing the general content of an entire image before all of the data has been processed.

B Protocol - A CompuServe-developed error-correcting file transfer protocol available in the public domain and implemented in CompuServe VIDTEX products. This error checking mechanism will be used in transfers of GIF images for interactive applications.

LZW - A sophisticated data compression algorithm based on work done by Lempel-Ziv & Welch which has the feature of very efficient one-pass encoding and decoding. This allows the image to be decompressed and displayed at the same time. The original article from which this technique was adapted is:

> Terry A. Welch, "A Technique for High Performance Data Compression", IEEE Computer, vol 17 no 6 (June 1984)

This basic algorithm is also used in the public domain ARC file compression utilities. The CompuServe adaptation of LZW for GIF is described in Appendix C.

GIF Sequence Exchanges for an Interactive Environment

The following sequences are defined for use in mediating control between a GIF sender and GIF receiver over an interactive communications line. These sequences do not apply to applications that involve downloading of static GIF files and are not considered part of a GIF file.

GIF CAPABILITIES ENQUIRY

The GCE sequence is issued from a host and requests an interactive GIF decoder to return a response message that defines the graphics parameters for the decoder. This involves returning information about available screen sizes, number of bits/color supported and the amount of color detail supported. The escape sequence for the GCE is defined as:

```
     ESC [ > 0 g     (g is lower case, spaces inserted for clarity)
                      (0x1B 0x5B 0x3E 0x30 0x67)
```

GIF CAPABILITIES RESPONSE

The GIF Capabilities Response message is returned by an interactive GIF decoder and defines the decoder's display capabilities for all graphics modes that are supported by the software. Note that this can also include graphics printers as well as a monitor screen. The general format of this message is:

```
  #version;protocol{;dev, width, height, color-bits, color-res}... <CR>
'#'          - GCR identifier character (Number Sign)
version      - GIF format version number;  initially '87a'
protocol='0' - No end-to-end protocol supported by decoder
               Transfer as direct 8-bit data stream.
protocol='1' - Can use an error correction protocol to transfer GIF data
            interactively from the host directly to the display.
dev = '0'    - Screen parameter set follows
dev = '1'    - Printer parameter set follows
width        - Maximum supported display width in pixels
height       - Maximum supported display height in pixels
color-bits   - Number of  bits  per  pixel  supported.   The  number  of
            supported colors is therefore 2**color-bits.
color-res    - Number of bits  per  color  component  supported  in  the
            hardware  color  palette.   If  color-res  is  '0'  then  no
            hardware palette table is available.
```

Note that all values in the GCR are returned as ASCII decimal numbers and the message is terminated by a Carriage Return character.

Appendix B - Interactive Sequences

The following GCR message describes three standard EGA configurations with no printer; the GIF data stream can be processed within an error correcting protocol:

```
     #87a;1 ;0,320,200,4,0 ;0,640,200,2,2 ;0,640,350,4,2<CR>
```

ENTER GIF GRAPHICS MODE

Two sequences are currently defined to invoke an interactive GIF decoder into action. The only difference between them is that different output media are selected. These sequences are:

```
  ESC [ > 1 g   Display GIF image on screen
                (0x1B 0x5B 0x3E 0x31 0x67)
  ESC [ > 2 g   Display image directly to an attached graphics  printer.
                The  image  may optionally be displayed on the screen as
                well.
                (0x1B 0x5B 0x3E 0x32 0x67)
```

Note that the 'g' character terminating each sequence is in lower case.

INTERACTIVE ENVIRONMENT

The assumed environment for the transmission of GIF image data from an interactive application is a full 8-bit data stream from host to micro. All 256 character codes must be transferrable. The establishing of an 8-bit data path for communications will normally be taken care of by the host application programs. It is however up to the receiving communications programs supporting GIF to be able to receive and pass on all 256 8-bit codes to the GIF decoder software.

Appendix C - Image Packaging & Compression

The Raster Data stream that represents the actual output image can be represented as:

```
      7 6 5 4 3 2 1 0
     +---------------+
     |   code size   |
     +---------------+     ---+
     |blok byte count|        |
     +---------------+        |
     :               :        +-- Repeated as many times as necessary
     |  data bytes   |        |
     :               :        |
     +---------------+     ---+
     . . .       . . .
     +---------------+
     |0 0 0 0 0 0 0 0|       zero byte count (terminates data stream)
     +---------------+
```

The conversion of the image from a series of pixel values to a transmitted or stored character stream involves several steps. In brief these steps are:

1. Establish the Code Size - Define the number of bits needed to represent the actual data.
2. Compress the Data - Compress the series of image pixels to a series of compression codes.
3. Build a Series of Bytes - Take the set of compression codes and convert to a string of 8-bit bytes.
4. Package the Bytes - Package sets of bytes into blocks preceeded by character counts and output.

ESTABLISH CODE SIZE

The first byte of the GIF Raster Data stream is a value indicating the minimum number of bits required to represent the set of actual pixel values. Normally this will be the same as the number of color bits. Because of some algorithmic constraints however, black & white images which have one color bit must be indicated as having a code size of 2. This code size value also implies that the compression codes must start out one bit longer.

COMPRESSION

The LZW algorithm converts a series of data values into a series of codes which may be raw values or a code designating a series of values. Using text characters as an analogy, the output code consists of a character or a code representing a string of characters.

The LZW algorithm used in GIF matches algorithmically with the standard LZW algorithm with the following differences:

1. A special Clear code is defined which resets all compression/decompression parameters and tables to a start-up state. The value of this code is 2**<code size>. For example if the code size indicated was 4 (image was 4 bits/pixel) the Clear code value would be 16 (10000 binary). The Clear code can appear at any point in the image data stream and therefore requires the LZW algorithm to process succeeding codes as if a new data stream was starting. Encoders should output a Clear code as the first code of each image data stream.
2. An End of Information code is defined that explicitly indicates the end of the image data stream. LZW processing terminates when this code is encountered. It must be the last code output by the encoder for an image. The value of this code is <Clear code>+1.
3. The first available compression code value is <Clear code>+2.
4. The output codes are of variable length, starting at <code size>+1 bits per code, up to 12 bits per code. This defines a maximum code value of 4095 (hex FFF). Whenever the LZW code value would exceed the current code length, the code length is increased by one. The packing/unpacking of these codes must then be altered to reflect the new code length.

BUILD 8-BIT BYTES

Because the LZW compression used for GIF creates a series of variable length codes, of between 3 and 12 bits each, these codes must be reformed into a series of 8-bit bytes that will be the characters actually stored or transmitted. This provides additional compression of the image. The codes are formed into a stream of bits as if they were packed right to left and then picked off 8 bits at a time to be output. Assuming a character array of 8 bits per character and using 5 bit codes to be packed, an example layout would be similar to:

```
 byte n       byte 5   byte 4   byte 3   byte 2   byte 1
+-.....-----+--------+--------+--------+--------+--------+
| and so on |hhhhhggg|ggfffffe|eeeedddd|dcccccbb|bbbaaaaa|
+-.....-----+--------+--------+--------+--------+--------+
```

Note that the physical packing arrangement will change as the number of bits per compression code change but the concept remains the same.

PACKAGE THE BYTES

Once the bytes have been created, they are grouped into blocks for output by preceeding each block of 0 to 255 bytes with a character count byte. A block with a zero byte count terminates the Raster Data stream for a given image. These blocks are what are actually output for the

GIF image. This block format has the side effect of allowing a decoding program the ability to read past the actual image data if necessary by reading block counts and then skipping over the data.

Appendix D - Multiple Image Processing

Since a GIF data stream can contain multiple images, it is necessary to describe processing and display of such a file. Because the image descriptor allows for placement of the image within the logical screen, it is possible to define a sequence of images that may each be a partial screen, but in total fill the entire screen. The guidelines for handling the multiple image situation are:

1. There is no pause between images. Each is processed immediately as seen by the decoder.
2. Each image explicitly overwrites any image already on the screen inside of its window. The only screen clears are at the beginning and end of the GIF image process. See discussion on the GIF terminator.

4444444444444444444444444444444444444444444444444444444444444444444444444444

Cover Sheet for the GIF89a Specification

DEFERRED CLEAR CODE IN LZW COMPRESSION

There has been confusion about where clear codes can be found in the data stream. As the specification says, they may appear at anytime. There is not a requirement to send a clear code when the string table is full.

It is the encoder's decision as to when the table should be cleared. When the table is full, the encoder can chose to use the table as is, making no changes to it until the encoder chooses to clear it. The encoder during this time sends out codes that are of the maximum Code Size.

As we can see from the above, when the decoder's table is full, it must not change the table until a clear code is received. The Code Size is that of the maximum Code Size. Processing other than this is done normally.

Because of a large base of decoders that do not handle the decompression in this manner, we ask developers of GIF encoding software to NOT implement this feature until at least January 1991 and later if they see that their particular market is not ready for it. This will give developers of GIF decoding software time to implement this feature and to get it into the hands of their clients before the decoders start "breaking" on the new GIF's. It is not required that encoders change their software to take advantage of the deferred clear code, but it is for decoders.

APPLICATION EXTENSION BLOCK - APPLICATION IDENTIFIER

There will be a Courtesy Directory file located on CompuServe in the PICS forum. This directory will contain Application Identifiers for Application Extension Blocks that have been used by developers of GIF applications. This file is intended to help keep developers that wish to create Application Extension Blocks from using the same Application Identifiers. This is not an official directory; it is for voluntary participation only and does not guarantee that someone will not use the same identifier.

E-Mail can be sent to Larry Wood (forum manager of PICS) indicating the request for inclusion in this file with an identifier.

GRAPHICS INTERCHANGE FORMAT(sm)

Version 89a (modified)

Table of Contents

Disclaimer... 1

Foreword... 1

Licensing.. 1

About the Document... 2

General Description.. 2

Version Numbers.. 2

The Encoder.. 3

The Decoder.. 3

Compliance... 3

About Recommendations.. 4

About Color Tables... 4

Blocks, Extensions and Scope... 4

Block Sizes.. 5

Using GIF as an embedded protocol.. 5

Data Sub-blocks.. 5

Block Terminator... 6

Header... 7

Logical Screen Descriptor.. 8

Global Color Table... 10

Image Descriptor... 11

Local Color Table.. 13

Table Based Image Data... 14

Graphic Control Extension.. 15

Comment Extension.. 17

Plain Text Extension... 18

Application Extension.. 21

Trailer... 23

Quick Reference Table.. 24

GIF Grammar.. 25

Glossary... 27

Conventions.. 28

Interlaced Images.. 29

Variable-Length-Code LZW Compression....................................... 30

On-line Capabilities Dialogue.. 33

1. Disclaimer.

The information provided herein is subject to change without notice. In no event will CompuServe Incorporated be liable for damages, including any loss of revenue, loss of profits or other incidental or consequential damages arising out of the use or inability to use the information; CompuServe Incorporated makes no claim as to the suitability of the information.

2. Foreword.

This document defines the Graphics Interchange Format(sm). The specification given here defines version 89a, which is an extension of version 87a.

The Graphics Interchange Format(sm) as specified here should be considered complete; any deviation from it should be considered invalid, including but not limited to, the use of reserved or undefined fields within control or data blocks, the inclusion of extraneous data within or between blocks, the use of methods or algorithms not specifically listed as part of the format, etc. In general, any and all deviations, extensions or modifications not specified in this document should be considered to be in violation of the format and should be avoided.

3. Licensing.

The Graphics Interchange Format(c) is the copyright property of CompuServe Incorporated. Only CompuServe Incorporated is authorized to define, redefine, enhance, alter, modify or change in any way the definition of the format.

CompuServe Incorporated hereby grants a limited, non-exclusive, royalty-free license for the use of the Graphics Interchange Format(sm) in computer software; computer software utilizing GIF(sm) must acknowledge ownership of the Graphics Interchange Format and its Service Mark by CompuServe Incorporated, in User and Technical Documentation. Computer software utilizing GIF, which is distributed or may be distributed without User or Technical Documentation must display to the screen or printer a message acknowledging ownership of the Graphics Interchange Format and the Service Mark by CompuServe Incorporated; in this case, the acknowledgement may be displayed in an opening screen or leading banner, or a closing screen or trailing banner. A message such as the following may be used:

> "The Graphics Interchange Format(c) is the Copyright property of CompuServe Incorporated. GIF(sm) is a Service Mark property of CompuServe Incorporated."

For further information, please contact :

CompuServe Incorporated
Graphics Technology Department
5000 Arlington Center Boulevard
Columbus, Ohio 43220
U. S. A.

CompuServe Incorporated maintains a mailing list with all those individuals and organizations who wish to receive copies of this document when it is corrected or revised. This service is offered free of charge; please provide us with your mailing address.

Users of this specification should note that the LZW compression and decompression methods described in U.S. Patent No. 4,558,302 and certain corresponding foreign patents are owned by Unisys Corporation. Software and hardware developers may be required to obtain a license under this patent in order to develop and market products using GIF LZW compression and decompression. Unisys has agreed that developers may obtain such a license on reasonable, non-discriminatory terms and conditions. Further information may be obtained from: Welch Licensing Department, Office of the General Counsel, M/S C1SW19, Unisys Corporation, Blue Bell, PA 19424.

4. About the Document.

This document describes in detail the definition of the Graphics Interchange Format. This document is intended as a programming reference; it is recommended that the entire document be read carefully before programming, because of the interdependence of the various parts. There is an individual section for each of the Format blocks. Within each section, the sub-section labeled Required Version refers to the version number that an encoder will have to use if the corresponding block is used in the Data Stream. Within each section, a diagram describes the individual fields in the block; the diagrams are drawn vertically; top bytes in the diagram appear first in the Data Stream. Bits within a byte are drawn most significant on the left end. Multi-byte numeric fields are ordered Least Significant Byte first. Numeric constants are represented as Hexadecimal numbers, preceded by "0x". Bit fields within a byte are described in order from most significant bits to least significant bits.

5. General Description.

The Graphics Interchange Format(sm) defines a protocol intended for the on-line transmission and interchange of raster graphic data in a way that is independent of the hardware used in their creation or display.

The Graphics Interchange Format is defined in terms of blocks and sub-blocks which contain relevant parameters and data used in the reproduction of a graphic. A GIF Data Stream is a sequence of protocol blocks and sub-blocks representing a collection of graphics. In general, the graphics in a Data Stream are assumed to be related to some degree, and to share some control information; it is recommended that encoders attempt to group together related graphics in order to minimize hardware changes during processing and to minimize control information overhead. For the same reason, unrelated graphics or graphics which require resetting hardware parameters should be encoded separately to the extent possible.

A Data Stream may originate locally, as when read from a file, or it may originate remotely, as when transmitted over a data communications line. The Format is defined with the assumption that an error-free Transport Level Protocol is used for communications; the Format makes no provisions for error-detection and error-correction.

The GIF Data Stream must be interpreted in context, that is, the application program must rely on information external to the Data Stream to invoke the decoder process.

6. Version Numbers.

The version number in the Header of a Data Stream is intended to identify the minimum set of capabilities required of a decoder in order to fully process the Data Stream. An encoder should use the earliest possible version number that includes all the blocks used in the Data Stream. Within each block section in this document, there is an entry labeled Required Version which specifies the earliest version number that includes the corresponding block. The encoder should make every attempt to use the earliest version number covering all the blocks in the Data Stream; the unnecessary use of later version numbers will hinder processing by some decoders.

7. The Encoder.

The Encoder is the program used to create a GIF Data Stream. From raster data and other information, the encoder produces the necessary control and data blocks needed for reproducing the original graphics.

The encoder has the following primary responsibilities.

- Include in the Data Stream all the necessary information to reproduce the graphics.

- Insure that a Data Stream is labeled with the earliest possible Version Number that will cover the definition of all the blocks in it; this is to ensure that the largest number of decoders can process the Data Stream.

- Ensure encoding of the graphics in such a way that the decoding process is optimized. Avoid redundant information as much as possible.

- To the extent possible, avoid grouping graphics which might require resetting hardware parameters during the decoding process.

- Set to zero (off) each of the bits of each and every field designated as reserved. Note that some fields in the Logical Screen Descriptor and the Image Descriptor were reserved under Version 87a, but are used under version 89a.

8. The Decoder.

The Decoder is the program used to process a GIF Data Stream. It processes the Data Stream sequentially, parsing the various blocks and sub-blocks, using the control information to set hardware and process parameters and interpreting the data to render the graphics.

The decoder has the following primary responsibilities.

- Process each graphic in the Data Stream in sequence, without delays other than those specified in the control information.

- Set its hardware parameters to fit, as closely as possible, the control information contained in the Data Stream.

9. Compliance.

An encoder or a decoder is said to comply with a given version of the Graphics Interchange Format if and only if it fully conforms with and correctly implements the definition of the standard associated with that version. An encoder or a decoder may be compliant with a given version number and not compliant with some subsequent version.

10. About Recommendations.

Each block section in this document contains an entry labeled Recommendation; this section lists a set of recommendations intended to guide and organize the use of the particular blocks. Such recommendations are geared towards making the functions of encoders and decoders more efficient, as well as making optimal use of the communications bandwidth. It is advised that these recommendations be followed.

11. About Color Tables.

The GIF format utilizes color tables to render raster-based graphics. A color table can have one of two different scopes: global or local. A Global Color Table is used by all those graphics in the Data Stream which do not have a Local Color Table associated with them. The scope of the Global Color Table is the entire Data Stream. A Local Color Table is always associated with the graphic that immediately follows it; the scope of a Local Color Table is limited to that single graphic. A Local Color Table supersedes a Global Color Table, that is, if a Data Stream contains a Global Color Table, and an image has a Local Color Table associated with it, the decoder must save the Global Color Table, use the Local Color Table to render the image, and then restore the Global Color Table. Both types of color tables are optional, making it possible for a Data Stream to contain numerous graphics without a color table at all. For this reason, it is recommended that the decoder save the last Global Color Table used until another Global Color Table is encountered. In this way, a Data Stream which does not contain either a Global Color Table or a Local Color Table may be processed using the last Global Color Table saved. If a Global Color Table from a previous Stream is used, that table becomes the Global Color Table of the present Stream. This is intended to reduce the overhead incurred by color tables. In particular, it is recommended that an encoder use only one Global Color Table if all the images in related Data Streams can be rendered with the same table. If no color table is available at all, the decoder is free to use a system color table or a table of its own. In that case, the decoder may use a color table with as many colors as its hardware is able to support; it is recommended that such a table have black and white as its first two entries, so that monochrome images can be rendered adequately.

The Definition of the GIF Format allows for a Data Stream to contain only the Header, the Logical Screen Descriptor, a Global Color Table and the GIF Trailer. Such a Data Stream would be used to load a decoder with a Global Color Table, in preparation for subsequent Data Streams without a color table at all.

12. Blocks, Extensions and Scope.

Blocks can be classified into three groups : Control, Graphic-Rendering and Special Purpose. Control blocks, such as the Header, the Logical Screen Descriptor, the Graphic Control Extension and the Trailer, contain information used to control the process of the Data Stream or information used in setting hardware parameters. Graphic-Rendering blocks such as the Image Descriptor and the Plain Text Extension contain information and data used to render a graphic on the display device. Special Purpose blocks such as the Comment Extension and the Application Extension are neither used to control the process of the Data Stream nor do they contain information or data used to render a graphic on the display device. With the exception of the Logical Screen Descriptor and the Global Color Table, whose scope is the entire Data Stream, all other Control blocks have a limited scope, restricted to the Graphic-Rendering block that follows them. Special Purpose blocks do not delimit the scope of any Control blocks; Special Purpose blocks are transparent to the decoding process. Graphic-Rendering blocks and extensions are used as scope delimiters for Control blocks and extensions. The labels used to identify labeled blocks fall into three ranges : 0x00-0x7F (0-127) are the Graphic Rendering blocks, excluding the Trailer (0x3B); 0x80-0xF9 (128-249) are the Control blocks; 0xFA-0xFF (250-255) are the Special Purpose blocks. These ranges are defined so that decoders can handle block scope by appropriately identifying block labels, even when the block itself cannot be processed.

13. Block Sizes.

The Block Size field in a block, counts the number of bytes remaining in the block, not counting the Block Size field itself, and not counting the Block Terminator, if one is to follow. Blocks other than Data Blocks are intended to be of fixed length; the Block Size field is provided in order to facilitate skipping them, not to allow their size to change in the future. Data blocks and sub-blocks are of variable length to accommodate the amount of data.

14. Using GIF as an embedded protocol.

As an embedded protocol, GIF may be part of larger application protocols, within which GIF is used to render graphics. In such a case, the application protocol could define a block within which the GIF Data Stream would be contained. The application program would then invoke a GIF decoder upon encountering a block of type GIF. This approach is recommended in favor of using Application Extensions, which become overhead for all other applications that do not process them. Because a GIF Data Stream must be processed in context, the application must rely on some means of identifying the GIF Data Stream outside of the Stream itself.

15. Data Sub-blocks.

a. Description. Data Sub-blocks are units containing data. They do not have a label, these blocks are processed in the context of control blocks, wherever data blocks are specified in the format. The first byte of the Data sub-block indicates the number of data bytes to follow. A data sub-block may contain from 0 to 255 data bytes. The size of the block does not account for the size byte itself, therefore, the empty sub-block is one whose size field contains 0x00.

b. Required Version. 87a.

c. Syntax.

```
      7 6 5 4 3 2 1 0        Field Name                    Type
     +---------------+
  0  |               |       Block Size                    Byte
     +---------------+
  1  |               |
     +-             -+
  2  |               |
     +-             -+
  3  |               |
     +-             -+
     |               |       Data Values                   Byte
     +-             -+
 up  |               |
     +-   . . . .   -+
 to  |               |
     +-             -+
     |               |
     +-             -+
255  |               |
     +---------------+
```

i) Block Size - Number of bytes in the Data Sub-block; the size must be within 0 and 255 bytes, inclusive.

ii) Data Values - Any 8-bit value. There must be exactly as many Data Values as specified by the Block Size field.

d. Extensions and Scope. This type of block always occurs as part of a larger unit. It does not have a scope of itself.

e. Recommendation. None.

16. Block Terminator.

a. Description. This zero-length Data Sub-block is used to terminate a sequence of Data Sub-blocks. It contains a single byte in the position of the Block Size field and does not contain data.

b. Required Version. 87a.

c. Syntax.

```
      7 6 5 4 3 2 1 0        Field Name                    Type
     +---------------+
  0  |               |       Block Size                    Byte
```

```
      +---------------+
```

i) Block Size - Number of bytes in the Data Sub-block; this field contains the fixed value 0x00.

ii) Data Values - This block does not contain any data.

d. Extensions and Scope. This block terminates the immediately preceding sequence of Data Sub-blocks. This block cannot be modified by any extension.

e. Recommendation. None.

17. Header.

a. Description. The Header identifies the GIF Data Stream in context. The Signature field marks the beginning of the Data Stream, and the Version field identifies the set of capabilities required of a decoder to fully process the Data Stream. This block is REQUIRED; exactly one Header must be present per Data Stream.

b. Required Version. Not applicable. This block is not subject to a version number. This block must appear at the beginning of every Data Stream.

c. Syntax.

```
       7 6 5 4 3 2 1 0        Field Name                    Type
      +---------------+
   0  |               |       Signature                     3 Bytes
      +-             -+
   1  |               |
      +-             -+
   2  |               |
      +---------------+
   3  |               |       Version                       3 Bytes
      +-             -+
   4  |               |
      +-             -+
   5  |               |
      +---------------+
```

i) Signature - Identifies the GIF Data Stream. This field contains the fixed value 'GIF'.

ii) Version - Version number used to format the data stream. Identifies the minimum set of capabilities necessary to a decoder to fully process the contents of the Data Stream.

```
Version Numbers as of 10 July 1990 :       "87a" - May 1987
                                           "89a" - July 1989
```

Version numbers are ordered numerically increasing on the first two digits starting with 87 (87,88,...,99,00,...,85,86) and alphabetically increasing on the third character (a,...,z).

iii) Extensions and Scope. The scope of this block is the entire Data Stream. This block cannot be modified by any extension.

d. Recommendations.

i) Signature - This field identifies the beginning of the GIF Data Stream; it is not intended to provide a unique signature for the identification of the data. It is recommended that the GIF Data Stream be identified externally by the application. (Refer to Appendix G for on-line identification of the GIF Data Stream.)

ii) Version - ENCODER : An encoder should use the earliest possible version number that defines all the blocks used in the Data Stream. When two or more Data Streams are combined, the latest of the individual version numbers should be used for the resulting Data Stream. DECODER : A decoder should attempt to process the data stream to the best of its ability; if it encounters a version number which it is not capable of processing fully, it should nevertheless, attempt to process the data stream to the best of its ability, perhaps after warning the user that the data may be incomplete.

18. Logical Screen Descriptor.

a. Description. The Logical Screen Descriptor contains the parameters necessary to define the area of the display device within which the images will be rendered. The coordinates in this block are given with respect to the top-left corner of the virtual screen; they do not necessarily refer to absolute coordinates on the display device. This implies that they could refer to window coordinates in a window-based environment or printer coordinates when a printer is used.

This block is REQUIRED; exactly one Logical Screen Descriptor must be present per Data Stream.

b. Required Version. Not applicable. This block is not subject to a version number. This block must appear immediately after the Header.

c. Syntax.

```
      7 6 5 4 3 2 1 0        Field Name                    Type
     +---------------+
  0  |               |       Logical Screen Width          Unsigned
     +-             -+
  1  |               |
     +---------------+
  2  |               |       Logical Screen Height         Unsigned
     +-             -+
  3  |               |
     +---------------+
  4  | |     | |     |       <Packed Fields>               See below
     +---------------+
  5  |               |       Background Color Index        Byte
     +---------------+
  6  |               |       Pixel Aspect Ratio            Byte
     +---------------+
```

<Packed Fields> =	Global Color Table Flag	1 Bit
	Color Resolution	3 Bits
	Sort Flag	1 Bit
	Size of Global Color Table	3 Bits

i) Logical Screen Width - Width, in pixels, of the Logical Screen where the images will be rendered in the displaying device.

ii) Logical Screen Height - Height, in pixels, of the Logical Screen where the images will be rendered in the displaying device.

iii) Global Color Table Flag - Flag indicating the presence of a Global Color Table; if the flag is set, the Global Color Table will immediately follow the Logical Screen Descriptor. This flag also selects the interpretation of the Background Color Index; if the flag is set, the value of the Background Color Index field should be used as the table index of the background color. (This field is the most significant bit of the byte.)

Values :   0 -   No Global Color Table follows, the Background Color Index field is meaningless.
            1 -   A Global Color Table will immediately follow, the Background Color Index field is meaningful.

iv) Color Resolution - Number of bits per primary color available to the original image, minus 1. This value represents the size of the entire palette from which the colors in the graphic were selected, not the number of colors actually used in the graphic. For example, if the value in this field is 3, then the palette of the original image had 4 bits per primary color available to create the image. This value should be set to indicate the richness of the original palette, even if not every color from the whole palette is available on the source machine.

v) Sort Flag - Indicates whether the Global Color Table is sorted. If the flag is set, the Global Color Table is sorted, in order of decreasing importance. Typically, the order would be decreasing frequency, with most frequent color first. This assists a decoder, with fewer available colors, in choosing the best subset of colors; the decoder may use an initial segment of the table to render the graphic.

Values :   0 -   Not ordered.
            1 -   Ordered by decreasing importance, most important color first.

vi) Size of Global Color Table - If the Global Color Table Flag is set to 1, the value in this field is used to calculate the number of bytes contained in the Global Color Table. To determine that actual size of the color table, raise 2 to [the value of the field + 1]. Even if there is no Global Color Table specified, set this field according to the above formula so that decoders can choose the best graphics mode to display the stream in. (This field is made up of the 3 least significant bits of the byte.)

vii) Background Color Index - Index into the Global Color Table for

the Background Color. The Background Color is the color used for those pixels on the screen that are not covered by an image. If the Global Color Table Flag is set to (zero), this field should be zero and should be ignored.

viii) Pixel Aspect Ratio - Factor used to compute an approximation of the aspect ratio of the pixel in the original image. If the value of the field is not 0, this approximation of the aspect ratio is computed based on the formula:

Aspect Ratio = (Pixel Aspect Ratio + 15) / 64

The Pixel Aspect Ratio is defined to be the quotient of the pixel's width over its height. The value range in this field allows specification of the widest pixel of 4:1 to the tallest pixel of 1:4 in increments of 1/64th.

```
Values :        0 -   No aspect ratio information is given.
           1..255 -   Value used in the computation.
```

d. Extensions and Scope. The scope of this block is the entire Data Stream. This block cannot be modified by any extension.

e. Recommendations. None.

19. Global Color Table.

a. Description. This block contains a color table, which is a sequence of bytes representing red-green-blue color triplets. The Global Color Table is used by images without a Local Color Table and by Plain Text Extensions. Its presence is marked by the Global Color Table Flag being set to 1 in the Logical Screen Descriptor; if present, it immediately follows the Logical Screen Descriptor and contains a number of bytes equal to

3 x 2^(Size of Global Color Table+1).

This block is OPTIONAL; at most one Global Color Table may be present per Data Stream.

b. Required Version. 87a

c. Syntax.

```
      7 6 5 4 3 2 1 0        Field Name                    Type
     +===============+
   0 |               |       Red 0                         Byte
     +-             -+
   1 |               |       Green 0                       Byte
     +-             -+
   2 |               |       Blue 0                        Byte
     +-             -+
   3 |               |       Red 1                         Byte
     +-             -+
     |               |       Green 1                       Byte
     +-             -+
  up |               |
     +-   . . . .   -+       ...
  to |               |
     +-             -+
     |               |       Green 255                     Byte
     +-             -+
 767 |               |       Blue 255                      Byte
     +===============+
```

d. Extensions and Scope. The scope of this block is the entire Data Stream. This block cannot be modified by any extension.

e. Recommendation. None.

20. Image Descriptor.

a. Description. Each image in the Data Stream is composed of an Image Descriptor, an optional Local Color Table, and the image data. Each image must fit within the boundaries of the Logical Screen, as defined in the Logical Screen Descriptor.

The Image Descriptor contains the parameters necessary to process a table based image. The coordinates given in this block refer to coordinates within the Logical Screen, and are given in pixels. This block is a Graphic-Rendering Block, optionally preceded by one or more Control blocks such as the Graphic Control Extension, and may be optionally followed by a Local Color Table; the Image Descriptor is always followed by the image data.

This block is REQUIRED for an image. Exactly one Image Descriptor must be present per image in the Data Stream. An unlimited number of images may be present per Data Stream.

b. Required Version. 87a.

c. Syntax.

```
      7 6 5 4 3 2 1 0        Field Name                    Type
     +---------------+
   0 |               |       Image Separator               Byte
     +---------------+
   1 |               |       Image Left Position           Unsigned
     +-             -+
   2 |               |
     +---------------+
   3 |               |       Image Top Position            Unsigned
     +-             -+
   4 |               |
     +---------------+
   5 |               |       Image Width                   Unsigned
     +-             -+
   6 |               |
     +---------------+
   7  |               |       Image Height                  Unsigned
     +-             -+
   8  |               |
     +---------------+
   9  | | | |   |     |       <Packed Fields>               See below
     +---------------+

     <Packed Fields>  =      Local Color Table Flag        1 Bit
                             Interlace Flag                1 Bit
                             Sort Flag                     1 Bit
                             Reserved                      2 Bits
                             Size of Local Color Table     3 Bits
```

i) Image Separator - Identifies the beginning of an Image Descriptor. This field contains the fixed value 0x2C.

ii) Image Left Position - Column number, in pixels, of the left edge of the image, with respect to the left edge of the Logical Screen. Leftmost column of the Logical Screen is 0.

iii) Image Top Position - Row number, in pixels, of the top edge of the image with respect to the top edge of the Logical Screen. Top row of the Logical Screen is 0.

iv) Image Width - Width of the image in pixels.

v) Image Height - Height of the image in pixels.

vi) Local Color Table Flag - Indicates the presence of a Local Color Table immediately following this Image Descriptor. (This field is the most significant bit of the byte.)

```
Values :    0 -   Local Color Table is not present. Use
                  Global Color Table if available.
            1 -   Local Color Table present, and to follow
                  immediately after this Image Descriptor.
```

vii) Interlace Flag - Indicates if the image is interlaced. An image is interlaced in a four-pass interlace pattern; see Appendix E for details.

```
Values :    0 - Image is not interlaced.
            1 - Image is interlaced.
```

viii) Sort Flag - Indicates whether the Local Color Table is sorted. If the flag is set, the Local Color Table is sorted, in order of decreasing importance. Typically, the order would be decreasing frequency, with most frequent color first. This assists a decoder, with fewer available colors, in choosing the best subset of colors; the decoder may use an initial segment of the table to render the graphic.

```
Values :    0 -   Not ordered.
            1 -   Ordered by decreasing importance, most
                  important color first.
```

ix) Size of Local Color Table - If the Local Color Table Flag is set to 1, the value in this field is used to calculate the number of bytes contained in the Local Color Table. To determine that actual size of the color table, raise 2 to the value of the field + 1. This value should be 0 if there is no Local Color Table specified. (This field is made up of the 3 least significant bits of the byte.)

d. Extensions and Scope. The scope of this block is the Table-based Image Data Block that follows it. This block may be modified by the Graphic Control Extension.

e. Recommendation. None.

21. Local Color Table.

a. Description. This block contains a color table, which is a sequence of bytes representing red-green-blue color triplets. The Local Color Table is used by the image that immediately follows. Its presence is marked by the Local Color Table Flag being set to 1 in the Image Descriptor; if present, the Local Color Table immediately follows the Image Descriptor and contains a number of bytes equal to

3x2^(Size of Local Color Table+1).

If present, this color table temporarily becomes the active color table and the following image should be processed using it. This block is OPTIONAL; at most one Local Color Table may be present per Image Descriptor and its scope is the single image associated with the Image Descriptor that precedes it.

b. Required Version. 87a.

c. Syntax.

```
      7 6 5 4 3 2 1 0        Field Name                    Type
     +===============+
   0 |               |       Red 0                         Byte
     +-             -+
   1 |               |       Green 0                       Byte
     +-             -+
   2 |               |       Blue 0                        Byte
     +-             -+
   3 |               |       Red 1                         Byte
     +-             -+
     |               |       Green 1                       Byte
     +-             -+
  up |               |
     +-   . . . .   -+       ...
  to |               |
     +-             -+
     |               |       Green 255                     Byte
     +-             -+
 767 |               |       Blue 255                      Byte
     +===============+
```

d. Extensions and Scope. The scope of this block is the Table-based Image Data Block that immediately follows it. This block cannot be modified by any extension.

e. Recommendations. None.

22. Table Based Image Data.

a. Description. The image data for a table based image consists of a sequence of sub-blocks, of size at most 255 bytes each, containing an index into the active color table, for each pixel in the image. Pixel indices are in order of left to right and from top to bottom. Each index must be within the range of the size of the active color table, starting at 0. The sequence of indices is encoded using the LZW Algorithm with variable-length code, as described in Appendix F

b. Required Version. 87a.

c. Syntax. The image data format is as follows:

```
 7 6 5 4 3 2 1 0        Field Name                    Type
+---------------+
|               |       LZW Minimum Code Size         Byte
+---------------+

+===============+
|               |
/               /       Image Data                    Data Sub-blocks
|               |
+===============+
```

i) LZW Minimum Code Size. This byte determines the initial number of bits used for LZW codes in the image data, as described in Appendix F.

d. Extensions and Scope. This block has no scope, it contains raster data. Extensions intended to modify a Table-based image must appear before the corresponding Image Descriptor.

e. Recommendations. None.

23. Graphic Control Extension.

a. Description. The Graphic Control Extension contains parameters used when processing a graphic rendering block. The scope of this extension is the first graphic rendering block to follow. The extension contains only one data sub-block.

This block is OPTIONAL; at most one Graphic Control Extension may precede a graphic rendering block. This is the only limit to the number of Graphic Control Extensions that may be contained in a Data Stream.

b. Required Version. 89a.

c. Syntax.

```
     7 6 5 4 3 2 1 0        Field Name                    Type
    +---------------+
  0 |               |       Extension Introducer          Byte
    +---------------+
  1 |               |       Graphic Control Label         Byte
    +---------------+

    +---------------+
  0 |               |       Block Size                    Byte
    +---------------+
  1 |     |     | | |       <Packed Fields>               See below
    +---------------+
  2  |               |       Delay Time                    Unsigned
    +-             -+
  3  |               |
    +---------------+
  4  |               |       Transparent Color Index       Byte
    +---------------+

    +---------------+
  0  |               |       Block Terminator              Byte
    +---------------+


     <Packed Fields>  =     Reserved                      3 Bits
                            Disposal Method               3 Bits
                            User Input Flag               1 Bit
                            Transparent Color Flag        1 Bit
```

i) Extension Introducer - Identifies the beginning of an extension block. This field contains the fixed value 0x21.

ii) Graphic Control Label - Identifies the current block as a Graphic Control Extension. This field contains the fixed value 0xF9.

iii) Block Size - Number of bytes in the block, after the Block Size field and up to but not including the Block Terminator. This field contains the fixed value 4.

iv) Disposal Method - Indicates the way in which the graphic is to be treated after being displayed.

```
Values :    0 -   No disposal specified. The decoder is
                  not required to take any action.
            1 -   Do not dispose. The graphic is to be left
                  in place.
            2 -   Restore to background color. The area used by the
                  graphic must be restored to the background color.
            3 -   Restore to previous. The decoder is required to
                  restore the area overwritten by the graphic with
                  what was there prior to rendering the graphic.
         4-7 -    To be defined.
```

v) User Input Flag - Indicates whether or not user input is expected before continuing. If the flag is set, processing will continue when user input is entered. The nature of the User input is determined by the application (Carriage Return, Mouse Button Click, etc.).

```
Values :    0 -   User input is not expected.
            1 -   User input is expected.
```

When a Delay Time is used and the User Input Flag is set, processing will continue when user input is received or when the delay time expires, whichever occurs first.

vi) Transparency Flag - Indicates whether a transparency index is given in the Transparent Index field. (This field is the least significant bit of the byte.)

```
Values :    0 -   Transparent Index is not given.
            1 -   Transparent Index is given.
```

vii) Delay Time - If not 0, this field specifies the number of hundredths (1/100) of a second to wait before continuing with the processing of the Data Stream. The clock starts ticking immediately after the graphic is rendered. This field may be used in conjunction with the User Input Flag field.

viii) Transparency Index - The Transparency Index is such that when encountered, the corresponding pixel of the display device is not modified and processing goes on to the next pixel. The index is present if and only if the Transparency Flag is set to 1.

ix) Block Terminator - This zero-length data block marks the end of

the Graphic Control Extension.

d. Extensions and Scope. The scope of this Extension is the graphic rendering block that follows it; it is possible for other extensions to be present between this block and its target. This block can modify the Image Descriptor Block and the Plain Text Extension.

e. Recommendations.

i) Disposal Method - The mode Restore To Previous is intended to be used in small sections of the graphic; the use of this mode imposes severe demands on the decoder to store the section of the graphic that needs to be saved. For this reason, this mode should be used sparingly. This mode is not intended to save an entire graphic or large areas of a graphic; when this is the case, the encoder should make every attempt to make the sections of the graphic to be restored be separate graphics in the data stream. In the case where a decoder is not capable of saving an area of a graphic marked as Restore To Previous, it is recommended that a decoder restore to the background color.

ii) User Input Flag - When the flag is set, indicating that user input is expected, the decoder may sound the bell (0x07) to alert the user that input is being expected. In the absence of a specified Delay Time, the decoder should wait for user input indefinitely. It is recommended that the encoder not set the User Input Flag without a Delay Time specified.

24. Comment Extension.

a. Description. The Comment Extension contains textual information which is not part of the actual graphics in the GIF Data Stream. It is suitable for including comments about the graphics, credits, descriptions or any other type of non-control and non-graphic data. The Comment Extension may be ignored by the decoder, or it may be saved for later processing; under no circumstances should a Comment Extension disrupt or interfere with the processing of the Data Stream.

This block is OPTIONAL; any number of them may appear in the Data Stream.

b. Required Version. 89a.

c. Syntax.

```
      7 6 5 4 3 2 1 0        Field Name                    Type
     +---------------+
   0 |               |       Extension Introducer          Byte
     +---------------+
   1 |               |       Comment Label                 Byte
     +---------------+

     +===============+
     |               |
   N |               |       Comment Data                  Data Sub-blocks
     |               |
     +===============+

     +---------------+
   0 |               |       Block Terminator              Byte
     +---------------+
```

i) Extension Introducer - Identifies the beginning of an extension block. This field contains the fixed value 0x21.

ii) Comment Label - Identifies the block as a Comment Extension. This field contains the fixed value 0xFE.

iii) Comment Data - Sequence of sub-blocks, each of size at most 255 bytes and at least 1 byte, with the size in a byte preceding the data. The end of the sequence is marked by the Block Terminator.

iv) Block Terminator - This zero-length data block marks the end of the Comment Extension.

d. Extensions and Scope. This block does not have scope. This block cannot be modified by any extension.

e. Recommendations.

i) Data - This block is intended for humans. It should contain text using the 7-bit ASCII character set. This block should not be used to store control information for custom processing.

ii) Position - This block may appear at any point in the Data Stream at which a block can begin; however, it is recommended that Comment Extensions do not interfere with Control or Data blocks; they should be located at the beginning or at the end of the Data Stream to the extent possible.

25. Plain Text Extension.

a. Description. The Plain Text Extension contains textual data and the parameters necessary to render that data as a graphic, in a simple form. The textual data will be encoded with the 7-bit printable ASCII characters. Text data are rendered using a grid of character cells defined by the parameters in the block fields. Each character is rendered in an individual cell. The textual data in this block is to be rendered as mono-spaced characters, one character per cell, with a best fitting font and size. For further information, see the section on Recommendations below. The data characters are taken sequentially from the data portion of the block and rendered within a cell, starting with the upper left cell in the grid and proceeding from left to right and from top to bottom. Text data is rendered until the end of data is reached or the character grid is filled. The Character Grid contains an integral number of cells; in the case that the cell dimensions do not allow for an integral number, fractional cells must be discarded; an encoder must be careful to specify the grid dimensions accurately so that this does not happen. This block requires a Global Color Table to be available; the colors used by this block reference the Global Color Table in the Stream if there is one, or the Global Color Table from a previous Stream, if one was saved. This block is a graphic rendering block, therefore it may be modified by a Graphic Control Extension. This block is OPTIONAL; any number of them may appear in the Data Stream.

b. Required Version. 89a.

c. Syntax.

```
      7 6 5 4 3 2 1 0        Field Name                    Type
     +---------------+
   0 |               |       Extension Introducer          Byte
     +---------------+
   1 |               |       Plain Text Label              Byte
     +---------------+

     +---------------+
   0 |               |       Block Size                    Byte
     +---------------+
   1 |               |       Text Grid Left Position       Unsigned
     +-             -+
   2 |               |
     +---------------+
   3 |               |       Text Grid Top Position        Unsigned
     +-             -+
   4 |               |
     +---------------+
   5 |               |       Text Grid Width               Unsigned
     +-             -+
   6 |               |
     +---------------+
   7 |               |       Text Grid Height              Unsigned
     +-             -+
   8 |               |
     +---------------+
   9 |               |       Character Cell Width          Byte
     +---------------+
  10 |               |       Character Cell Height         Byte
     +---------------+
  11 |               |       Text Foreground Color Index   Byte
     +---------------+
  12 |               |       Text Background Color Index   Byte
     +---------------+

     +===============+
     |               |
   N |               |       Plain Text Data               Data Sub-blocks
     |               |
     +===============+

     +---------------+
   0 |               |       Block Terminator              Byte
     +---------------+
```

i) Extension Introducer - Identifies the beginning of an extension block. This field contains the fixed value 0x21.

ii) Plain Text Label - Identifies the current block as a Plain Text Extension. This field contains the fixed value 0x01.

iii) Block Size - Number of bytes in the extension, after the Block Size field and up to but not including the beginning of the data portion. This field contains the fixed value 12.

iv) Text Grid Left Position - Column number, in pixels, of the left edge of the text grid, with respect to the left edge of the Logical Screen.

v) Text Grid Top Position - Row number, in pixels, of the top edge of the text grid, with respect to the top edge of the Logical Screen.

vi) Image Grid Width - Width of the text grid in pixels.

vii) Image Grid Height - Height of the text grid in pixels.

viii) Character Cell Width - Width, in pixels, of each cell in the grid.

ix) Character Cell Height - Height, in pixels, of each cell in the grid.

x) Text Foreground Color Index - Index into the Global Color Table to be used to render the text foreground.

xi) Text Background Color Index - Index into the Global Color Table to be used to render the text background.

xii) Plain Text Data - Sequence of sub-blocks, each of size at most 255 bytes and at least 1 byte, with the size in a byte preceding the data. The end of the sequence is marked by the Block Terminator.

xiii) Block Terminator - This zero-length data block marks the end of the Plain Text Data Blocks.

d. Extensions and Scope. The scope of this block is the Plain Text Data Block contained in it. This block may be modified by the Graphic Control Extension.

e. Recommendations. The data in the Plain Text Extension is assumed to be preformatted. The selection of font and size is left to the discretion of the decoder. If characters less than 0x20 or greater than 0xf7 are encountered, it is recommended that the decoder display a Space character (0x20). The encoder should use grid and cell dimensions such that an integral number of cells fit in the grid both horizontally as well as vertically. For broadest compatibility, character cell dimensions should be around 8x8 or 8x16 (width x height); consider an image for unusual sized text.

26. Application Extension.

a. Description. The Application Extension contains application-specific information; it conforms with the extension block syntax, as described below, and its block label is 0xFF.

b. Required Version. 89a.

c. Syntax.

```
      7 6 5 4 3 2 1 0        Field Name                    Type
     +---------------+
   0 |               |       Extension Introducer          Byte
     +---------------+
   1 |               |       Extension Label               Byte
     +---------------+

     +---------------+
   0 |               |       Block Size                    Byte
     +---------------+
   1 |               |
     +-             -+
  2  |               |
     +-             -+
  3  |               |       Application Identifier        8 Bytes
     +-             -+
  4  |               |
     +-             -+
  5  |               |
     +-             -+
  6  |               |
     +-             -+
  7  |               |
     +-             -+
  8  |               |
     +---------------+
  9  |               |
     +-             -+
 10  |               |       Appl. Authentication Code     3 Bytes
     +-             -+
 11  |               |
     +---------------+

     +===============+
     |               |
     |               |       Application Data              Data Sub-blocks
     |               |
     |               |
     +===============+

     +---------------+
  0  |               |       Block Terminator              Byte
     +---------------+
```

i) Extension Introducer - Defines this block as an extension. This field contains the fixed value 0x21.

ii) Application Extension Label - Identifies the block as an Application Extension. This field contains the fixed value 0xFF.

iii) Block Size - Number of bytes in this extension block, following the Block Size field, up to but not including the beginning of the Application Data. This field contains the fixed value 11.

iv) Application Identifier - Sequence of eight printable ASCII characters used to identify the application owning the Application Extension.

v) Application Authentication Code - Sequence of three bytes used to authenticate the Application Identifier. An Application program may use an algorithm to compute a binary code that uniquely identifies it as the application owning the Application Extension.

d. Extensions and Scope. This block does not have scope. This block cannot be modified by any extension.

e. Recommendation. None.

27. Trailer.

a. Description. This block is a single-field block indicating the end of the GIF Data Stream. It contains the fixed value 0x3B.

b. Required Version. 87a.

c. Syntax.

```
      7 6 5 4 3 2 1 0        Field Name                    Type
     +---------------+
  0  |               |       GIF Trailer                   Byte
     +---------------+
```

d. Extensions and Scope. This block does not have scope, it terminates the GIF Data Stream. This block may not be modified by any extension.

e. Recommendations. None.

Appendix
A. Quick Reference Table.

Block Name	Required	Label	Ext.	Vers.
Application Extension	Opt. (*)	0xFF (255)	yes	89a
Comment Extension	Opt. (*)	0xFE (254)	yes	89a
Global Color Table	Opt. (1)	none	no	87a
Graphic Control Extension	Opt. (*)	0xF9 (249)	yes	89a
Header	Req. (1)	none	no	N/A
Image Descriptor	Opt. (*)	0x2C (044)	no	87a (89a)
Local Color Table	Opt. (*)	none	no	87a
Logical Screen Descriptor	Req. (1)	none	no	87a (89a)
Plain Text Extension	Opt. (*)	0x01 (001)	yes	89a
Trailer	Req. (1)	0x3B (059)	no	87a
Unlabeled Blocks				
Header	Req. (1)	none	no	N/A
Logical Screen Descriptor	Req. (1)	none	no	87a (89a)
Global Color Table	Opt. (1)	none	no	87a
Local Color Table	Opt. (*)	none	no	87a
Graphic-Rendering Blocks				
Plain Text Extension	Opt. (*)	0x01 (001)	yes	89a
Image Descriptor	Opt. (*)	0x2C (044)	no	87a (89a)
Control Blocks				
Graphic Control Extension	Opt. (*)	0xF9 (249)	yes	89a

Special Purpose Blocks				
Trailer	Req. (1)	0x3B (059)	no	87a
Comment Extension	Opt. (*)	0xFE (254)	yes	89a
Application Extension	Opt. (*)	0xFF (255)	yes	89a

legend:	(1)	if present, at most one occurrence
	(*)	zero or more occurrences
	(+)	one or more occurrences

Notes : The Header is not subject to Version Numbers.
(89a) The Logical Screen Descriptor and the Image Descriptor retained their syntax from version 87a to version 89a, but some fields reserved under version 87a are used under version 89a.

Appendix
B. GIF Grammar.

A Grammar is a form of notation to represent the sequence in which certain objects form larger objects. A grammar is also used to represent the number of objects that can occur at a given position. The grammar given here represents the sequence of blocks that form the GIF Data Stream. A grammar is given by listing its rules. Each rule consists of the left-hand side, followed by some form of equals sign, followed by the right-hand side. In a rule, the right-hand side describes how the left-hand side is defined. The right-hand side consists of a sequence of entities, with the possible presence of special symbols. The following legend defines the symbols used in this grammar for GIF.

Legend:	<>	grammar word
	::=	defines symbol
	*	zero or more occurrences
	+	one or more occurrences
	|	alternate element
	[]	optional element

Example:

```
<GIF Data Stream> ::= Header <Logical Screen> <Data>* Trailer
```

This rule defines the entity <GIF Data Stream> as follows. It must begin with a Header. The Header is followed by an entity called Logical Screen, which is defined below by another rule. The Logical Screen is followed by the entity Data, which is also defined below by another rule. Finally, the entity Data is followed by the Trailer. Since there is no rule defining the Header or the Trailer, this means that these blocks are defined in the document. The entity Data has a special symbol (*) following it which means that, at this position, the entity Data may be repeated any number of times, including 0 times. For further reading on this subject, refer to a standard text on Programming Languages.

The Grammar.

```
<GIF Data Stream> ::=     Header <Logical Screen> <Data>* Trailer

<Logical Screen> ::=      Logical Screen Descriptor [Global Color Table]

<Data> ::=                <Graphic Block>  |
                          <Special-Purpose Block>

<Graphic Block> ::=       [Graphic Control Extension] <Graphic-Rendering Block>

<Graphic-Rendering Block> ::=  <Table-Based Image>  |
                               Plain Text Extension

<Table-Based Image> ::=   Image Descriptor [Local Color Table] Image Data

<Special-Purpose Block> ::=    Application Extension  |
                               Comment Extension
```

NOTE : The grammar indicates that it is possible for a GIF Data Stream to contain the Header, the Logical Screen Descriptor, a Global Color Table and the GIF Trailer. This special case is used to load a GIF decoder with a Global Color Table, in preparation for subsequent Data Streams without color tables at all.

Appendix
C. Glossary.

Active Color Table - Color table used to render the next graphic. If the next graphic is an image which has a Local Color Table associated with it, the active color table becomes the Local Color Table associated with that image. If the next graphic is an image without a Local Color Table, or a Plain Text Extension, the active color table is the Global Color Table associated with the Data Stream, if there is one; if there is no Global Color Table in the Data Stream, the active color table is a color table saved from a previous Data Stream, or one supplied by the decoder.

Block - Collection of bytes forming a protocol unit. In general, the term includes labeled and unlabeled blocks, as well as Extensions.

Data Stream - The GIF Data Stream is composed of blocks and sub-blocks representing images and graphics, together with control information to render them on a display device. All control and data blocks in the Data Stream must follow the Header and must precede the Trailer.

Decoder - A program capable of processing a GIF Data Stream to render the images and graphics contained in it.

Encoder - A program capable of capturing and formatting image and graphic raster data, following the definitions of the Graphics Interchange Format.

Extension - A protocol block labeled by the Extension Introducer 0x21.

Extension Introducer - Label (0x21) defining an Extension.

Graphic - Data which can be rendered on the screen by virtue of some algorithm. The term graphic is more general than the term image; in addition to images, the term graphic also includes data such as text, which is rendered using character bit-maps.

Image - Data representing a picture or a drawing; an image is represented by an array of pixels called the raster of the image.

Raster - Array of pixel values representing an image.

Appendix
D. Conventions.

Animation - The Graphics Interchange Format is not intended as a platform for animation, even though it can be done in a limited way.

Byte Ordering - Unless otherwise stated, multi-byte numeric fields are ordered with the Least Significant Byte first.

Color Indices - Color indices always refer to the active color table, either the Global Color Table or the Local Color Table.

Color Order - Unless otherwise stated, all triple-component RGB color values are specified in Red-Green-Blue order.

Color Tables - Both color tables, the Global and the Local, are optional; if present, the Global Color Table is to be used with every image in the Data Stream for which a Local Color Table is not given; if present, a Local Color Table overrides the Global Color Table. However, if neither color table is present, the application program is free to use an arbitrary color table. If the graphics in several Data Streams are related and all use the same color table, an encoder could place the color table as the Global Color Table in the first Data Stream and leave subsequent Data Streams without a Global Color Table or any Local Color Tables; in this way, the overhead for the table is eliminated. It is recommended that the decoder save the previous Global Color Table to be used with the Data Stream that follows, in case it does not contain either a Global Color Table or any Local Color Tables. In general, this allows the application program to use past color tables, significantly reducing transmission overhead.

Extension Blocks - Extensions are defined using the Extension Introducer code to mark the beginning of the block, followed by a block label, identifying the type of extension. Extension Codes are numbers in the range from 0x00 to 0xFF, inclusive. Special purpose extensions are transparent to the decoder and may be omitted when transmitting the Data Stream on-line. The GIF capabilities dialogue makes the provision for the receiver to request the transmission of all blocks; the default state in this regard is no transmission of Special purpose blocks.

Reserved Fields - All Reserved Fields are expected to have each bit set to zero (off).

Appendix
E. Interlaced Images.

The rows of an Interlaced images are arranged in the following order:

```
      Group 1 : Every 8th. row, starting with row 0.              (Pass 1)
      Group 2 : Every 8th. row, starting with row 4.              (Pass 2)
      Group 3 : Every 4th. row, starting with row 2.              (Pass 3)
      Group 4 : Every 2nd. row, starting with row 1.              (Pass 4)
```

The Following example illustrates how the rows of an interlaced image are ordered.

```
      Row Number                                        Interlace Pass

 0    -----------------------------------------       1
 1    -----------------------------------------                         4
 2    -----------------------------------------                   3
 3    -----------------------------------------                         4
 4    -----------------------------------------             2
 5    -----------------------------------------                         4
 6    -----------------------------------------                   3
 7    -----------------------------------------                         4
 8    -----------------------------------------       1
 9    -----------------------------------------                         4
 10   -----------------------------------------                   3
 11   -----------------------------------------                         4
 12   -----------------------------------------             2
 13   -----------------------------------------                         4
 14   -----------------------------------------                   3
 15   -----------------------------------------                         4
 16   -----------------------------------------       1
 17   -----------------------------------------                         4
 18   -----------------------------------------                   3
 19   -----------------------------------------                         4
```

Appendix
F. Variable-Length-Code LZW Compression.

The Variable-Length-Code LZW Compression is a variation of the Lempel-Ziv Compression algorithm in which variable-length codes are used to replace patterns detected in the original data. The algorithm uses a code or translation table constructed from the patterns encountered in the original data; each new pattern is entered into the table and its index is used to replace it in the compressed stream.

The compressor takes the data from the input stream and builds a code or translation table with the patterns as it encounters them; each new pattern is entered into the code table and its index is added to the output stream; when a pattern is encountered which had been detected since the last code table refresh, its index from the code table is put on the output stream, thus achieving the data compression. The expander takes input from the compressed data stream and builds the code or translation table from it; as the compressed data stream is processed, codes are used to index into the code table and the corresponding data is put on the decompressed output stream, thus achieving data decompression. The details of the algorithm are explained below. The Variable-Length-Code aspect of the algorithm is based on an initial code size (LZW-initial code size), which specifies the initial number of bits used for the compression codes. When the number of patterns detected by the compressor in the input stream exceeds the number of patterns encodable with the current number of bits, the number of bits per LZW code is increased by one.

The Raster Data stream that represents the actual output image can be represented as:

```
         7 6 5 4 3 2 1 0
        +---------------+
        | LZW code size |
        +---------------+

        +---------------+ ----+
        |  block size   |     |
        +---------------+     |
        |               |     +-- Repeated as many
        |  data bytes   |     |   times as necessary.
        |               |     |
        +---------------+ ----+

        . . .       . . . ------- The code that terminates the LZW
                                  compressed data must appear before
                                  Block Terminator.
        +---------------+
        |0 0 0 0 0 0 0 0|  Block Terminator
        +---------------+
```

The conversion of the image from a series of pixel values to a transmitted or stored character stream involves several steps. In brief these steps are:

1. Establish the Code Size - Define the number of bits needed to represent the actual data.

2. Compress the Data - Compress the series of image pixels to a series of

compression codes.

3. Build a Series of Bytes - Take the set of compression codes and convert to a string of 8-bit bytes.

4. Package the Bytes - Package sets of bytes into blocks preceded by character counts and output.

ESTABLISH CODE SIZE

The first byte of the Compressed Data stream is a value indicating the minimum number of bits required to represent the set of actual pixel values. Normally this will be the same as the number of color bits. Because of some algorithmic constraints however, black & white images which have one color bit must be indicated as having a code size of 2.
This code size value also implies that the compression codes must start out one bit longer.

COMPRESSION

The LZW algorithm converts a series of data values into a series of codes which may be raw values or a code designating a series of values. Using text characters as an analogy, the output code consists of a character or a code representing a string of characters.

The LZW algorithm used in GIF matches algorithmically with the standard LZW algorithm with the following differences:

1. A special Clear code is defined which resets all compression/decompression parameters and tables to a start-up state. The value of this code is 2**<code size>. For example if the code size indicated was 4 (image was 4 bits/pixel) the Clear code value would be 16 (10000 binary). The Clear code can appear at any point in the image data stream and therefore requires the LZW algorithm to process succeeding codes as if a new data stream was starting. Encoders should output a Clear code as the first code of each image data stream.

2. An End of Information code is defined that explicitly indicates the end of the image data stream. LZW processing terminates when this code is encountered. It must be the last code output by the encoder for an image. The value of this code is <Clear code>+1.

3. The first available compression code value is <Clear code>+2.

4. The output codes are of variable length, starting at <code size>+1 bits per code, up to 12 bits per code. This defines a maximum code value of 4095 (0xFFF). Whenever the LZW code value would exceed the current code length, the code length is increased by one. The packing/unpacking of these codes must then be altered to reflect the new code length.

BUILD 8-BIT BYTES

Because the LZW compression used for GIF creates a series of variable length codes, of between 3 and 12 bits each, these codes must be reformed into a series of 8-bit bytes that will be the characters actually stored or transmitted. This provides additional compression of the image. The codes are formed into a stream of bits as if they were packed right to left and then

picked off 8 bits at a time to be output.

Assuming a character array of 8 bits per character and using 5 bit codes to be packed, an example layout would be similar to:

```
     +---------------+
  0  |               |    bbbaaaaa
     +---------------+
  1  |               |    dcccccbb
     +---------------+
 2 |               |    eeeedddd
   +---------------+
 3 |               |    ggfffffe
   +---------------+
 4 |               |    hhhhhggg
   +---------------+
         . . .
   +---------------+
 N |               |
   +---------------+
```

Note that the physical packing arrangement will change as the number of bits per compression code change but the concept remains the same.

PACKAGE THE BYTES

Once the bytes have been created, they are grouped into blocks for output by preceding each block of 0 to 255 bytes with a character count byte. A block with a zero byte count terminates the Raster Data stream for a given image. These blocks are what are actually output for the GIF image. This block format has the side effect of allowing a decoding program the ability to read past the actual image data if necessary by reading block counts and then skipping over the data.

FURTHER READING

[1] Ziv, J. and Lempel, A. : "A Universal Algorithm for Sequential Data Compression", IEEE Transactions on Information Theory, May 1977.
[2] Welch, T. : "A Technique for High-Performance Data Compression", Computer, June 1984.
[3] Nelson, M.R. : "LZW Data Compression", Dr. Dobb's Journal, October 1989.

Appendix
G. On-line Capabilities Dialogue.

NOTE : This section is currently (10 July 1990) under revision; the information provided here should be used as general guidelines. Code written based on this information should be designed in a flexible way to accommodate any changes resulting from the revisions.

The following sequences are defined for use in mediating control between a GIF sender and GIF receiver over an interactive communications line. These sequences do not apply to applications that involve downloading of static GIF files and are not considered part of a GIF file.

GIF CAPABILITIES ENQUIRY

The GIF Capabilities Enquiry sequence is issued from a host and requests an interactive GIF decoder to return a response message that defines the graphics parameters for the decoder. This involves returning information about available screen sizes, number of bits/color supported and the amount of color detail supported. The escape sequence for the GIF Capabilities Enquiry is defined as:

```
ESC[>0g           0x1B 0x5B 0x3E 0x30 0x67
```

GIF CAPABILITIES RESPONSE

The GIF Capabilities Response message is returned by an interactive GIF decoder and defines the decoder's display capabilities for all graphics modes that are supported by the software. Note that this can also include graphics printers as well as a monitor screen. The general format of this message is:

```
#version;protocol{;dev, width, height, color-bits, color-res}...<CR>
```

```
'#'            GIF Capabilities Response identifier character.
version        GIF format version number;  initially '87a'.
protocol='0'   No end-to-end protocol supported by decoder Transfer as direct
               8-bit data stream.
protocol='1'   Can use CIS B+ error correction protocol to transfer GIF data
               interactively from the host directly to the display.
dev = '0'      Screen parameter set follows.
dev = '1'      Printer parameter set follows.
width          Maximum supported display width in pixels.
height         Maximum supported display height in pixels.
color-bits     Number of bits per pixel supported. The number of supported
               colors is therefore 2**color-bits.
color-res      Number of bits per color component supported in the hardware
               color palette. If color-res is '0' then no hardware palette
               table is available.
```

Note that all values in the GIF Capabilities Response are returned as ASCII decimal numbers and the message is terminated by a Carriage Return character.

The following GIF Capabilities Response message describes three standard IBM PC Enhanced Graphics Adapter configurations with no printer; the GIF data stream

can be processed within an error correcting protocol:

```
#87a;1;0,320,200,4,0;0,640,200,2,2;0,640,350,4,2<CR>
```

ENTER GIF GRAPHICS MODE

Two sequences are currently defined to invoke an interactive GIF decoder into action. The only difference between them is that different output media are selected. These sequences are:

```
ESC[>1g     Display GIF image on screen

                  0x1B 0x5B 0x3E 0x31 0x67
```

ESC[>2g   Display image directly to an attached graphics printer. The image may optionally be displayed on the screen as well.

```
                  0x1B 0x5B 0x3E 0x32 0x67
```

Note that the 'g' character terminating each sequence is in lowercase.

INTERACTIVE ENVIRONMENT

The assumed environment for the transmission of GIF image data from an interactive application is a full 8-bit data stream from host to micro. All

256 character codes must be transferrable. The establishing of an 8-bit data path for communications will normally be taken care of by the host application programs. It is however up to the receiving communications programs supporting GIF to be able to receive and pass on all 256 8-bit codes to the GIF decoder software.

Cover Sheet for the GIF89a Specification

DEFERRED CLEAR CODE IN LZW COMPRESSION

There has been confusion about where clear codes can be found in the data stream. As the specification says, they may appear at anytime. There is not a requirement to send a clear code when the string table is full.

It is the encoder's decision as to when the table should be cleared. When the table is full, the encoder can chose to use the table as is, making no changes to it until the encoder chooses to clear it. The encoder during this time sends out codes that are of the maximum Code Size.

As we can see from the above, when the decoder's table is full, it must not change the table until a clear code is received. The Code Size is that of the maximum Code Size. Processing other than this is done normally.

Because of a large base of decoders that do not handle the decompression in this manner, we ask developers of GIF encoding software to NOT implement this feature until at least January 1991 and later if they see that their particular market is not ready for it. This will give developers of GIF decoding software time to implement this feature and to get it into the hands of their clients before the decoders start "breaking" on the new GIF's. It is not required that encoders change their software to take advantage of the deferred clear code, but it is for decoders.

APPLICATION EXTENSION BLOCK - APPLICATION IDENTIFIER

There will be a Courtesy Directory file located on CompuServe in the PICS forum. This directory will contain Application Identifiers for Application Extension Blocks that have been used by developers of GIF applications. This file is intended to help keep developers that wish to create Application Extension Blocks from using the same Application Identifiers. This is not an official directory; it is for voluntary participation only and does not guarantee that someone will not use the same identifier.

E-Mail can be sent to Larry Wood (forum manager of PICS) indicating the request for inclusion in this file with an identifier.

GRAPHICS INTERCHANGE FORMAT(sm)

Version 89a

Table of Contents

Disclaimer... 1

Foreword... 1

Licensing.. 1

About the Document... 2

General Description... 2

Version Numbers... 2

The Encoder... 3

The Decoder... 3

Compliance.. 3

About Recommendations... 4

About Color Tables.. 4

Blocks, Extensions and Scope.. 4

Block Sizes... 5

Using GIF as an embedded protocol... 5

Data Sub-blocks... 5

Block Terminator.. 6

Header.. 7

Logical Screen Descriptor... 8

Global Color Table.. 10

Image Descriptor.. 11

Local Color Table... 13

Table Based Image Data.. 14

Graphic Control Extension... 15

Comment Extension... 17

Plain Text Extension.. 18

Application Extension... 21

Trailer... 23

Quick Reference Table... 24

GIF Grammar... 25

Glossary.. 27

Conventions... 28

Interlaced Images... 29

Variable-Length-Code LZW Compression...................................... 30

On-line Capabilities Dialogue... 33

1. Disclaimer.

The information provided herein is subject to change without notice. In no event will CompuServe Incorporated be liable for damages, including any loss of revenue, loss of profits or other incidental or consequential damages arising out of the use or inability to use the information; CompuServe Incorporated makes no claim as to the suitability of the information.

2. Foreword.

This document defines the Graphics Interchange Format(sm). The specification given here defines version 89a, which is an extension of version 87a.

The Graphics Interchange Format(sm) as specified here should be considered complete; any deviation from it should be considered invalid, including but not limited to, the use of reserved or undefined fields within control or data blocks, the inclusion of extraneous data within or between blocks, the use of methods or algorithms not specifically listed as part of the format, etc. In general, any and all deviations, extensions or modifications not specified in this document should be considered to be in violation of the format and should be avoided.

3. Licensing.

The Graphics Interchange Format(c) is the copyright property of CompuServe Incorporated. Only CompuServe Incorporated is authorized to define, redefine, enhance, alter, modify or change in any way the definition of the format.

CompuServe Incorporated hereby grants a limited, non-exclusive, royalty-free license for the use of the Graphics Interchange Format(sm) in computer software; computer software utilizing GIF(sm) must acknowledge ownership of the Graphics Interchange Format and its Service Mark by CompuServe Incorporated, in User and Technical Documentation. Computer software utilizing GIF, which is distributed or may be distributed without User or Technical Documentation must display to the screen or printer a message acknowledging ownership of the Graphics Interchange Format and the Service Mark by CompuServe Incorporated; in this case, the acknowledgement may be displayed in an opening screen or leading banner, or a closing screen or trailing banner. A message such as the following may be used:

> "The Graphics Interchange Format(c) is the Copyright property of CompuServe Incorporated. GIF(sm) is a Service Mark property of CompuServe Incorporated."

For further information, please contact :

CompuServe Incorporated
Graphics Technology Department
5000 Arlington Center Boulevard
Columbus, Ohio 43220
U. S. A.

CompuServe Incorporated maintains a mailing list with all those individuals and organizations who wish to receive copies of this document when it is corrected or revised. This service is offered free of charge; please provide us with your mailing address.

4. About the Document.

This document describes in detail the definition of the Graphics Interchange Format. This document is intended as a programming reference; it is recommended that the entire document be read carefully before programming, because of the interdependence of the various parts. There is an individual section for each of the Format blocks. Within each section, the sub-section labeled Required Version refers to the version number that an encoder will have to use if the corresponding block is used in the Data Stream. Within each section, a diagram describes the individual fields in the block; the diagrams are drawn vertically; top bytes in the diagram appear first in the Data Stream. Bits within a byte are drawn most significant on the left end. Multi-byte numeric fields are ordered Least Significant Byte first. Numeric constants are represented as Hexadecimal numbers, preceded by "0x". Bit fields within a byte are described in order from most significant bits to least significant bits.

5. General Description.

The Graphics Interchange Format(sm) defines a protocol intended for the on-line transmission and interchange of raster graphic data in a way that is independent of the hardware used in their creation or display.

The Graphics Interchange Format is defined in terms of blocks and sub-blocks which contain relevant parameters and data used in the reproduction of a graphic. A GIF Data Stream is a sequence of protocol blocks and sub-blocks representing a collection of graphics. In general, the graphics in a Data

Stream are assumed to be related to some degree, and to share some control information; it is recommended that encoders attempt to group together related graphics in order to minimize hardware changes during processing and to minimize control information overhead. For the same reason, unrelated graphics or graphics which require resetting hardware parameters should be encoded separately to the extent possible.

A Data Stream may originate locally, as when read from a file, or it may originate remotely, as when transmitted over a data communications line. The Format is defined with the assumption that an error-free Transport Level Protocol is used for communications; the Format makes no provisions for error-detection and error-correction.

The GIF Data Stream must be interpreted in context, that is, the application program must rely on information external to the Data Stream to invoke the decoder process.

6. Version Numbers.

The version number in the Header of a Data Stream is intended to identify the minimum set of capabilities required of a decoder in order to fully process the Data Stream. An encoder should use the earliest possible version number that includes all the blocks used in the Data Stream. Within each block section in this document, there is an entry labeled Required Version which specifies the earliest version number that includes the corresponding block. The encoder should make every attempt to use the earliest version number covering all the blocks in the Data Stream; the unnecessary use of later version numbers will hinder processing by some decoders.

7. The Encoder.

The Encoder is the program used to create a GIF Data Stream. From raster data and other information, the encoder produces the necessary control and data blocks needed for reproducing the original graphics.

The encoder has the following primary responsibilities.

- Include in the Data Stream all the necessary information to reproduce the graphics.

- Insure that a Data Stream is labeled with the earliest possible Version Number that will cover the definition of all the blocks in it; this is to ensure that the largest number of decoders can process the Data Stream.

- Ensure encoding of the graphics in such a way that the decoding process is optimized. Avoid redundant information as much as possible.

- To the extent possible, avoid grouping graphics which might require resetting hardware parameters during the decoding process.

- Set to zero (off) each of the bits of each and every field designated as reserved. Note that some fields in the Logical Screen Descriptor and the Image Descriptor were reserved under Version 87a, but are used under version 89a.

8. The Decoder.

The Decoder is the program used to process a GIF Data Stream. It processes the

Data Stream sequentially, parsing the various blocks and sub-blocks, using the control information to set hardware and process parameters and interpreting the data to render the graphics.

The decoder has the following primary responsibilities.

- Process each graphic in the Data Stream in sequence, without delays other than those specified in the control information.

- Set its hardware parameters to fit, as closely as possible, the control information contained in the Data Stream.

9. Compliance.

An encoder or a decoder is said to comply with a given version of the Graphics Interchange Format if and only if it fully conforms with and correctly implements the definition of the standard associated with that version. An

encoder or a decoder may be compliant with a given version number and not compliant with some subsequent version.

10. About Recommendations.

Each block section in this document contains an entry labeled Recommendation; this section lists a set of recommendations intended to guide and organize the use of the particular blocks. Such recommendations are geared towards making the functions of encoders and decoders more efficient, as well as making optimal use of the communications bandwidth. It is advised that these recommendations be followed.

11. About Color Tables.

The GIF format utilizes color tables to render raster-based graphics. A color table can have one of two different scopes: global or local. A Global Color Table is used by all those graphics in the Data Stream which do not have a Local Color Table associated with them. The scope of the Global Color Table is the entire Data Stream. A Local Color Table is always associated with the graphic that immediately follows it; the scope of a Local Color Table is limited to that single graphic. A Local Color Table supersedes a Global Color Table, that is, if a Data Stream contains a Global Color Table, and an image has a Local Color Table associated with it, the decoder must save the Global Color Table, use the Local Color Table to render the image, and then restore the Global Color Table. Both types of color tables are optional, making it possible for a Data Stream to contain numerous graphics without a color table at all. For this reason, it is recommended that the decoder save the last Global Color Table used until another Global Color Table is encountered. In this way, a Data Stream which does not contain either a Global Color Table or a Local Color Table may be processed using the last Global Color Table saved. If a Global Color Table from a previous Stream is used, that table becomes the Global Color Table of the present Stream. This is intended to reduce the overhead incurred by color tables. In particular, it is recommended that an encoder use only one Global Color Table if all the images in related Data Streams can be rendered with the same table. If no color table is available at all, the decoder is free to use a system color table or a table of its own. In that case, the decoder may use a color table with as many colors as its hardware is able to support; it is recommended that such a table have black and white as its first two entries, so that monochrome images can be rendered adequately.

The Definition of the GIF Format allows for a Data Stream to contain only the Header, the Logical Screen Descriptor, a Global Color Table and the GIF

Trailer. Such a Data Stream would be used to load a decoder with a Global Color Table, in preparation for subsequent Data Streams without a color table at all.

12. Blocks, Extensions and Scope.

Blocks can be classified into three groups : Control, Graphic-Rendering and Special Purpose. Control blocks, such as the Header, the Logical Screen Descriptor, the Graphic Control Extension and the Trailer, contain information used to control the process of the Data Stream or information used in setting hardware parameters. Graphic-Rendering blocks such as the Image Descriptor and

the Plain Text Extension contain information and data used to render a graphic on the display device. Special Purpose blocks such as the Comment Extension and the Application Extension are neither used to control the process of the Data Stream nor do they contain information or data used to render a graphic on the display device. With the exception of the Logical Screen Descriptor and the Global Color Table, whose scope is the entire Data Stream, all other Control blocks have a limited scope, restricted to the Graphic-Rendering block that follows them. Special Purpose blocks do not delimit the scope of any Control blocks; Special Purpose blocks are transparent to the decoding process. Graphic-Rendering blocks and extensions are used as scope delimiters for Control blocks and extensions. The labels used to identify labeled blocks fall into three ranges : 0x00-0x7F (0-127) are the Graphic Rendering blocks, excluding the Trailer (0x3B); 0x80-0xF9 (128-249) are the Control blocks; 0xFA-0xFF (250-255) are the Special Purpose blocks. These ranges are defined so that decoders can handle block scope by appropriately identifying block labels, even when the block itself cannot be processed.

13. Block Sizes.

The Block Size field in a block, counts the number of bytes remaining in the block, not counting the Block Size field itself, and not counting the Block Terminator, if one is to follow. Blocks other than Data Blocks are intended to be of fixed length; the Block Size field is provided in order to facilitate skipping them, not to allow their size to change in the future. Data blocks and sub-blocks are of variable length to accommodate the amount of data.

14. Using GIF as an embedded protocol.

As an embedded protocol, GIF may be part of larger application protocols, within which GIF is used to render graphics. In such a case, the application protocol could define a block within which the GIF Data Stream would be contained. The application program would then invoke a GIF decoder upon encountering a block of type GIF. This approach is recommended in favor of using Application Extensions, which become overhead for all other applications that do not process them. Because a GIF Data Stream must be processed in context, the application must rely on some means of identifying the GIF Data Stream outside of the Stream itself.

15. Data Sub-blocks.

a. Description. Data Sub-blocks are units containing data. They do not have a label, these blocks are processed in the context of control blocks, wherever data blocks are specified in the format. The first byte of the Data sub-block indicates the number of data bytes to follow. A data sub-block may contain from 0 to 255 data bytes. The size of the block does not account for the size byte itself, therefore, the empty sub-block is one whose size field contains 0x00.

b. Required Version. 87a.

c. Syntax.

```
      7 6 5 4 3 2 1 0        Field Name                    Type
     +---------------+
   0 |               |       Block Size                    Byte
     +---------------+
   1 |               |
     +-             -+
   2 |               |
     +-             -+
   3 |               |
     +-             -+
     |               |       Data Values                   Byte
     +-             -+
  up |               |
     +-   . . . .   -+
  to |               |
     +-             -+
     |               |
     +-             -+
 255 |               |
     +---------------+
```

i) Block Size - Number of bytes in the Data Sub-block; the size must be within 0 and 255 bytes, inclusive.

ii) Data Values - Any 8-bit value. There must be exactly as many Data Values as specified by the Block Size field.

d. Extensions and Scope. This type of block always occurs as part of a larger unit. It does not have a scope of itself.

e. Recommendation. None.

16. Block Terminator.

a. Description. This zero-length Data Sub-block is used to terminate a sequence of Data Sub-blocks. It contains a single byte in the position of the Block Size field and does not contain data.

b. Required Version. 87a.

c. Syntax.

```
      7 6 5 4 3 2 1 0        Field Name                    Type
     +---------------+
   0 |               |       Block Size                    Byte
     +---------------+
```

i) Block Size - Number of bytes in the Data Sub-block; this field contains the fixed value 0x00.

ii) Data Values - This block does not contain any data.

d. Extensions and Scope. This block terminates the immediately preceding sequence of Data Sub-blocks. This block cannot be modified by any extension.

e. Recommendation. None.

17. Header.

a. Description. The Header identifies the GIF Data Stream in context. The Signature field marks the beginning of the Data Stream, and the Version field identifies the set of capabilities required of a decoder to fully process the Data Stream. This block is REQUIRED; exactly one Header must be present per Data Stream.

b. Required Version. Not applicable. This block is not subject to a version number. This block must appear at the beginning of every Data Stream.

c. Syntax.

```
      7 6 5 4 3 2 1 0        Field Name                    Type
     +---------------+
   0 |               |       Signature                     3 Bytes
     +-             -+
   1 |               |
     +-             -+
   2 |               |
     +---------------+
   3 |               |       Version                       3 Bytes
     +-             -+
   4 |               |
     +-             -+
   5 |               |
     +---------------+
```

i) Signature - Identifies the GIF Data Stream. This field contains the fixed value 'GIF'.

ii) Version - Version number used to format the data stream. Identifies the minimum set of capabilities necessary to a decoder to fully process the contents of the Data Stream.

```
Version Numbers as of 10 July 1990 :       "87a" - May 1987
                                           "89a" - July 1989
```

Version numbers are ordered numerically increasing on the first two digits starting with 87 (87,88,...,99,00,...,85,86) and alphabetically increasing on the third character (a,...,z).

iii) Extensions and Scope. The scope of this block is the entire Data Stream. This block cannot be modified by any extension.

d. Recommendations.

i) Signature - This field identifies the beginning of the GIF Data Stream; it is not intended to provide a unique signature for the identification of the data. It is recommended that the GIF Data Stream be identified externally by the application. (Refer to Appendix G for on-line identification of the GIF Data Stream.)

ii) Version - ENCODER : An encoder should use the earliest possible version number that defines all the blocks used in the Data Stream. When two or more Data Streams are combined, the latest of the individual version numbers should be used for the resulting Data Stream. DECODER : A decoder should attempt to process the data stream to the best of its ability; if it encounters a version number which it is not capable of processing fully, it should nevertheless, attempt to process the data stream to the best of its ability, perhaps after warning the user that the data may be incomplete.

18. Logical Screen Descriptor.

a. Description. The Logical Screen Descriptor contains the parameters necessary to define the area of the display device within which the images will be rendered. The coordinates in this block are given with respect to the top-left corner of the virtual screen; they do not necessarily refer to absolute coordinates on the display device. This implies that they could refer to window coordinates in a window-based environment or printer coordinates when a printer is used.

This block is REQUIRED; exactly one Logical Screen Descriptor must be present per Data Stream.

b. Required Version. Not applicable. This block is not subject to a version number. This block must appear immediately after the Header.

c. Syntax.

```
      7 6 5 4 3 2 1 0        Field Name                    Type
     +---------------+
  0  |               |       Logical Screen Width          Unsigned
     +-             -+
  1  |               |
     +---------------+
  2  |               |       Logical Screen Height         Unsigned
     +-             -+
  3  |               |
     +---------------+
  4  | |     | |     |       <Packed Fields>               See below
     +---------------+
  5  |               |       Background Color Index        Byte
     +---------------+
  6  |               |       Pixel Aspect Ratio            Byte
     +---------------+
```

```
     <Packed Fields>  =      Global Color Table Flag       1 Bit
                             Color Resolution              3 Bits
                             Sort Flag                     1 Bit
                             Size of Global Color Table    3 Bits
```

i) Logical Screen Width - Width, in pixels, of the Logical Screen where the images will be rendered in the displaying device.

ii) Logical Screen Height - Height, in pixels, of the Logical Screen where the images will be rendered in the displaying device.

iii) Global Color Table Flag - Flag indicating the presence of a Global Color Table; if the flag is set, the Global Color Table will immediately follow the Logical Screen Descriptor. This flag also selects the interpretation of the Background Color Index; if the flag is set, the value of the Background Color Index field should be used as the table index of the background color. (This field is the most significant bit of the byte.)

```
Values :    0 -   No Global Color Table follows, the Background
                  Color Index field is meaningless.
            1 -   A Global Color Table will immediately follow, the
                  Background Color Index field is meaningful.
```

iv) Color Resolution - Number of bits per primary color available to the original image, minus 1. This value represents the size of the entire palette from which the colors in the graphic were selected, not the number of colors actually used in the graphic. For example, if the value in this field is 3, then the palette of the original image had 4 bits per primary color available to create the image. This value should be set to indicate the richness of the original palette, even if not every color from the whole palette is available on the source machine.

v) Sort Flag - Indicates whether the Global Color Table is sorted. If the flag is set, the Global Color Table is sorted, in order of decreasing importance. Typically, the order would be decreasing frequency, with most frequent color first. This assists a decoder, with fewer available colors, in choosing the best subset of colors; the decoder may use an initial segment of the table to render the graphic.

```
Values :    0 -   Not ordered.
            1 -   Ordered by decreasing importance, most
                  important color first.
```

vi) Size of Global Color Table - If the Global Color Table Flag is set to 1, the value in this field is used to calculate the number of bytes contained in the Global Color Table. To determine that actual size of the color table, raise 2 to [the value of the field + 1]. Even if there is no Global Color Table specified, set this field according to the above formula so that decoders can choose the best graphics mode to display the stream in. (This field is made up of the 3 least significant bits of the byte.)

vii) Background Color Index - Index into the Global Color Table for

the Background Color. The Background Color is the color used for those pixels on the screen that are not covered by an image. If the Global Color Table Flag is set to (zero), this field should be zero and should be ignored.

viii) Pixel Aspect Ratio - Factor used to compute an approximation of the aspect ratio of the pixel in the original image. If the value of the field is not 0, this approximation of the aspect ratio is computed based on the formula:

Aspect Ratio = (Pixel Aspect Ratio + 15) / 64

The Pixel Aspect Ratio is defined to be the quotient of the pixel's width over its height. The value range in this field allows specification of the widest pixel of 4:1 to the tallest pixel of 1:4 in increments of 1/64th.

```
      Values :        0 -   No aspect ratio information is given.
                 1..255 -   Value used in the computation.
```

d. Extensions and Scope. The scope of this block is the entire Data Stream. This block cannot be modified by any extension.

e. Recommendations. None.

19. Global Color Table.

a. Description. This block contains a color table, which is a sequence of bytes representing red-green-blue color triplets. The Global Color Table is used by images without a Local Color Table and by Plain Text Extensions. Its presence is marked by the Global Color Table Flag being set to 1 in the Logical Screen Descriptor; if present, it immediately follows the Logical Screen Descriptor and contains a number of bytes equal to

3 x 2^(Size of Global Color Table+1).

This block is OPTIONAL; at most one Global Color Table may be present per Data Stream.

b. Required Version. 87a

c. Syntax.

```
      7 6 5 4 3 2 1 0        Field Name                    Type
     +================+
   0 |                |       Red 0                         Byte
     +-              -+
   1 |                |       Green 0                       Byte
     +-              -+
   2 |                |       Blue 0                        Byte
     +-              -+
   3 |                |       Red 1                         Byte
     +-              -+
     |                |       Green 1                       Byte
     +-              -+
  up |                |
     +-   . . . .    -+       ...
  to |                |
     +-              -+
     |                |       Green 255                     Byte
     +-              -+
 767 |                |       Blue 255                      Byte
```

```
     +===============+
```

d. Extensions and Scope. The scope of this block is the entire Data Stream. This block cannot be modified by any extension.

e. Recommendation. None.

20. Image Descriptor.

a. Description. Each image in the Data Stream is composed of an Image Descriptor, an optional Local Color Table, and the image data. Each image must fit within the boundaries of the Logical Screen, as defined in the Logical Screen Descriptor.

The Image Descriptor contains the parameters necessary to process a table based image. The coordinates given in this block refer to coordinates within the Logical Screen, and are given in pixels. This block is a Graphic-Rendering Block, optionally preceded by one or more Control blocks such as the Graphic Control Extension, and may be optionally followed by a Local Color Table; the Image Descriptor is always followed by the image data.

This block is REQUIRED for an image. Exactly one Image Descriptor must be present per image in the Data Stream. An unlimited number of images may be present per Data Stream.

b. Required Version. 87a.

c. Syntax.

```
      7 6 5 4 3 2 1 0        Field Name                    Type
     +---------------+
   0 |               |       Image Separator               Byte
     +---------------+
   1 |               |       Image Left Position           Unsigned
     +-             -+
   2 |               |
     +---------------+
   3 |               |       Image Top Position            Unsigned
     +-             -+
   4 |               |
     +---------------+
   5 |               |       Image Width                   Unsigned
     +-             -+
   6 |               |
     +---------------+
   7 |               |       Image Height                  Unsigned
     +-             -+
   8 |               |
     +---------------+
   9 | | | |   |     |       <Packed Fields>               See below
     +---------------+

     <Packed Fields>  =      Local Color Table Flag        1 Bit
                             Interlace Flag                1 Bit
                             Sort Flag                     1 Bit
```

Reserved	2 Bits
Size of Local Color Table	3 Bits

i) Image Separator - Identifies the beginning of an Image Descriptor. This field contains the fixed value 0x2C.

ii) Image Left Position - Column number, in pixels, of the left edge of the image, with respect to the left edge of the Logical Screen. Leftmost column of the Logical Screen is 0.

iii) Image Top Position - Row number, in pixels, of the top edge of the image with respect to the top edge of the Logical Screen. Top row of the Logical Screen is 0.

iv) Image Width - Width of the image in pixels.

v) Image Height - Height of the image in pixels.

vi) Local Color Table Flag - Indicates the presence of a Local Color Table immediately following this Image Descriptor. (This field is the most significant bit of the byte.)

Values :   0 -   Local Color Table is not present. Use
                  Global Color Table if available.
            1 -   Local Color Table present, and to follow
                  immediately after this Image Descriptor.

vii) Interlace Flag - Indicates if the image is interlaced. An image is interlaced in a four-pass interlace pattern; see Appendix E for details.

Values :   0 - Image is not interlaced.
            1 - Image is interlaced.

viii) Sort Flag - Indicates whether the Local Color Table is sorted. If the flag is set, the Local Color Table is sorted, in order of decreasing importance. Typically, the order would be decreasing frequency, with most frequent color first. This assists a decoder, with fewer available colors, in choosing the best subset of colors; the decoder may use an initial segment of the table to render the graphic.

Values :   0 -   Not ordered.
            1 -   Ordered by decreasing importance, most
                  important color first.

ix) Size of Local Color Table - If the Local Color Table Flag is set to 1, the value in this field is used to calculate the number of bytes contained in the Local Color Table. To determine that actual size of the color table, raise 2 to the value of the field + 1. This value should be 0 if there is no Local Color Table specified. (This field is made up of the 3 least significant bits of the byte.)

d. Extensions and Scope. The scope of this block is the Table-based Image Data Block that follows it. This block may be modified by the Graphic Control Extension.

e. Recommendation. None.

21. Local Color Table.

a. Description. This block contains a color table, which is a sequence of bytes representing red-green-blue color triplets. The Local Color Table is used by the image that immediately follows. Its presence is marked by the Local Color Table Flag being set to 1 in the Image Descriptor; if present, the Local Color Table immediately follows the Image Descriptor and contains a number of bytes equal to

3x2^(Size of Local Color Table+1).

If present, this color table temporarily becomes the active color table and the following image should be processed using it. This block is OPTIONAL; at most one Local Color Table may be present per Image Descriptor and its scope is the single image associated with the Image Descriptor that precedes it.

b. Required Version. 87a.

c. Syntax.

```
      7 6 5 4 3 2 1 0        Field Name                    Type
     +===============+
   0 |               |       Red 0                         Byte
     +-             -+
   1 |               |       Green 0                       Byte
     +-             -+
   2 |               |       Blue 0                        Byte
     +-             -+
   3 |               |       Red 1                         Byte
     +-             -+
     |               |       Green 1                       Byte
     +-             -+
  up |               |
     +-   . . . .   -+       ...
  to |               |
     +-             -+
     |               |       Green 255                     Byte
     +-             -+
 767 |               |       Blue 255                      Byte
     +===============+
```

d. Extensions and Scope. The scope of this block is the Table-based Image Data Block that immediately follows it. This block cannot be modified by any extension.

e. Recommendations. None.

22. Table Based Image Data.

a. Description. The image data for a table based image consists of a sequence of sub-blocks, of size at most 255 bytes each, containing an index into the active color table, for each pixel in the image. Pixel indices are in order of left to right and from top to bottom. Each index must be within the range of the size of the active color table, starting at 0. The sequence of indices is encoded using the LZW Algorithm with variable-length code, as described in Appendix F

b. Required Version. 87a.

c. Syntax. The image data format is as follows:

```
 7 6 5 4 3 2 1 0        Field Name                    Type
+---------------+
|               |       LZW Minimum Code Size         Byte
+---------------+

+===============+
|               |
/               /       Image Data                    Data Sub-blocks
|               |
+===============+
```

i) LZW Minimum Code Size. This byte determines the initial number of bits used for LZW codes in the image data, as described in Appendix F.

d. Extensions and Scope. This block has no scope, it contains raster data. Extensions intended to modify a Table-based image must appear before the corresponding Image Descriptor.

e. Recommendations. None.

23. Graphic Control Extension.

a. Description. The Graphic Control Extension contains parameters used when processing a graphic rendering block. The scope of this extension is the first graphic rendering block to follow. The extension contains only one data sub-block.

This block is OPTIONAL; at most one Graphic Control Extension may precede a graphic rendering block. This is the only limit to the number of Graphic Control Extensions that may be contained in a Data Stream.

b. Required Version. 89a.

c. Syntax.

```
     7 6 5 4 3 2 1 0        Field Name                    Type
    +---------------+
  0 |               |       Extension Introducer          Byte
    +---------------+
  1 |               |       Graphic Control Label         Byte
    +---------------+

    +---------------+
  0 |               |       Block Size                    Byte
    +---------------+
  1 |     |     | | |       <Packed Fields>               See below
    +---------------+
  2 |               |       Delay Time                    Unsigned
    +-             -+
  3 |               |
    +---------------+
  4 |               |       Transparent Color Index       Byte
    +---------------+

    +---------------+
  0 |               |       Block Terminator              Byte
    +---------------+
<Packed Fields>  =     Reserved                      3 Bits
                       Disposal Method               3 Bits
                       User Input Flag               1 Bit
                       Transparent Color Flag        1 Bit
```

i) Extension Introducer - Identifies the beginning of an extension block. This field contains the fixed value 0x21.

ii) Graphic Control Label - Identifies the current block as a Graphic Control Extension. This field contains the fixed value 0xF9.

iii) Block Size - Number of bytes in the block, after the Block Size field and up to but not including the Block Terminator. This field contains the fixed value 4.

iv) Disposal Method - Indicates the way in which the graphic is to be treated after being displayed.

```
Values :    0 -   No disposal specified. The decoder is
                  not required to take any action.
            1 -   Do not dispose. The graphic is to be left
                  in place.
            2 -   Restore to background color. The area used by the
                  graphic must be restored to the background color.
            3 -   Restore to previous. The decoder is required to
                  restore the area overwritten by the graphic with
                  what was there prior to rendering the graphic.
         4-7 -    To be defined.
```

v) User Input Flag - Indicates whether or not user input is expected before continuing. If the flag is set, processing will continue when user input is entered. The nature of the User input is determined by the application (Carriage Return, Mouse Button Click, etc.).

```
Values :    0 -   User input is not expected.
            1 -   User input is expected.
```

When a Delay Time is used and the User Input Flag is set, processing will continue when user input is received or when the delay time expires, whichever occurs first.

vi) Transparency Flag - Indicates whether a transparency index is given in the Transparent Index field. (This field is the least significant bit of the byte.)

```
Values :    0 -   Transparent Index is not given.
            1 -   Transparent Index is given.
```

vii) Delay Time - If not 0, this field specifies the number of hundredths (1/100) of a second to wait before continuing with the processing of the Data Stream. The clock starts ticking immediately after the graphic is rendered. This field may be used in conjunction with the User Input Flag field.

viii) Transparency Index - The Transparency Index is such that when encountered, the corresponding pixel of the display device is not modified and processing goes on to the next pixel. The index is present if and only if the Transparency Flag is set to 1.

ix) Block Terminator - This zero-length data block marks the end of the Graphic Control Extension.

d. Extensions and Scope. The scope of this Extension is the graphic rendering block that follows it; it is possible for other extensions to be present between this block and its target. This block can modify the Image Descriptor Block and the Plain Text Extension.

e. Recommendations.

i) Disposal Method - The mode Restore To Previous is intended to be used in small sections of the graphic; the use of this mode imposes severe demands on the decoder to store the section of the graphic that needs to be saved. For this reason, this mode should be used sparingly. This mode is not intended to save an entire graphic or large areas of a graphic; when this is the case, the encoder should make every attempt to make the sections of the graphic to be restored be separate graphics in the data stream. In the case where a decoder is not capable of saving an area of a graphic marked as Restore To Previous, it is recommended that a decoder restore to the background color.

ii) User Input Flag - When the flag is set, indicating that user input is expected, the decoder may sound the bell (0x07) to alert the user that input is being expected. In the absence of a specified Delay Time, the decoder should wait for user input indefinitely. It is recommended that the encoder not set the User Input Flag without a Delay Time specified.

24. Comment Extension.

a. Description. The Comment Extension contains textual information which is not part of the actual graphics in the GIF Data Stream. It is suitable for including comments about the graphics, credits, descriptions or any other type of non-control and non-graphic data. The Comment Extension may be ignored by the decoder, or it may be saved for later processing; under no circumstances should a Comment Extension disrupt or interfere with the processing of the Data Stream.

This block is OPTIONAL; any number of them may appear in the Data Stream.

b. Required Version. 89a.

c. Syntax.

```
      7 6 5 4 3 2 1 0        Field Name                    Type
     +---------------+
  0  |               |       Extension Introducer          Byte
     +---------------+
  1  |               |       Comment Label                 Byte
     +---------------+

     +===============+
     |               |
  N  |               |       Comment Data                  Data Sub-blocks
     |               |
     +===============+

     +---------------+
  0  |               |       Block Terminator              Byte
     +---------------+
```

i) Extension Introducer - Identifies the beginning of an extension block. This field contains the fixed value 0x21.

ii) Comment Label - Identifies the block as a Comment Extension. This field contains the fixed value 0xFE.

iii) Comment Data - Sequence of sub-blocks, each of size at most 255 bytes and at least 1 byte, with the size in a byte preceding the data. The end of the sequence is marked by the Block Terminator.

iv) Block Terminator - This zero-length data block marks the end of the Comment Extension.

d. Extensions and Scope. This block does not have scope. This block cannot be modified by any extension.

e. Recommendations.

i) Data - This block is intended for humans. It should contain text using the 7-bit ASCII character set. This block should not be used to store control information for custom processing.

ii) Position - This block may appear at any point in the Data Stream at which a block can begin; however, it is recommended that Comment Extensions do not interfere with Control or Data blocks; they should be located at the beginning or at the end of the Data Stream to the extent possible.

25. Plain Text Extension.

a. Description. The Plain Text Extension contains textual data and the parameters necessary to render that data as a graphic, in a simple form. The textual data will be encoded with the 7-bit printable ASCII characters. Text data are rendered using a grid of character cells

defined by the parameters in the block fields. Each character is rendered in an individual cell. The textual data in this block is to be rendered as mono-spaced characters, one character per cell, with a best fitting font and size. For further information, see the section on

Recommendations below. The data characters are taken sequentially from the data portion of the block and rendered within a cell, starting with the upper left cell in the grid and proceeding from left to right and from top to bottom. Text data is rendered until the end of data is reached or the character grid is filled. The Character Grid contains an integral number of cells; in the case that the cell dimensions do not allow for an integral number, fractional cells must be discarded; an encoder must be careful to specify the grid dimensions accurately so that this does not happen. This block requires a Global Color Table to be available; the colors used by this block reference the Global Color Table in the Stream if there is one, or the Global Color Table from a previous Stream, if one was saved. This block is a graphic rendering block, therefore it may be modified by a Graphic Control Extension. This block is OPTIONAL; any number of them may appear in the Data Stream.

b. Required Version. 89a.

c. Syntax.

```
      7 6 5 4 3 2 1 0        Field Name                    Type
     +---------------+
   0 |               |       Extension Introducer          Byte
     +---------------+
   1 |               |       Plain Text Label              Byte
     +---------------+

     +---------------+
   0 |               |       Block Size                    Byte
```

```
      +---------------+
   1  |               |       Text Grid Left Position       Unsigned
      +-             -+
   2  |               |
      +---------------+
   3  |               |       Text Grid Top Position        Unsigned
      +-             -+
   4  |               |
      +---------------+
   5  |               |       Text Grid Width               Unsigned
      +-             -+
   6  |               |
      +---------------+
   7  |               |       Text Grid Height              Unsigned
      +-             -+
   8  |               |
      +---------------+
   9  |               |       Character Cell Width          Byte
      +---------------+
  10  |               |       Character Cell Height         Byte
      +---------------+
  11  |               |       Text Foreground Color Index   Byte
      +---------------+
  12  |               |       Text Background Color Index   Byte
      +---------------+

      +===============+
      |               |
   N  |               |       Plain Text Data               Data Sub-blocks
      |               |
      +===============+

      +---------------+
   0  |               |       Block Terminator              Byte
      +---------------+
```

i) Extension Introducer - Identifies the beginning of an extension block. This field contains the fixed value 0x21.

ii) Plain Text Label - Identifies the current block as a Plain Text Extension. This field contains the fixed value 0x01.

iii) Block Size - Number of bytes in the extension, after the Block Size field and up to but not including the beginning of the data portion. This field contains the fixed value 12.

iv) Text Grid Left Position - Column number, in pixels, of the left edge of the text grid, with respect to the left edge of the Logical Screen.

v) Text Grid Top Position - Row number, in pixels, of the top edge of the text grid, with respect to the top edge of the Logical Screen.

vi) Image Grid Width - Width of the text grid in pixels.

vii) Image Grid Height - Height of the text grid in pixels.

viii) Character Cell Width - Width, in pixels, of each cell in the grid.

ix) Character Cell Height - Height, in pixels, of each cell in the grid.

x) Text Foreground Color Index - Index into the Global Color Table to be used to render the text foreground.

xi) Text Background Color Index - Index into the Global Color Table to be used to render the text background.

xii) Plain Text Data - Sequence of sub-blocks, each of size at most 255 bytes and at least 1 byte, with the size in a byte preceding the data. The end of the sequence is marked by the Block Terminator.

xiii) Block Terminator - This zero-length data block marks the end of the Plain Text Data Blocks.

d. Extensions and Scope. The scope of this block is the Plain Text Data Block contained in it. This block may be modified by the Graphic Control Extension.

e. Recommendations. The data in the Plain Text Extension is assumed to be preformatted. The selection of font and size is left to the discretion of the decoder. If characters less than 0x20 or greater than 0xf7 are encountered, it is recommended that the decoder display a Space character (0x20). The encoder should use grid and cell dimensions such that an integral number of cells fit in the grid both horizontally as well as vertically. For broadest compatibility, character cell dimensions should be around 8x8 or 8x16 (width x height); consider an image for unusual sized text.

26. Application Extension.

a. Description. The Application Extension contains application-specific information; it conforms with the extension block syntax, as described below, and its block label is 0xFF.

b. Required Version. 89a.

c. Syntax.

```
      7 6 5 4 3 2 1 0        Field Name                    Type
     +---------------+
   0 |               |       Extension Introducer          Byte
     +---------------+
   1 |               |       Extension Label               Byte
     +---------------+

     +---------------+
   0 |               |       Block Size                    Byte
     +---------------+
   1 |               |
     +-             -+
   2 |               |
     +-             -+
   3 |               |       Application Identifier        8 Bytes
     +-             -+
   4 |               |
     +-             -+
   5 |               |
     +-             -+
   6 |               |
     +-             -+
   7 |               |
      +-             -+
   8  |               |
      +---------------+
   9  |               |
      +-             -+
  10  |               |       Appl. Authentication Code     3 Bytes
      +-             -+
  11  |               |
      +---------------+

      +===============+
      |               |
      |               |       Application Data              Data Sub-blocks
      |               |
      |               |
      +===============+

      +---------------+
   0  |               |       Block Terminator              Byte
      +---------------+
```

i) Extension Introducer - Defines this block as an extension. This field contains the fixed value 0x21.

ii) Application Extension Label - Identifies the block as an Application Extension. This field contains the fixed value 0xFF.

iii) Block Size - Number of bytes in this extension block, following the Block Size field, up to but not including the beginning of the Application Data. This field contains the fixed value 11.

iv) Application Identifier - Sequence of eight printable ASCII characters used to identify the application owning the Application Extension.

v) Application Authentication Code - Sequence of three bytes used to authenticate the Application Identifier. An Application program may use an algorithm to compute a binary code that uniquely identifies it as the application owning the Application Extension.

d. Extensions and Scope. This block does not have scope. This block cannot be modified by any extension.

e. Recommendation. None.

27. Trailer.

a. Description. This block is a single-field block indicating the end of the GIF Data Stream. It contains the fixed value 0x3B.

b. Required Version. 87a.

c. Syntax.

```
       7 6 5 4 3 2 1 0        Field Name                    Type
      +---------------+
   0  |               |       GIF Trailer                   Byte
      +---------------+
```

d. Extensions and Scope. This block does not have scope, it terminates the GIF Data Stream. This block may not be modified by any extension.

e. Recommendations. None.

Appendix

A. Quick Reference Table.

Block Name	Required	Label	Ext.	Vers.
Application Extension	Opt. (*)	0xFF (255)	yes	89a
Comment Extension	Opt. (*)	0xFE (254)	yes	89a
Global Color Table	Opt. (1)	none	no	87a
Graphic Control Extension	Opt. (*)	0xF9 (249)	yes	89a
Header	Req. (1)	none	no	N/A
Image Descriptor	Opt. (*)	0x2C (044)	no	87a (89a)
Local Color Table	Opt. (*)	none	no	87a
Logical Screen Descriptor	Req. (1)	none	no	87a (89a)
Plain Text Extension	Opt. (*)	0x01 (001)	yes	89a
Trailer	Req. (1)	0x3B (059)	no	87a
Unlabeled Blocks				
Header	Req. (1)	none	no	N/A
Logical Screen Descriptor	Req. (1)	none	no	87a (89a)
Global Color Table	Opt. (1)	none	no	87a
Local Color Table	Opt. (*)	none	no	87a
Graphic-Rendering Blocks				
Plain Text Extension	Opt. (*)	0x01 (001)	yes	89a
Image Descriptor	Opt. (*)	0x2C (044)	no	87a (89a)
Control Blocks				
Graphic Control Extension	Opt. (*)	0xF9 (249)	yes	89a
Special Purpose Blocks				
Trailer	Req. (1)	0x3B (059)	no	87a
Comment Extension	Opt. (*)	0xFE (254)	yes	89a
Application Extension	Opt. (*)	0xFF (255)	yes	89a

legend:	(1)	if present, at most one occurrence
	(*)	zero or more occurrences
	(+)	one or more occurrences

Notes : The Header is not subject to Version Numbers.
(89a) The Logical Screen Descriptor and the Image Descriptor retained their

syntax from version 87a to version 89a, but some fields reserved under version 87a are used under version 89a.

Appendix
B. GIF Grammar.

A Grammar is a form of notation to represent the sequence in which certain objects form larger objects. A grammar is also used to represent the number of objects that can occur at a given position. The grammar given here represents the sequence of blocks that form the GIF Data Stream. A grammar is given by listing its rules. Each rule consists of the left-hand side, followed by some form of equals sign, followed by the right-hand side. In a rule, the right-hand side describes how the left-hand side is defined. The right-hand side consists of a sequence of entities, with the possible presence of special symbols. The following legend defines the symbols used in this grammar for GIF.

<table>
<tr><td>Legend:</td><td>&lt;&gt;</td><td>grammar word</td></tr>
<tr><td></td><td>::=</td><td>defines symbol</td></tr>
<tr><td></td><td>*</td><td>zero or more occurrences</td></tr>
<tr><td></td><td>+</td><td>one or more occurrences</td></tr>
<tr><td></td><td>|</td><td>alternate element</td></tr>
<tr><td></td><td>[]</td><td>optional element</td></tr>
</table>

Example:

```
<GIF Data Stream> ::= Header <Logical Screen> <Data>* Trailer
```

This rule defines the entity <GIF Data Stream> as follows. It must begin with a Header. The Header is followed by an entity called Logical Screen, which is defined below by another rule. The Logical Screen is followed by the entity Data, which is also defined below by another rule. Finally, the entity Data is followed by the Trailer. Since there is no rule defining the Header or the Trailer, this means that these blocks are defined in the document. The entity Data has a special symbol (*) following it which means that, at this position, the entity Data may be repeated any number of times, including 0 times. For further reading on this subject, refer to a standard text on Programming Languages.

The Grammar.

```
<GIF Data Stream> ::=     Header <Logical Screen> <Data>* Trailer

<Logical Screen> ::=      Logical Screen Descriptor [Global Color Table]

<Data> ::=                <Graphic Block>  |
                          <Special-Purpose Block>

<Graphic Block> ::=       [Graphic Control Extension] <Graphic-Rendering Block>
```

```
<Graphic-Rendering Block> ::=  <Table-Based Image>  |
                               Plain Text Extension

<Table-Based Image> ::=   Image Descriptor [Local Color Table] Image Data

<Special-Purpose Block> ::=    Application Extension  |
                               Comment Extension
```

NOTE : The grammar indicates that it is possible for a GIF Data Stream to contain the Header, the Logical Screen Descriptor, a Global Color Table and the GIF Trailer. This special case is used to load a GIF decoder with a Global Color Table, in preparation for subsequent Data Streams without color tables at all.

Appendix
C. Glossary.

Active Color Table - Color table used to render the next graphic. If the next graphic is an image which has a Local Color Table associated with it, the active color table becomes the Local Color Table associated with that image. If the next graphic is an image without a Local Color Table, or a Plain Text Extension, the active color table is the Global Color Table associated with the Data Stream, if there is one; if there is no Global Color Table in the Data Stream, the active color table is a color table saved from a previous Data Stream, or one supplied by the decoder.

Block - Collection of bytes forming a protocol unit. In general, the term includes labeled and unlabeled blocks, as well as Extensions.

Data Stream - The GIF Data Stream is composed of blocks and sub-blocks representing images and graphics, together with control information to render them on a display device. All control and data blocks in the Data Stream must follow the Header and must precede the Trailer.

Decoder - A program capable of processing a GIF Data Stream to render the images and graphics contained in it.

Encoder - A program capable of capturing and formatting image and graphic raster data, following the definitions of the Graphics Interchange Format.

Extension - A protocol block labeled by the Extension Introducer 0x21.

Extension Introducer - Label (0x21) defining an Extension.

Graphic - Data which can be rendered on the screen by virtue of some algorithm. The term graphic is more general than the term image; in addition to images, the term graphic also includes data such as text, which is rendered using character bit-maps.

Image - Data representing a picture or a drawing; an image is represented by an array of pixels called the raster of the image.

Raster - Array of pixel values representing an image.

Appendix
D. Conventions.

Animation - The Graphics Interchange Format is not intended as a platform for animation, even though it can be done in a limited way.

Byte Ordering - Unless otherwise stated, multi-byte numeric fields are ordered with the Least Significant Byte first.

Color Indices - Color indices always refer to the active color table, either the Global Color Table or the Local Color Table.

Color Order - Unless otherwise stated, all triple-component RGB color values are specified in Red-Green-Blue order.

Color Tables - Both color tables, the Global and the Local, are optional; if present, the Global Color Table is to be used with every image in the Data Stream for which a Local Color Table is not given; if present, a Local Color Table overrides the Global Color Table. However, if neither color table is present, the application program is free to use an arbitrary color table. If the graphics in several Data Streams are related and all use the same color table, an encoder could place the color table as the Global Color Table in the first Data Stream and leave subsequent Data Streams without a Global Color Table or any Local Color Tables; in this way, the overhead for the table is eliminated. It is recommended that the decoder save the previous Global Color Table to be used with the Data Stream that follows, in case it does not contain either a Global Color Table or any Local Color Tables. In general, this allows the application program to use past color tables, significantly reducing transmission overhead.

Extension Blocks - Extensions are defined using the Extension Introducer code to mark the beginning of the block, followed by a block label, identifying the type of extension. Extension Codes are numbers in the range from 0x00 to 0xFF, inclusive. Special purpose extensions are transparent to the decoder and may be omitted when transmitting the Data Stream on-line. The GIF capabilities dialogue makes the provision for the receiver to request the transmission of all blocks; the default state in this regard is no transmission of Special purpose blocks.

Reserved Fields - All Reserved Fields are expected to have each bit set to zero (off).

Appendix

E. Interlaced Images.

The rows of an Interlaced images are arranged in the following order:

```
      Group 1 : Every 8th. row, starting with row 0.             (Pass 1)
      Group 2 : Every 8th. row, starting with row 4.             (Pass 2)
      Group 3 : Every 4th. row, starting with row 2.             (Pass 3)
      Group 4 : Every 2nd. row, starting with row 1.             (Pass 4)
```

The Following example illustrates how the rows of an interlaced image are ordered.

```
      Row Number                                        Interlace Pass

 0    -----------------------------------------       1
 1    -----------------------------------------                         4
 2    -----------------------------------------                   3
 3    -----------------------------------------                         4
 4    -----------------------------------------             2
 5    -----------------------------------------                         4
 6    -----------------------------------------                   3
 7    -----------------------------------------                         4
 8    -----------------------------------------       1
 9    -----------------------------------------                         4
 10   -----------------------------------------                   3
 11   -----------------------------------------                         4
 12   -----------------------------------------             2
 13   -----------------------------------------                         4
 14   -----------------------------------------                   3
 15   -----------------------------------------                         4
 16   -----------------------------------------       1
 17   -----------------------------------------                         4
 18   -----------------------------------------                   3
 19   -----------------------------------------                         4
```

Appendix
F. Variable-Length-Code LZW Compression.

The Variable-Length-Code LZW Compression is a variation of the Lempel-Ziv Compression algorithm in which variable-length codes are used to replace patterns detected in the original data. The algorithm uses a code or translation table constructed from the patterns encountered in the original data; each new pattern is entered into the table and its index is used to replace it in the compressed stream.

The compressor takes the data from the input stream and builds a code or translation table with the patterns as it encounters them; each new pattern is entered into the code table and its index is added to the output stream; when a pattern is encountered which had been detected since the last code table refresh, its index from the code table is put on the output stream, thus achieving the data compression. The expander takes input from the compressed data stream and builds the code or translation table from it; as the compressed data stream is processed, codes are used to index into the code table and the corresponding data is put on the decompressed output stream, thus achieving data decompression. The details of the algorithm are explained below. The Variable-Length-Code aspect of the algorithm is based on an initial code size (LZW-initial code size), which specifies the initial number of bits used for the compression codes. When the number of patterns detected by the compressor in the input stream exceeds the number of patterns encodable with the current number of bits, the number of bits per LZW code is increased by one.

The Raster Data stream that represents the actual output image can be represented as:

```
         7 6 5 4 3 2 1 0
        +---------------+
        | LZW code size |
        +---------------+

        +---------------+ ----+
        |  block size   |     |
        +---------------+     |
        |               |     +-- Repeated as many
        |  data bytes   |     |   times as necessary.
        |               |     |
        +---------------+ ----+

        . . .       . . . ------- The code that terminates the LZW
                                  compressed data must appear before
                                  Block Terminator.
        +---------------+
        |0 0 0 0 0 0 0 0|  Block Terminator
        +---------------+
```

The conversion of the image from a series of pixel values to a transmitted or stored character stream involves several steps. In brief these steps are:

1. Establish the Code Size - Define the number of bits needed to represent the actual data.

2. Compress the Data - Compress the series of image pixels to a series of

compression codes.

3. Build a Series of Bytes - Take the set of compression codes and convert to a string of 8-bit bytes.

4. Package the Bytes - Package sets of bytes into blocks preceded by character counts and output.

ESTABLISH CODE SIZE

The first byte of the Compressed Data stream is a value indicating the minimum number of bits required to represent the set of actual pixel values. Normally this will be the same as the number of color bits. Because of some algorithmic constraints however, black & white images which have one color bit must be indicated as having a code size of 2.

This code size value also implies that the compression codes must start out one bit longer.

COMPRESSION

The LZW algorithm converts a series of data values into a series of codes which may be raw values or a code designating a series of values. Using text characters as an analogy, the output code consists of a character or a code representing a string of characters.

The LZW algorithm used in GIF matches algorithmically with the standard LZW algorithm with the following differences:

1. A special Clear code is defined which resets all compression/decompression parameters and tables to a start-up state. The value of this code is 2**<code size>. For example if the code size indicated was 4 (image was 4 bits/pixel) the Clear code value would be 16 (10000 binary). The Clear code can appear at any point in the image data stream and therefore requires the LZW algorithm to process succeeding codes as if a new data stream was starting. Encoders should output a Clear code as the first code of each image data stream.

2. An End of Information code is defined that explicitly indicates the end of the image data stream. LZW processing terminates when this code is encountered. It must be the last code output by the encoder for an image. The value of this code is <Clear code>+1.

3. The first available compression code value is <Clear code>+2.

4. The output codes are of variable length, starting at <code size>+1 bits per code, up to 12 bits per code. This defines a maximum code value of 4095 (0xFFF). Whenever the LZW code value would exceed the current code length, the code length is increased by one. The packing/unpacking of these codes must then be altered to reflect the new code length.

BUILD 8-BIT BYTES

Because the LZW compression used for GIF creates a series of variable length codes, of between 3 and 12 bits each, these codes must be reformed into a series of 8-bit bytes that will be the characters actually stored or transmitted. This provides additional compression of the image. The codes are formed into a stream of bits as if they were packed right to left and then

picked off 8 bits at a time to be output.

Assuming a character array of 8 bits per character and using 5 bit codes to be packed, an example layout would be similar to:

```
     +---------------+
  0  |               |    bbbaaaaa
     +---------------+
  1  |               |    dcccccbb
     +---------------+
  2  |               |    eeeedddd
     +---------------+
  3  |               |    ggfffffe
     +---------------+
  4  |               |    hhhhhggg
     +---------------+
           . . .
     +---------------+
  N  |               |
     +---------------+
```

Note that the physical packing arrangement will change as the number of bits per compression code change but the concept remains the same.

PACKAGE THE BYTES

Once the bytes have been created, they are grouped into blocks for output by preceding each block of 0 to 255 bytes with a character count byte. A block with a zero byte count terminates the Raster Data stream for a given image. These blocks are what are actually output for the GIF image. This block format has the side effect of allowing a decoding program the ability to read past the actual image data if necessary by reading block counts and then skipping over the data.

FURTHER READING

[1] Ziv, J. and Lempel, A. : "A Universal Algorithm for Sequential Data Compression", IEEE Transactions on Information Theory, May 1977.
[2] Welch, T. : "A Technique for High-Performance Data Compression", Computer, June 1984.
[3] Nelson, M.R. : "LZW Data Compression", Dr. Dobb's Journal, October 1989.

Appendix
G. On-line Capabilities Dialogue.

NOTE : This section is currently (10 July 1990) under revision; the information provided here should be used as general guidelines. Code written based on this information should be designed in a flexible way to accommodate any changes resulting from the revisions.

The following sequences are defined for use in mediating control between a GIF sender and GIF receiver over an interactive communications line. These sequences do not apply to applications that involve downloading of static GIF files and are not considered part of a GIF file.

GIF CAPABILITIES ENQUIRY

The GIF Capabilities Enquiry sequence is issued from a host and requests an interactive GIF decoder to return a response message that defines the graphics parameters for the decoder. This involves returning information about available screen sizes, number of bits/color supported and the amount of color detail supported. The escape sequence for the GIF Capabilities Enquiry is defined as:

```
ESC[>0g           0x1B 0x5B 0x3E 0x30 0x67
```

GIF CAPABILITIES RESPONSE

The GIF Capabilities Response message is returned by an interactive GIF decoder and defines the decoder's display capabilities for all graphics modes that are supported by the software. Note that this can also include graphics printers as well as a monitor screen. The general format of this message is:

```
#version;protocol{;dev, width, height, color-bits, color-res}...<CR>
```

```
'#'            GIF Capabilities Response identifier character.
version        GIF format version number;  initially '87a'.
protocol='0'   No end-to-end protocol supported by decoder Transfer as direct
               8-bit data stream.
protocol='1'   Can use CIS B+ error correction protocol to transfer GIF data
               interactively from the host directly to the display.
dev = '0'      Screen parameter set follows.
dev = '1'      Printer parameter set follows.
width          Maximum supported display width in pixels.
height         Maximum supported display height in pixels.
color-bits     Number of bits per pixel supported. The number of supported
               colors is therefore 2**color-bits.
color-res      Number of bits per color component supported in the hardware
               color palette. If color-res is '0' then no hardware palette
               table is available.
```

Note that all values in the GIF Capabilities Response are returned as ASCII decimal numbers and the message is terminated by a Carriage Return character.

The following GIF Capabilities Response message describes three standard IBM PC Enhanced Graphics Adapter configurations with no printer; the GIF data stream

can be processed within an error correcting protocol:

```
#87a;1;0,320,200,4,0;0,640,200,2,2;0,640,350,4,2<CR>
```

ENTER GIF GRAPHICS MODE

Two sequences are currently defined to invoke an interactive GIF decoder into action. The only difference between them is that different output media are selected. These sequences are:

ESC[>1g   Display GIF image on screen

```
                  0x1B 0x5B 0x3E 0x31 0x67
```

ESC[>2g   Display image directly to an attached graphics printer. The image may optionally be displayed on the screen as well.

```
                  0x1B 0x5B 0x3E 0x32 0x67
```

Note that the 'g' character terminating each sequence is in lowercase.

INTERACTIVE ENVIRONMENT

The assumed environment for the transmission of GIF image data from an interactive application is a full 8-bit data stream from host to micro. All 256 character codes must be transferrable. The establishing of an 8-bit data path for communications will normally be taken care of by the host application programs. It is however up to the receiving communications programs supporting GIF to be able to receive and pass on all 256 8-bit codes to the GIF decoder software.

GZIP file format specification version 4.3

Status of This Memo

This memo provides information for the Internet community. This memo does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

IESG Note:

The IESG takes no position on the validity of any Intellectual Property Rights statements contained in this document.

Notices

A pointer to the latest version of this and related documentation in HTML format can be found at the URL <ftp://ftp.uu.net/graphics/png/documents/zlib/zdoc-index.html>.

Abstract

This specification defines a lossless compressed data format that is compatible with the widely used GZIP utility. The format includes a cyclic redundancy check value for detecting data corruption. The format presently uses the DEFLATE method of compression but can be easily extended to use other compression methods. The format can be implemented readily in a manner not covered by patents.

Table of Contents

* 1. Introduction
    o 1.1. Purpose
    o 1.2. Intended audience
    o 1.3. Scope
    o 1.4. Compliance
    o 1.5. Definitions of terms and conventions used
    o 1.6. Changes from previous versions
* 2. Detailed specification
    o 2.1. Overall conventions
    o 2.2. File format
    o 2.3. Member format
        + 2.3.1. Member header and trailer
            + 2.3.1.1. Extra field
            + 2.3.1.2. Compliance
    o 3. References
    o 4. Security Considerations
    o 5. Acknowledgements
    o 6. Author's Address
    o 7. Appendix: Jean-Loup Gailly's gzip utility
    o 8. Appendix: Sample CRC Code

1. Introduction

Purpose

The purpose of this specification is to define a lossless compressed data format that:

* Is independent of CPU type, operating system, file system, and character set, and hence can be used for interchange;
* Can compress or decompress a data stream (as opposed to a randomly accessible file) to produce another data stream, using only an a priori bounded amount of intermediate storage, and hence can be used in data communications or similar structures such as Unix filters;
* Compresses data with efficiency comparable to the best currently

available general-purpose compression methods, and in particular considerably better than the "compress" program;

* Can be implemented readily in a manner not covered by patents, and hence can be practiced freely;
* Is compatible with the file format produced by the current widely used gzip utility, in that conforming decompressors will be able to read data produced by the existing gzip compressor.

The data format defined by this specification does not attempt to:

* Provide random access to compressed data;
* Compress specialized data (e.g., raster graphics) as well as the best currently available specialized algorithms.

Intended audience

This specification is intended for use by implementors of software to compress data into gzip format and/or decompress data from gzip format.

The text of the specification assumes a basic background in programming at the level of bits and other primitive data representations.

Scope

The specification specifies a compression method and a file format (the latter assuming only that a file can store a sequence of arbitrary bytes). It does not specify any particular interface to a file system or anything about character sets or encodings (except for file names and comments, which are optional).

Compliance

Unless otherwise indicated below, a compliant decompressor must be able to accept and decompress any file that conforms to all the specifications presented here; a compliant compressor must produce files that conform to all the specifications presented here. The material in the appendices is not part of the specification per se and is not relevant to compliance.

Definitions of terms and conventions used

byte: 8 bits stored or transmitted as a unit (same as an octet). (For this specification, a byte is exactly 8 bits, even on machines which store a character on a number of bits different from 8.) See below for the numbering of bits within a byte.

1.6. Changes from previous versions

There have been no technical changes to the gzip format since version 4.1 of this specification. In version 4.2, some terminology was changed, and the sample CRC code was rewritten for clarity and to eliminate the requirement for the caller to do pre- and post-conditioning. Version 4.3 is a conversion of the specification to RFC style.

2. Detailed specification

Overall conventions

In the diagrams below, a box like this:

```
+---+
|   | <-- the vertical bars might be missing
+---+
```

represents one byte; a box like this:

```
+==============+
|              |
+==============+
```

represents a variable number of bytes.

Bytes stored within a computer do not have a "bit order", since they are always treated as a unit. However, a byte considered as an integer between

0 and 255 does have a most- and least-significant bit, and since we write numbers with the most-significant digit on the left, we also write bytes with the most-significant bit on the left. In the diagrams below, we number the bits of a byte so that bit 0 is the least-significant bit, i.e., the bits are numbered:

```
+--------+
|76543210|
+--------+
```

This document does not address the issue of the order in which bits of a byte are transmitted on a bit-sequential medium, since the data format described here is byte- rather than bit-oriented.

Within a computer, a number may occupy multiple bytes. All multi-byte numbers in the format described here are stored with the least-significant byte first (at the lower memory address). For example, the decimal number 520 is stored as:

```
    0        1
+--------+--------+
|00001000|00000010|
+--------+--------+
 ^        ^
 |        |
 |        + more significant byte = 2 x 256
 + less significant byte = 8
```

File format

A gzip file consists of a series of "members" (compressed data sets). The format of each member is specified in the following section. The members simply appear one after another in the file, with no additional information before, between, or after them.

Member format

Each member has the following structure:

```
+---+---+---+---+---+---+---+---+---+---+
|ID1|ID2|CM |FLG|     MTIME     |XFL|OS | (more-->)
+---+---+---+---+---+---+---+---+---+---+
```

(if FLG.FEXTRA set)

```
+---+---+=================================+
| XLEN  |...XLEN bytes of "extra field"...| (more-->)
+---+---+=================================+
```

(if FLG.FNAME set)

```
+=========================================+
|...original file name, zero-terminated...| (more-->)
+=========================================+
```

(if FLG.FCOMMENT set)

```
+===================================+
|...file comment, zero-terminated...| (more-->)
+===================================+
```

(if FLG.FHCRC set)

```
+---+---+
| CRC16 |
+---+---+
```

```
+=======================+
|...compressed blocks...| (more-->)
+=======================+
```

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
|     CRC32     |     ISIZE     |
+---+---+---+---+---+---+---+---+
```

Member header and trailer

ID1 (IDentification 1)
ID2 (IDentification 2)

These have the fixed values ID1 = 31 (0x1f, \037), ID2 = 139 (0x8b, \213), to identify the file as being in gzip format.

CM (Compression Method)

This identifies the compression method used in the file. CM = 0-7 are reserved. CM = 8 denotes the "deflate" compression method, which is the one customarily used by gzip and which is documented elsewhere.

FLG (FLaGs)

This flag byte is divided into individual bits as follows:

bit 0	FTEXT
bit 1	FHCRC
bit 2	FEXTRA
bit 3	FNAME
bit 4	FCOMMENT
bit 5	reserved
bit 6	reserved
bit 7	reserved

If FTEXT is set, the file is probably ASCII text. This is an optional indication, which the compressor may set by checking a small amount of the input data to see whether any non-ASCII characters are present. In case of doubt, FTEXT is cleared, indicating binary data. For systems which have different file formats for ascii text and binary data, the decompressor can use FTEXT to choose the appropriate format. We deliberately do not specify the algorithm used to set this bit, since a compressor always has the option of leaving it cleared and a decompressor always has the option of ignoring it and letting some other program handle issues of data conversion.

If FHCRC is set, a CRC16 for the gzip header is present, immediately before the compressed data. The CRC16 consists of the two least significant bytes of the CRC32 for all bytes of the gzip header up to and not including the CRC16. [The FHCRC bit was never set by versions of gzip up to 1.2.4, even though it was documented with a different meaning in gzip 1.2.4.]

If FEXTRA is set, optional extra fields are present, as described in a following section.

If FNAME is set, an original file name is present, terminated by a zero byte. The name must consist of ISO 8859-1 (LATIN-1) characters; on operating systems using EBCDIC or any other character set for file names, the name must be translated to the ISO LATIN-1 character set. This is the original name of the file being compressed, with any directory components removed, and, if the file being compressed is on a file system with case insensitive names, forced to lower case. There is no original file name if the data was compressed from a source other than a named file; for example, if the source was stdin on a Unix system, there is no file name.

If FCOMMENT is set, a zero-terminated file comment is present. This comment is not interpreted; it is only intended for human consumption. The comment must consist of ISO 8859-1 (LATIN-1) characters. Line breaks should be denoted by a single line feed character (10 decimal).

Reserved FLG bits must be zero.

MTIME (Modification TIME)

This gives the most recent modification time of the original file being compressed. The time is in Unix format, i.e., seconds since 00:00:00 GMT, Jan. 1, 1970. (Note that this may cause problems for MS-DOS and other systems that use local rather than Universal time.) If the compressed data did not come from a file, MTIME is set to the time at which compression started. MTIME = 0 means no time stamp is available.

XFL (eXtra FLags)

These flags are available for use by specific compression methods. The "deflate" method (CM = 8) sets these flags as follows:

```
XFL = 2 - compressor used maximum compression,
          slowest algorithm
XFL = 4 - compressor used fastest algorithm
```

OS (Operating System)

This identifies the type of file system on which compression took place. This may be useful in determining end-of-line convention for text files. The currently defined values are as follows:

```
  0 - FAT filesystem (MS-DOS, OS/2, NT/Win32)
  1 - Amiga
  2 - VMS (or OpenVMS)
  3 - Unix
  4 - VM/CMS
  5 - Atari TOS
  6 - HPFS filesystem (OS/2, NT)
  7 - Macintosh
  8 - Z-System
  9 - CP/M
 10 - TOPS-20
 11 - NTFS filesystem (NT)
 12 - QDOS
 13 - Acorn RISCOS
255 - unknown
```

XLEN (eXtra LENgth)

If FLG.FEXTRA is set, this gives the length of the optional extra field. See below for details.

CRC32 (CRC-32)

This contains a Cyclic Redundancy Check value of the uncompressed data computed according to CRC-32 algorithm used in the ISO 3309 standard and in section 8.1.1.6.2 of ITU-T recommendation V.42. (See http://www.iso.ch for ordering ISO documents. See gopher://info.itu.ch for an online version of ITU-T V.42.)

ISIZE (Input SIZE)

This contains the size of the original (uncompressed) input data modulo 2^32.

Extra field

If the FLG.FEXTRA bit is set, an "extra field" is present in the header, with total length XLEN bytes. It consists of a series of subfields, each of the form:

```
+---+---+---+---+==================================+
|SI1|SI2|  LEN  |... LEN bytes of subfield data ...|
+---+---+---+---+==================================+
```

SI1 and SI2 provide a subfield ID, typically two ASCII letters with some mnemonic value. Jean-Loup Gailly <gzip@prep.ai.mit.edu> is maintaining a registry of subfield IDs; please send him any subfield ID you wish to use. Subfield IDs with SI2 = 0 are reserved for future use. The following IDs are currently defined:

SI1	SI2	Data
0x41 ('A')	0x70 ('P')	Apollo file type information

LEN gives the length of the subfield data, excluding the 4 initial bytes.

Compliance

A compliant compressor must produce files with correct ID1, ID2, CM, CRC32, and ISIZE, but may set all the other fields in the fixed-length part of the header to default values (255 for OS, 0 for all others). The compressor must set all reserved bits to zero.

A compliant decompressor must check ID1, ID2, and CM, and provide an error indication if any of these have incorrect values. It must examine

FEXTRA/XLEN, FNAME, FCOMMENT and FHCRC at least so it can skip over the optional fields if they are present. It need not examine any other part of the header or trailer; in particular, a decompressor may ignore FTEXT and OS and always produce binary output, and still be compliant. A compliant decompressor must give an error indication if any reserved bit is non-zero, since such a bit could indicate the presence of a new field that would cause subsequent data to be interpreted incorrectly.

3. References

[1] "Information Processing - 8-bit single-byte coded graphic character sets - Part 1: Latin alphabet No.1" (ISO 8859-1:1987). The ISO 8859-1 (Latin-1) character set is a superset of 7-bit ASCII. Files defining this character set are available as iso_8859-1.* in ftp://ftp.uu.net/graphics/png/documents/

[2] ISO 3309

[3] ITU-T recommendation V.42

[4] Deutsch, L.P.,"DEFLATE Compressed Data Format Specification", available in ftp://ftp.uu.net/pub/archiving/zip/doc/

[5] Gailly, J.-L., GZIP documentation, available as gzip-*.tar in ftp://prep.ai.mit.edu/pub/gnu/

[6] Sarwate, D.V., "Computation of Cyclic Redundancy Checks via Table Look-Up", Communications of the ACM, 31(8), pp.1008-1013.

[7] Schwaderer, W.D., "CRC Calculation", April 85 PC Tech Journal, pp.118-133.

[8] ftp://ftp.adelaide.edu.au/pub/rocksoft/papers/crc_v3.txt, describing the CRC concept.

4. Security Considerations

Any data compression method involves the reduction of redundancy in the data. Consequently, any corruption of the data is likely to have severe effects and be difficult to correct. Uncompressed text, on the other hand, will probably still be readable despite the presence of some corrupted bytes. It is recommended that systems using this data format provide some means of validating the integrity of the compressed data, such as by setting and checking the CRC-32 check value.

5. Acknowledgements

Trademarks cited in this document are the property of their respective owners.

Jean-Loup Gailly designed the gzip format and wrote, with Mark Adler, the related software described in this specification. Glenn Randers-Pehrson converted this document to RFC and HTML format.

6. Author's Address

L. Peter Deutsch

Aladdin Enterprises
203 Santa Margarita Ave.
Menlo Park, CA 94025

Phone: (415) 322-0103 (AM only)
FAX: (415) 322-1734
EMail: <ghost@aladdin.com>

Questions about the technical content of this specification can be sent by email to:

Jean-Loup Gailly <gzip@prep.ai.mit.edu> and
Mark Adler <madler@alumni.caltech.edu>

Editorial comments on this specification can be sent by email to:

L. Peter Deutsch <ghost@aladdin.com> and
Glenn Randers-Pehrson <randeg@alumni.rpi.edu>

7. Appendix: Jean-Loup Gailly's gzip utility

The most widely used implementation of gzip compression, and the original documentation on which this specification is based, were created by Jean-Loup Gailly <gzip@prep.ai.mit.edu>. Since this implementation is a de facto standard, we mention some more of its features here. Again, the material in this section is not part of the specification per se, and implementations need not follow it to be compliant.

When compressing or decompressing a file, gzip preserves the protection, ownership, and modification time attributes on the local file system, since there is no provision for representing protection attributes in the gzip file format itself. Since the file format includes a modification time, the gzip decompressor provides a command line switch that assigns the modification time from the file, rather than the local modification time of the compressed input, to the decompressed output.

8. Appendix: Sample CRC Code

The following sample code represents a practical implementation of the CRC (Cyclic Redundancy Check). (See also ISO 3309 and ITU-T V.42 for a formal specification.)

The sample code is in the ANSI C programming language. Non C users may find it easier to read with these hints:

&	Bitwise AND operator.
^	Bitwise exclusive-OR operator.
>>	Bitwise right shift operator. When applied to an unsigned quantity, as here, right shift inserts zero bit(s) at the left.
!	Logical NOT operator.
++	"n++" increments the variable n.
0xNNN	0x introduces a hexadecimal (base 16) constant. Suffix L indicates a long value (at least 32 bits).

```
/* Table of CRCs of all 8-bit messages. */
unsigned long crc_table[256];

/* Flag: has the table been computed? Initially false. */
int crc_table_computed = 0;

/* Make the table for a fast CRC. */
void make_crc_table(void)
{
  unsigned long c;
  int n, k;

  for (n = 0; n < 256; n++) {
    c = (unsigned long) n;
    for (k = 0; k < 8; k++) {
      if (c & 1) {
        c = 0xedb88320L ^ (c >> 1);
      } else {
        c = c >> 1;
      }
    }
    crc_table[n] = c;
  }
  crc_table_computed = 1;
}

/*
   Update a running crc with the bytes buf[0..len-1] and return
 the updated crc. The crc should be initialized to zero. Pre- and
 post-conditioning (one's complement) is performed within this
 function so it shouldn't be done by the caller. Usage example:

   unsigned long crc = 0L;
   while (read_buffer(buffer, length) != EOF) {
     crc = update_crc(crc, buffer, length);
   }
   if (crc != original_crc) error();
*/
unsigned long update_crc(unsigned long crc,
                unsigned char *buf, int len)
{
  unsigned long c = crc ^ 0xffffffffL;
  int n;

  if (!crc_table_computed)
    make_crc_table();
  for (n = 0; n < len; n++) {
    c = crc_table[(c ^ buf[n]) & 0xff] ^ (c >> 8);
  }
  return c ^ 0xffffffffL;
}

/* Return the CRC of the bytes buf[0..len-1]. */
unsigned long crc(unsigned char *buf, int len)
{
  return update_crc(0L, buf, len);
}
```

OS/2 HELP Format
Intel byte order

Information from File Format List 2.0 by Max Maischein.

--------!-CONTACT_INFO----------------------
If you notice any mistakes or omissions, please let me know! It is only with YOUR help that the list can continue to grow. Please send all changes to me rather than distributing a modified version of the list.

This file has been authored in the style of the INTERxxy.* file list by Ralf Brown, and uses almost the same format.

Please read the file FILEFMTS.1ST before asking me any questions. You may find that they have already been addressed.

Max Maischein

Max Maischein, 2:244/1106.17
Max_Maischein@spam.fido.de
corion@informatik.uni-frankfurt.de
Corion on #coders@IRC

--------!-DISCLAIMER------------------------
DISCLAIMER: THIS MATERIAL IS PROVIDED "AS IS". I verify the information contained in this list to the best of my ability, but I cannot be held responsible for any problems caused by use or misuse of the information, especially for those file formats foreign to the PC, like AMIGA or SUN file formats. If an information it is marked "guesswork" or undocumented, you should check it carefully to make sure your program will not break with an unexpected value (and please let me know whether or not it works the same way).

Information marked with "???" is known to be incomplete or guesswork.

Some file formats were not released by their creators, others are regarded as proprietary, which means that if your programs deal with them, you might be looking for trouble. I don't care about this.

The OS/2 help files are different from the WinHelp help files,since the WinHelp format is proprietary to MicroSoft because of the patented LZ-packing they implemented.

```
OFFSET              Count TYPE   Description
0000h                   3 char   ID='HSP'
0003h                   1 byte   Flags :
                                   0 - INF style file
                                 1-3 - unknown
                                   4 - HLP style file
                                 Patching this file allows reading HLP files
                                 using the VIEW command, while HLP files seem to
                                 work with INF settings as well.
0005h                   1 word   Total size of header
0007h                   1 word   Unknown
????h                   other data
0047h                   ? char   ASCIIZ name of the HLP/INF file
```

EXTENSION:HLP,INF
OCCURENCES:OS/2
REFERENCE:INF02A.DOC
SEE ALSO:WinHelp HLP

Macintosh 7 & 8 bit File Transfer Format - Protocol Independent

Here is a description of the Hqx7 (7 bit format as implemented in BinHex 4.0) and Hqx8 (8 bit format) formats for Macintosh Application and File transfers. The main features of the formats are:

```
1) Error checking even using ASCII download (Hqx7 & Hqx8)
2) Compression of repetitive characters     (Hqx7 & Hqx8)
3) 7 bit encoding for ASCII download        (Hqx7)
```

HQX Format Description (This is not intended to be a programmer's reference).

The format is processed at three different levels:

1) 8 bit encoding of the file:

Byte:	Length of FileName (1->63)
Bytes:	FileName ("Length" bytes)
Byte:	Version
Long:	Type
Long:	Creator
Word:	Flags (And $F800)
Long:	Length of Data Fork
Long:	Length of Resource Fork
Word:	CRC
Bytes:	Data Fork ("Data Length" bytes)
Word:	CRC
Bytes:	Resource Fork ("Rsrc Length" bytes)
Word:	CRC

2) Compression of repetitive characters.

($90 is the marker, encoding is made for 3->255 characters)

```
00 11 22 33 44 55 66 77      ->      00 11 22 33 44 55 66 77
11 22 22 22 22 22 22 33      ->      11 22 90 06 33
11 22 90 33 44               ->      11 22 90 00 33 44
```

3) 7 bit encoding (Hqx7 only).

The whole file is considered as a stream of bits. This stream will be divided in blocks of 6 bits and then converted to one of 64 characters contained in a table. The characters in this table have been chosen for maximum noise protection.
The format will start with a ":" (first character on a line) and end with a ":". There will be a maximum of 64 characters on a line.
It can be preceeded by comments for novice users.

Table:
```
!"#$%&'()*+,-012345689@ABCDEFGHIJKLMNPQRSTUVXYZ[`abcdefhijklmpqr
```

Comment:
(This file must be converted with BinHex 4.0)

Yves Lempereur [YVES]

G I F (tm)

Graphics Interchange Format (tm)

A standard defining a mechanism
for the storage and transmission
of raster-based graphics information

June 15, 1987

an H&R Block Company
5000 Arlington Centre Blvd.
Columbus, Ohio 43220
(614) 457-8600

Graphics Interchange Format (GIF) Specification

Table of Contents

INTRODUCTION . . . . page 3
GENERAL FILE FORMAT . . . . page 3
GIF SIGNATURE . . . . page 4
SCREEN DESCRIPTOR . . . . page 4
GLOBAL COLOR MAP . . . . page 5
IMAGE DESCRIPTOR . . . . page 6
LOCAL COLOR MAP . . . . page 7
RASTER DATA . . . . page 7
GIF TERMINATOR . . . . page 8
GIF EXTENSION BLOCKS . . . . page 8
APPENDIX A - GLOSSARY . . . . page 9
APPENDIX B - INTERACTIVE SEQUENCES . . . . . . page 10

APPENDIX C - IMAGE PACKAGING & COMPRESSION . . page 12

APPENDIX D - MULTIPLE IMAGE PROCESSING . . . . page 15

Graphics Interchange Format (GIF) Page 3

Specification

INTRODUCTION

'GIF' (tm) is CompuServe's standard for defining generalized color raster images. This 'Graphics Interchange Format' (tm) allows high-quality, high-resolution graphics to be displayed on a variety of graphics hardware and is intended as an exchange and display mechanism for graphics images. The image format described in this document is designed to support current and future image technology and will in addition serve as a basis for future CompuServe graphics products.

The main focus of this document is to provide the technical information necessary for a programmer to implement GIF encoders and decoders. As such, some assumptions are made as to terminology relavent to graphics and programming in general.

The first section of this document describes the GIF data format and its components and applies to all GIF decoders, either as standalone programs or as part of a communications package. Appendix B is a section relavent to decoders that are part of a communications software package and describes the protocol requirements for entering and exiting GIF mode, and responding to host interrogations. A glossary in Appendix A defines some of the terminology used in this document. Appendix C gives a detailed explanation of how the graphics image itself is packaged as a series of data bytes.

Graphics Interchange Format Data Definition

GENERAL FILE FORMAT

```
+-----------------------+
| +-------------------+ |
| |   GIF Signature   | |
| +-------------------+ |
| +-------------------+ |
| | Screen Descriptor | |
| +-------------------+ |

| +-------------------+ |

| | Global Color Map  | |

| +-------------------+ |

. . .      . . .

| +-------------------+ |    ---+

| |  Image Descriptor | | |

| +-------------------+ | |

| +-------------------+ | |

| |  Local Color Map  | | |-   Repeated 1 to n times

| +-------------------+ | |

| +-------------------+ | |

| |    Raster Data    | | |

| +-------------------+ |    ---+

. . .      . . .

|-    GIF Terminator   -|

+-----------------------+
```

Graphics Interchange Format (GIF)   Page 4

Specification

GIF SIGNATURE

The following GIF Signature identifies the data following as a valid GIF image stream. It consists of the following six characters:

```
   G I F 8 7 a
```

The last three characters '87a' may be viewed as a version number for this particular GIF definition and will be used in general as a reference in documents regarding GIF that address any version dependencies.

SCREEN DESCRIPTOR

The Screen Descriptor describes the overall parameters for all GIF images following. It defines the overall dimensions of the image space or logical screen required, the existance of color mapping information, background screen color, and color depth information. This information is stored in a series of 8-bit bytes as described below.

```
       bits

  7 6 5 4 3 2 1 0  Byte #

 +---------------+

 |  |  1

 +-Screen Width -+      Raster width in pixels (LSB first)

 |  |  2

 +---------------+

 |  |  3

 +-Screen Height-+      Raster height in pixels (LSB first)

 |  |  4

 +-+-----+-+-----+      M = 1, Global color map follows Descriptor

 |M|  cr |0|pixel|  5   cr+1 = # bits of color resolution

 +-+-----+-+-----+      pixel+1 = # bits/pixel in image

 |   background |  6   background=Color index of screen background

 +---------------+    (color is defined from the Global color

 |0 0 0 0 0 0 0 0|  7     map or default map if none specified)

 +---------------+
```

The logical screen width and height can both be larger than the physical display. How images larger than the physical display are handled is implementation dependent and can take advantage of hardware characteristics (e.g. Macintosh scrolling windows). Otherwise images can be clipped to the edges of the display.

The value of 'pixel' also defines the maximum number of colors within an image. The range of values for 'pixel' is 0 to 7 which represents 1 to 8 bits. This translates to a range of 2 (B & W) to 256 colors. Bit 3 of word 5 is reserved for future definition and must be zero.

Graphics Interchange Format (GIF) Page 5

Specification

GLOBAL COLOR MAP

The Global Color Map is optional but recommended for images where accurate color rendition is desired. The existence of this color map is indicated in the 'M' field of byte 5 of the Screen Descriptor. A color map can also be associated with each image in a GIF file as described later. However this global map will normally be used because of hardware restrictions in equipment available today. In the individual Image Descriptors the 'M' flag will normally be zero. If the Global Color Map is present, it's definition immediately follows the Screen Descriptor. The number of color map entries following a Screen Descriptor is equal to 2**(# bits per pixel), where each entry consists of three byte values representing the relative intensities of red, green and blue respectively. The structure of the Color Map block is:

```
      bits
 7 6 5 4 3 2 1 0  Byte #
+---------------+
| red intensity |  1 Red value for color index 0
+---------------+
|green intensity|  2 Green value for color index 0
+---------------+
| blue intensity|  3 Blue value for color index 0
+---------------+
| red intensity |  4 Red value for color index 1
+---------------+
|green intensity|  5 Green value for color index 1
+---------------+
| blue intensity|  6 Blue value for color index 1
+---------------+
:  : (Continues for remaining colors)
```

Each image pixel value received will be displayed according to its closest match with an available color of the display based on this color map. The color components represent a fractional intensity value from none (0) to full (255). White would be represented as (255,255,255), black as (0,0,0) and medium yellow as (180,180,0). For display, if the device supports fewer than 8 bits per color component, the higher order bits of each component are used. In the creation of a GIF color map entry with hardware supporting fewer than 8 bits per component, the component values for the hardware should be converted to the 8-bit format with the following calculation:

<map_value> = <component_value>*255/(2**<nbits> -1)

This assures accurate translation of colors for all displays. In the cases of creating GIF images from hardware without color palette capability, a fixed palette should be created based on the available display colors for that hardware. If no Global Color Map is indicated, a default color map is generated internally which maps each possible incoming color index to the same hardware color index modulo <n> where <n> is the number of available hardware colors.

Graphics Interchange Format (GIF)   Page 6

Specification

IMAGE DESCRIPTOR

The Image Descriptor defines the actual placement and extents of the following image within the space defined in the Screen Descriptor. Also defined are flags to indicate the presence of a local color lookup map, and to define the pixel display sequence. Each Image Descriptor is introduced by an image separator character. The role of the Image Separator is simply to provide a synchronization character to introduce an Image Descriptor. This is desirable if a GIF file happens to contain more than one image. This character is defined as 0x2C hex or ',' (comma). When this character is encountered between images, the Image Descriptor will follow immediately.

Any characters encountered between the end of a previous image and the image separator character are to be ignored. This allows future GIF enhancements to be present in newer image formats and yet ignored safely by older software decoders.

```
      bits
 7 6 5 4 3 2 1 0  Byte #
+---------------+
|0 0 1 0 1 1 0 0|  1 ',' - Image separator character
+---------------+
|  |  2 Start of image in pixels from the
```

```
+-  Image Left -+ left side of the screen (LSB first)
|  |  3
+---------------+
|  |  4
+-  Image Top  -+ Start of image in pixels from the
|  |  5 top of the screen (LSB first)
+---------------+
|  |  6
+- Image Width -+ Width of the image in pixels (LSB first)
|  |  7
+---------------+
|  |  8
+- Image Height-+ Height of the image in pixels (LSB first)
|  |  9
+-+-+-+-+-+-----+ M=0 - Use global color map, ignore 'pixel'
|M|I|0|0|0|pixel| 10 M=1 - Local color map follows, use 'pixel'
+-+-+-+-+-+-----+ I=0 - Image formatted in Sequential order
   I=1 - Image formatted in Interlaced order
   pixel+1 - # bits per pixel for this image
```

The specifications for the image position and size must be confined to the dimensions defined by the Screen Descriptor. On the other hand it is not necessary that the image fill the entire screen defined.

LOCAL COLOR MAP

Graphics Interchange Format (GIF) Page 7

Specification

A Local Color Map is optional and defined here for future use. If the 'M' bit of byte 10 of the Image Descriptor is set, then a color map follows the Image Descriptor that applies only to the following image. At the end of the image, the color map will revert to that defined after the Screen Descriptor. Note that the 'pixel' field of byte 10 of the Image Descriptor is used only if a Local Color Map is indicated. This defines the parameters not only for the image pixel size, but determines the number of color map entries that follow. The bits per pixel value will also revert to the value specified in the Screen Descriptor when processing of the image is complete.

RASTER DATA

The format of the actual image is defined as the series of pixel color index values that make up the image. The pixels are stored left to right sequentially for an image row. By default each image row is written sequentially, top to bottom. In the case that the Interlace or 'I' bit is set in byte 10 of the Image Descriptor then the row order of the image display follows a four-pass process in which the image is filled in by widely spaced rows. The first pass writes every 8th row, starting with the top row of the image window. The second pass writes every 8th row starting at the fifth row from the top. The third pass writes every 4th row starting at the third row from the top. The fourth pass completes the image, writing every other row, starting at the second row from the top. A graphic description of this process follows:

```
Image
Row Pass 1 Pass 2 Pass 3 Pass 4  Result
---------------------------------------------------
  0 **1a**     **1a**
  1    **4a**  **4a**
  2   **3a**   **3a**
  3    **4b**  **4b**
  4  **2a**    **2a**
  5    **4c**  **4c**
  6   **3b**   **3b**
  7    **4d**  **4d**
  8 **1b**     **1b**
  9    **4e**  **4e**
 10   **3c**   **3c**
 11    **4f**  **4f**
 12  **2b**    **2b**
. . .
```

The image pixel values are processed as a series of color indices which map into the existing color map. The resulting color value from the map is what is actually displayed. This series of pixel indices, the number of which is equal to image-width*image-height pixels, are passed to the GIF image data stream one value per pixel, compressed and packaged according to a version of the LZW compression algorithm as defined in Appendix C.

Graphics Interchange Format (GIF)   Page 8

Specification

GIF TERMINATOR

In order to provide a synchronization for the termination of a GIF image file, a GIF decoder will process the end of GIF mode when the character 0x3B hex or ';' is found after an image has been processed. By convention the decoding software will pause and wait for an action indicating that the user is ready to continue. This may be a carriage return entered at the keyboard or a mouse click. For interactive applications this user action must be passed on to the host as a carriage return character so that the host application can continue. The decoding software will then typically leave graphics mode and resume any previous process.

GIF EXTENSION BLOCKS

To provide for orderly extension of the GIF definition, a mechanism for defining the packaging of extensions within a GIF data stream is necessary. Specific GIF extensions are to be defined and documented by CompuServe in order to provide a controlled enhancement path.

GIF Extension Blocks are packaged in a manner similar to that used by the raster data though not compressed. The basic structure is:

```
 7 6 5 4 3 2 1 0  Byte #
+---------------+
|0 0 1 0 0 0 0 1|  1    '!' - GIF Extension Block Introducer
+---------------+
| function code |  2    Extension function code (0 to 255)
+---------------+    ---+
|  byte count | |
+---------------+ |
:  : +-- Repeated as many times as necessary
|func data bytes| |
:  : |
+---------------+    ---+
. . .     . . .
+---------------+
|0 0 0 0 0 0 0 0| zero byte count (terminates block)
+---------------+
```

A GIF Extension Block may immediately preceed any Image Descriptor or occur before the GIF Terminator.

All GIF decoders must be able to recognize the existence of GIF Extension Blocks and read past them if unable to process the function code. This ensures that older decoders will be able to process extended GIF image files in the future, though without the additional functionality.

Appendix A - Glossary

GLOSSARY

Pixel - The smallest picture element of a graphics image. This usually corresponds to a single dot on a graphics screen. Image resolution is typically given in units of pixels. For example a fairly standard graphics screen format is one 320 pixels across and 200 pixels high. Each pixel can appear as one of several colors depending on the capabilities of the graphics hardware.

Raster - A horizontal row of pixels representing one line of an image. A typical method of working with images since most hardware is oriented to work most efficiently in this manner.

LSB - Least Significant Byte. Refers to a convention for two byte numeric values in which the less significant byte of the value preceeds the more significant byte. This convention is typical on many microcomputers.

Color Map - The list of definitions of each color used in a GIF image. These desired colors are converted to available colors through a table which is derived by assigning an incoming color index (from the image) to an output color index (of the hardware). While the color map definitons are specified in a GIF image, the output pixel colors will vary based on the hardware used and its ability to match the defined color.

Interlace - The method of displaying a GIF image in which multiple passes are made, outputting raster lines spaced apart to provide a way of visualizing the general content of an entire image before all of the data has been processed.

B Protocol - A CompuServe-developed error-correcting file transfer protocol available in the public domain and implemented in CompuServe VIDTEX products. This error checking mechanism will be used in transfers of GIF images for interactive applications.

LZW - A sophisticated data compression algorithm based on work done by Lempel-Ziv & Welch which has the feature of very efficient one-pass encoding and decoding. This allows the image to be decompressed and displayed at the same time. The original article from which this technique was adapted is:

Terry A. Welch, "A Technique for High Performance Data Compression", IEEE Computer, vol 17 no 6 (June 1984)

This basic algorithm is also used in the public domain ARC file compression utilities. The CompuServe adaptation of LZW for GIF is described in Appendix C.

Appendix B - Interactive Sequences

GIF Sequence Exchanges for an Interactive Environment

The following sequences are defined for use in mediating control between a GIF sender and GIF receiver over an interactive communications line. These sequences do not apply to applications that involve downloading of static GIF files and are not considered part of a GIF file.

GIF CAPABILITIES ENQUIRY

The GCE sequence is issued from a host and requests an interactive

GIF decoder to return a response message that defines the graphics parameters for the decoder. This involves returning information about available screen sizes, number of bits/color supported and the amount of color detail supported. The escape sequence for the GCE is defined as:

ESC [> 0 g (g is lower case, spaces inserted for clarity)

(0x1B 0x5B 0x3E 0x30 0x67)

GIF CAPABILITIES RESPONSE

The GIF Capabilities Response message is returned by an interactive GIF decoder and defines the decoder's display capabilities for all graphics modes that are supported by the software. Note that this can also include graphics printers as well as a monitor screen. The general format of this message is:

#version;protocol{;dev, width, height, color-bits, color-res}... <CR>

'#' - GCR identifier character (Number Sign)

version - GIF format version number; initially '87a'

protocol='0' - No end-to-end protocol supported by decoder
Transfer as direct 8-bit data stream.

protocol='1' - Can use an error correction protocol to transfer GIF data interactively from the host directly to the display.

dev = '0' - Screen parameter set follows

dev = '1' - Printer parameter set follows

width - Maximum supported display width in pixels

height - Maximum supported display height in pixels

color-bits - Number of bits per pixel supported. The number of supported colors is therefore 2**color-bits.

color-res - Number of bits per color component supported in the hardware color palette. If color-res is '0' then no hardware palette table is available.

Note that all values in the GCR are returned as ASCII decimal numbers and the message is terminated by a Carriage Return character.

The following GCR message describes three standard EGA configurations with no printer; the GIF data stream can be processed within an error correcting protocol:

```
#87a;1 ;0,320,200,4,0 ;0,640,200,2,2 ;0,640,350,4,2<CR>
```

ENTER GIF GRAPHICS MODE

Two sequences are currently defined to invoke an interactive GIF decoder into action. The only difference between them is that different output media are selected. These sequences are:

```
ESC [ > 1 g   Display GIF image on screen
 (0x1B 0x5B 0x3E 0x31 0x67)
ESC [ > 2 g   Display image directly to an attached graphics  printer.
The image  may optionally be displayed on the screen as
well.
 (0x1B 0x5B 0x3E 0x32 0x67)
```

Note that the 'g' character terminating each sequence is in lower case.

INTERACTIVE ENVIRONMENT

The assumed environment for the transmission of GIF image data from an interactive application is a full 8-bit data stream from host to micro. All 256 character codes must be transferrable. The establishing of an 8-bit data path for communications will normally be taken care of by the host application programs. It is however up to the receiving communications programs supporting GIF to be able to receive and pass on all 256 8-bit codes to the GIF decoder software.

Graphics Interchange Format (GIF)   Page 12

Appendix C - Image Packaging & Compression

The Raster Data stream that represents the actual output image can be represented as:

```
 7 6 5 4 3 2 1 0
+---------------+
|   code size |
+---------------+     ---+
|blok byte count|  |
+---------------+  |
:  :  +-- Repeated as many times as necessary
|  data bytes |  |
:  :  |
+---------------+     ---+
. . .     . . .
+---------------+
|0 0 0 0 0 0 0 0| zero byte count (terminates data stream)
+---------------+
```

The conversion of the image from a series of pixel values to a transmitted or stored character stream involves several steps. In brief these steps are:

1. Establish the Code Size - Define the number of bits needed to represent the actual data.
2. Compress the Data - Compress the series of image pixels to a series of compression codes.
3. Build a Series of Bytes - Take the set of compression codes and convert to a string of 8-bit bytes.
4. Package the Bytes - Package sets of bytes into blocks preceeded by character counts and output.

ESTABLISH CODE SIZE

The first byte of the GIF Raster Data stream is a value indicating the minimum number of bits required to represent the set of actual pixel values. Normally this will be the same as the number of color bits. Because of some algorithmic constraints however, black & white images which have one color bit must be indicated as having a code size of 2. This code size value also implies that the compression codes must start out one bit longer.

COMPRESSION

The LZW algorithm converts a series of data values into a series of codes which may be raw values or a code designating a series of values. Using text characters as an analogy, the output code consists of a character or a code representing a string of characters.

Appendix C - Image Packaging & Compression

The LZW algorithm used in GIF matches algorithmically with the standard LZW algorithm with the following differences:

1. A special Clear code is defined which resets all compression/decompression parameters and tables to a start-up state. The value of this code is 2**<code size>. For example if the code size indicated was 4 (image was 4 bits/pixel) the Clear code value would be 16 (10000 binary). The Clear code can appear at any point in the image data stream and therefore requires the LZW algorithm to process succeeding codes as if a new data stream was starting. Encoders should output a Clear code as the first code of each image data stream.
2. An End of Information code is defined that explicitly indicates the end of the image data stream. LZW processing terminates when this code is encountered. It must be the last code output by the encoder for an image. The value of this code is <Clear code>+1.
3. The first available compression code value is <Clear code>+2.
4. The output codes are of variable length, starting at <code size>+1 bits per code, up to 12 bits per code. This defines a maximum code value of 4095 (hex FFF). Whenever the LZW code value would exceed the current code length, the code length is increased by one. The packing/unpacking of these codes must then be altered to reflect the new code length.

BUILD 8-BIT BYTES

Because the LZW compression used for GIF creates a series of variable length codes, of between 3 and 12 bits each, these codes must be reformed into a series of 8-bit bytes that will be the characters actually stored or transmitted. This provides additional compression of the image. The codes are formed into a stream of bits as if they were packed right to left and then picked off 8 bits at a time to be output. Assuming a character array of 8 bits per character and using 5 bit codes to be packed, an example layout would be similar to:

```
 byte n       byte 5   byte 4 byte 3  byte 2   byte 1
 +-.....-----+--------+--------+--------+--------+--------+
 | and so on |hhhhhggg|ggfffffe|eeeedddd|dcccccbb|bbbaaaaa|
 +-.....-----+--------+--------+--------+--------+--------+
```

Note that the physical packing arrangement will change as the number of bits per compression code change but the concept remains the same.

PACKAGE THE BYTES

Once the bytes have been created, they are grouped into blocks for output by preceeding each block of 0 to 255 bytes with a character count byte. A block with a zero byte count terminates the Raster Data stream for a given image. These blocks are what are actually output for the

GIF image. This block format has the side effect of allowing a decoding program the ability to read past the actual image data if necessary by reading block counts and then skipping over the data.

Appendix D - Multiple Image Processing

Since a GIF data stream can contain multiple images, it is necessary to describe processing and display of such a file. Because the image descriptor allows for placement of the image within the logical screen, it is possible to define a sequence of images that may each be a partial screen, but in total fill the entire screen. The guidelines for handling the multiple image situation are:

1. There is no pause between images. Each is processed immediately as seen by the decoder.
2. Each image explicitly overwrites any image already on the screen inside of its window. The only screen clears are at the beginning and end of the GIF image process. See discussion on the GIF terminator.