

INTERNATIONAL STANDARD

ISO
8632-2

First edition
1987-08-01



INTERNATIONAL ORGANIZATION FOR STANDARDIZATION
ORGANISATION INTERNATIONALE DE NORMALISATION
МЕЖДУНАРОДНАЯ ОРГАНИЗАЦИЯ ПО СТАНДАРТИЗАЦИИ

Information processing systems — Computer graphics — Metafile for the storage and transfer of picture description information —

Part 2 : Character encoding

*Systèmes de traitement de l'information — Infographie — Métafichier de stockage et de transfert
des informations de description d'images —*

Partie 2 : Codage des caractères

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work.

Draft International Standards adopted by the technical committees are circulated to the member bodies for approval before their acceptance as International Standards by the ISO Council. They are approved in accordance with ISO procedures requiring at least 75 % approval by the member bodies voting.

International Standard ISO 8632-2 was prepared by Technical Committee ISO/TC 97, *Information processing systems*.

Users should note that all International Standards undergo revision from time to time and that any reference made herein to any other International Standard implies its latest edition, unless otherwise stated.

CONTENTS

0	Introduction	1
0.1	Purpose of the character encoding	1
0.2	Objectives	1
0.3	Metafile characteristics	1
0.4	Relationship to other International Standards	1
0.5	Status of annex	1
1	Scope and field of application	2
2	References	3
3	Notational conventions	4
3.1	7-Bit and 8-Bit code tables	4
3.2	Code extension techniques vocabulary	5
3.2.1	C0 sets	5
3.2.2	C1 sets	5
3.2.3	G-sets	5
4	Entering and leaving the metafile environment	8
4.1	Implicitly entering the metafile environment	8
4.2	Designating and invoking the CGM coding environment from ISO 2022	8
5	Method of encoding opcodes	9
5.1	Encoding technique of the basic opcode set	9
5.2	Extension mechanism	9
5.3	Opcode assignments	10
6	Method of encoding parameters	13
6.1	Basic format	13
6.2	Bitstream format	14
6.3	Coding integers	14
6.4	Coding real numbers	15
6.5	Coding VDCs and points	16
6.6	Coding point list parameters	17
6.6.1	Displacement mode	17
6.6.2	Incremental mode	17
6.6.3	Incremental mode encoding	21
6.7	Colour specifiers	22
6.8	Colour lists	22
6.8.1	Normal format (coding type=0)	23
6.8.2	Bitstream format (coding type=1)	23
6.8.3	Runlength format (coding type=2)	23
6.8.4	Runlength bitstream format (coding type=3)	24
6.8.5	Examples	24
6.9	String parameters	25
6.9.1	Overall string parameter format	25
6.9.2	Bit combinations permitted within string parameters of text elements	25
6.9.3	C0 control within string parameters	26
6.9.4	Using G-sets in string parameters	26
6.10	Enumerated parameters	27
6.11	Index parameters	27
6.12	Data record parameters	27

7	Character substitution	28
8	Representation of each element	30
8.1	Delimiter elements	31
8.1.1	BEGIN METAFILE	31
8.1.2	END METAFILE	31
8.1.3	BEGIN PICTURE	31
8.1.4	BEGIN PICTURE BODY	31
8.1.5	END PICTURE	31
8.2	Metafile descriptor elements	32
8.2.1	METAFILE VERSION	32
8.2.2	METAFILE DESCRIPTION	32
8.2.3	VDC TYPE	32
8.2.4	INTEGER PRECISION	32
8.2.5	REAL PRECISION	32
8.2.6	INDEX PRECISION	33
8.2.7	COLOUR PRECISION	33
8.2.8	COLOUR INDEX PRECISION	33
8.2.9	MAXIMUM COLOUR INDEX	33
8.2.10	COLOUR VALUE EXTENT	34
8.2.11	METAFILE ELEMENT LIST	34
8.2.12	METAFILE DEFAULTS REPLACEMENT	34
8.2.13	FONT LIST	34
8.2.14	CHARACTER SET LIST	35
8.2.15	CHARACTER CODING ANNOUNCER	36
8.3	Picture descriptor elements	37
8.3.1	SCALING MODE	37
8.3.2	COLOUR SELECTION MODE	37
8.3.3	LINE WIDTH SPECIFICATION MODE	37
8.3.4	MARKER SIZE SPECIFICATION MODE	37
8.3.5	EDGE WIDTH SPECIFICATION MODE	37
8.3.6	VDC EXTENT	37
8.3.7	BACKGROUND COLOUR	38
8.4	Control elements	39
8.4.1	VDC INTEGER PRECISION	39
8.4.2	VDC REAL PRECISION	39
8.4.3	AUXILIARY COLOUR	40
8.4.4	TRANSPARENCY	40
8.4.5	CLIP RECTANGLE	40
8.4.6	CLIP INDICATOR	40
8.5	Graphical primitive elements	41
8.5.1	POLYLINE	41
8.5.2	DISJOINT POLYLINE	41
8.5.3	POLYMARKER	41
8.5.4	TEXT	41
8.5.5	RESTRICTED TEXT	41
8.5.6	APPEND TEXT	42
8.5.7	POLYGON	42
8.5.8	POLYGON SET	42
8.5.9	CELL ARRAY	42
8.5.10	GENERALIZED DRAWING PRIMITIVE	44
8.5.11	RECTANGLE	44
8.5.12	CIRCLE	44
8.5.13	CIRCULAR ARC 3 POINT	44
8.5.14	CIRCULAR ARC 3 POINT CLOSE	44
8.5.15	CIRCULAR ARC CENTRE	44

8.5.16	CIRCULAR ARC CENTRE CLOSE	45
8.5.17	ELLIPSE	45
8.5.18	ELLIPTICAL ARC	45
8.5.19	ELLIPTICAL ARC CLOSE	45
8.6	Attribute elements	46
8.6.1	LINE BUNDLE INDEX	46
8.6.2	LINE TYPE	46
8.6.3	LINE WIDTH	46
8.6.4	LINE COLOUR	46
8.6.5	MARKER BUNDLE INDEX	46
8.6.6	MARKER TYPE	47
8.6.7	MARKER SIZE	47
8.6.8	MARKER COLOUR	47
8.6.9	TEXT BUNDLE INDEX	47
8.6.10	TEXT FONT INDEX	47
8.6.11	TEXT PRECISION	47
8.6.12	CHARACTER EXPANSION FACTOR	48
8.6.13	CHARACTER SPACING	48
8.6.14	TEXT COLOUR	48
8.6.15	CHARACTER HEIGHT	48
8.6.16	CHARACTER ORIENTATION	48
8.6.17	TEXT PATH	48
8.6.18	TEXT ALIGNMENT	49
8.6.19	CHARACTER SET INDEX	49
8.6.20	ALTERNATE CHARACTER SET INDEX	49
8.6.21	FILL BUNDLE INDEX	49
8.6.22	INTERIOR STYLE	49
8.6.23	FILL COLOUR	50
8.6.24	HATCH INDEX	50
8.6.25	PATTERN INDEX	50
8.6.26	EDGE BUNDLE INDEX	50
8.6.27	EDGE TYPE	50
8.6.28	EDGE WIDTH	51
8.6.29	EDGE COLOUR	51
8.6.30	EDGE VISIBILITY	51
8.6.31	FILL REFERENCE POINT	51
8.6.32	PATTERN TABLE	51
8.6.33	PATTERN SIZE	52
8.6.34	COLOUR TABLE	52
8.6.35	ASPECT SOURCE FLAGS	52
8.7	Escape elements	54
8.7.1	ESCAPE	54
8.7.2	DOMAIN RING	54
8.8	External elements	55
8.8.1	MESSAGE	55
8.8.2	APPLICATION DATA	55
9	Defaults	56
10	Conformance	57
A	Formal grammar	58

STANDARDSISO.COM : Click to view the full PDF of ISO 8632-2:1987

Information processing systems — Computer graphics — Metafile for the storage and transfer of picture description information —

Part 2 : Character encoding

0 Introduction

0.1 Purpose of the character encoding

The Character Encoding of the Computer Graphics Metafile (CGM) provides a representation of the Metafile syntax intended for situations in which it is important to minimize the size of the metafile or transmit the metafile through character-oriented communications services. The encoding uses compact representation of data that is optimized for storage or transfer between computer systems.

If minimizing the processing overhead is more important than data compaction, an encoding such as the Binary Encoding contained in ISO 8632/3 may be more appropriate. If human readability is the most important criterion, an encoding such as the Clear Text Encoding in ISO 8632/4 may be more appropriate.

0.2 Objectives

This encoding was designed with the following objectives:

- a) regular syntax: All elements of the metafile should be encoded in a uniform way so that parsing the metafile is simple;
- b) compactness: The encoding should provide a highly compact metafile, suitable for systems with restricted storage capacity or transfer bandwidth;
- c) extensibility: the encoding should allow for future extensions;
- d) transportability: the encoding should be suitable for use with transport mechanisms designed for character-oriented data based on a standard national character set derived from ISO 646.

0.3 Metafile characteristics

Each CGM command follows a simple regular syntax. Thus, new commands can be added in a future revision of ISO 8632 such that existing CGM interpreters can recognize (and ignore) the new commands. Also, new operands can be added to existing commands in the future revision of the standard such that existing CGM interpreters can recognize (and ignore) the additional operands.

Each CGM operand follows a simple regular syntax. Operands are variable in length. This permits small values to be represented by the smallest number of bytes.

A certain range of operand values of standard commands have been reserved for private use; the remaining range is either standardized or reserved for future standardization.

0.4 Relationship to other International Standards

The Character Encoding has been developed in collaboration with the ISO subcommittee responsible for character sets and coding, ECMA, and CEPT. The encoding conforms to the rules for code extension specified in ISO 2022 in the category of complete coding system.

The representation of character data in ISO 8632/2 follows the rules of ISO 646 and ISO 2022.

For certain elements, the CGM defines value ranges as being reserved for registration. The values and their meanings will be defined using the established procedures (see ISO 8632/1, sub-clause 4.11.)

0.5 Status of annex

The annex does not form an integral part of this part of ISO 8632/2 but is included for information only.

1 Scope and field of application

This part of ISO 8632 specifies a character encoding of the Computer Graphics Metafile. For each of the elements specified in ISO 8632/1 an encoding is specified.

This encoding of the Computer Graphics Metafile provides a highly compact representation of the metafile, suitable for applications that require the metafile to be of minimum size and suitable for transmission with character-oriented transmission services.

STANDARDSISO.COM : Click to view the full PDF of ISO 8632-2:1987

2 References

- ISO 646, *Information processing—ISO 7-bit coded character set for information interchange.*
- ISO 2022, *Information processing—ISO 7-bit and 8-bit coded character sets—Code extension techniques.*
- ISO 2375, *Data processing—Procedure for registration of escape sequences.*
- ISO 6429, *Information processing—ISO 7-bit and 8-bit coded character sets—Additional control functions for character-imaging devices.*
- ECMA 96, *Graphics Data Syntax for a multiple Workstation Interface.*
- CEPT, *Rev. of T/CD 6.1 Videotex Presentation Layer Data Syntax.*

STANDARDSISO.COM : Click to view the full PDF of ISO 8632-2:1987

3 Notational conventions

3.1 7-Bit and 8-Bit code tables

The bits of the bit combinations of the 7-bit code are identified by b7, b6, b5, b4, b3, b2, and b1, where b7 is the highest-order, or most-significant, bit and b1 is the lowest-order, or least-significant, bit.

The bit combinations may be interpreted to represent integers in the range 0 to 127 in binary notation by attributing the following weights to the individual bits:

Bit:	b7	b6	b5	b4	b3	b2	b1
Weight:	64	32	16	8	4	2	1

In ISO 8632/2, the bit combinations are identified by notation of the form x/y , where x is a number in the range 0 to 7 and y is a number in the range 0 to 15. The correspondence between the notations of the form x/y and the bit combinations consisting of the bits b7 to b1 is as follows:

- x is the number represented by b7, b6, and b5 where these bits are given the weights 4, 2, and 1 respectively;
- y is the number represented by b4, b3, b2, and b1 where these bits are given the weights 8, 4, 2, and 1 respectively.

The notations of the form x/y are the same as those used to identify code table positions, where x is the column number and y is the row number.

A 7-bit code table consists of 128 positions arranged in eight columns and sixteen rows. The columns are numbered 0 to 7 and the rows are numbered 0 to 15. Figure 1 shows a 7-bit code table.

An example illustrates the 7-bit code: "1/11" refers to the bit combination in column 1, row 11 of the code table, binary 0011011.

The bits of the bit combinations of the 8-bit code are identified by b8, b7, b6, b5, b4, b3, b2, and b1, where b8 is the highest-order, or most-significant, bit and b1 is the lowest-order, or least-significant, bit.

The bit combinations may be interpreted to represent integers in the range 0 to 255 in binary notation by attributing the following weights to the individual bits:

Bit:	b8	b7	b6	b5	b4	b3	b2	b1
Weight:	128	64	32	16	8	4	2	1

Using these weights, the bit combinations of the 8-bit code are interpreted to represent numbers in the range 0 to 255.

In ISO 8632/2, the bit combinations are identified by notation of the form xx/yy , where xx and yy are numbers in the range 00 to 15. The correspondence between the notations of the form xx/yy and the bit combinations consisting of the bits b8 to b1 is as follows:

- xx is the number represented by b8, b7, b6, and b5 where these bits are given the weights 8, 4, 2, and 1 respectively;
- yy is the number represented by b4, b3, b2, and b1 where these bits are given the weights 8, 4, 2, and 1 respectively.

The notations of the form xx/yy are the same as those used to identify code table positions, where xx is the column number and yy is the row number. An 8-bit code table consists of 256 positions arranged in sixteen columns and sixteen rows. The columns and rows are numbered 00 to 15. Figure 2 shows an 8-bit code table.

Notational conventions

7-Bit and 8-Bit code tables

An example illustrates the 8-bit code: 04/01 represents the 8-bit byte 01000001, whereas 4/1 represents the 7-bit byte 1000001.

3.2 Code extension techniques vocabulary

In describing the characters that may occur within string parameters, certain terms imported from other standards (e.g., ISO 2022) are useful. In the context of the CGM, these terms, and the concepts to which they refer, apply only within the string parameters of the TEXT, APPEND TEXT, and RESTRICTED TEXT metafile elements.

3.2.1 C0 sets

A C0 set is a set of 30 control characters represented in a 7-bit code by 0/0 to 1/15, except 0/14 and 0/15 which shall be unused, and in an 8-bit code by 00/00 to 01/15, except 00/14 and 00/15 which shall be unused. C0 sets occupy columns 0 and 1 of a 7-bit code table or columns 00 and 01 of an 8-bit code table. The meanings of C0 controls within string parameters are described in 6.9.3.

3.2.2 C1 sets

A C1 set is a set of up to 32 control characters represented by bit combinations 08/00 to 09/15 in an 8-bit code. C1 sets occupy columns 08 and 09 of the 8-bit code table. In a 7-bit code the C1 control functions are represented by 2-byte escape sequences. This CGM encoding reserves the bit combinations 9/8 and 9/12 (ESC 5/8 and ESC 5/12 in a 7-bit environment, ESC = 1/11); these shall not be part of the content of string parameters. Other C1 control characters from other standards, such as ISO 6429, may be used within string parameters by agreement between the interchanging parties.

3.2.3 G-sets

The G-sets (G0, G1, G2, G3) are coded character sets of 94 or 96 characters. CHARACTER SET INDEX designates which character set is to be the G0 set. ALTERNATE CHARACTER SET INDEX designates a character set to be used as both the G1 and G2 sets. The G-sets may be "invoked into" (caused to occupy) columns 2 through 7 of a 7-bit code table, or columns 02 through 07 and 10 through 15 of an 8-bit code table. This encoding of the CGM uses the G0 and G1/G2 sets within string parameters. The G3 set may be used within the string parameters of conforming metafiles; this requires selection of the extended 7-bit or extended 8-bit mode in the CHARACTER CODING ANNOUNCER. The CGM does not provide an element to explicitly designate the G3 sets; this may be done within a text string in accordance with ISO 2022, or by other means agreed upon by the interchanging parties.

					0	0	0	0	1	1	1	1
					0	0	1	1	0	0	1	1
					0	1	2	3	4	5	6	7
Bits	b ₄	b ₃	b ₂	b ₁	ROW	THE CO SET						
	0	0	0	0	0							
	0	0	0	1	1							
	0	0	1	0	2							
	0	0	1	1	3							
	0	1	0	0	4							
	0	1	0	1	5							
	0	1	1	0	6							
	0	1	1	1	7							
	1	0	0	0	8							
	1	0	0	1	9							
	1	0	1	0	10							
	1	0	1	1	11							
	1	1	0	0	12							
	1	1	0	1	13							
	1	1	1	0	14							
	1	1	1	1	15							
						A G-SET OF 94 OR 96 BIT COMBINATIONS						

Figure 1. The 7-bit code table.

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
0	C0 SET	"GL" SET OF 94 OR 96 BIT COMBINATIONS							C1 SET							
1																
2																
3																
4																
5																
6																
7																
8																
9																
10																
11																
12																
13																
14																
15																

Figure 2. The 8-bit code table.

4 Entering and leaving the metafile environment

4.1 Implicitly entering the metafile environment

The CGM coding environment may be entered implicitly, by agreement between the interchanging parties. This is suitable only if there is not to be any interchange with services using other coding techniques.

4.2 Designating and invoking the CGM coding environment from ISO 2022

For interchange with services using the code extension techniques of ISO 2022, the CGM coding environment shall be designated and invoked from ISO 2022 environment by the following escape sequence:

ESC 2/5 F

where ESC is the bit combination 1/11, and F refers to a bit combination that will be assigned by the ISO Registration Authority for ISO 2375.

The first bit combination occurring after this escape sequence will then represent the opcode of a CGM metafile element.

After the end of one or more metafiles (i.e., after the END METAFILE element) or between pictures (i.e., after the END PICTURE element), the following escape sequence may be used to return to the ISO 2022 coding environment:

ESC 2/5 4/0

This not only returns to the ISO 2022 coding environment, but also restores the designation and invocation of coded character sets to the state that existed prior to entering the CGM coding environment with the ESC 2/5 F sequence. (The terms "designation" and "invocation" are defined in ISO 2022.)

5 Method of encoding opcodes

Each metafile element is composed of one opcode and parameters as required. The opcodes are coded as a sequence of bit combinations from columns 2 and 3 of the code chart. The encoding technique supplies:

- the basic opcode set;
- extension opcode sets.

5.1 Encoding technique of the basic opcode set

The basic opcode set consists of single-byte and double-byte opcodes. Single-byte opcodes are from column 2 of the code chart. Bits b4 to b1 are used to encode the opcode. The format is as follows:

```

b8          b1
+---+-----+-----+
|X|0 1 0|b b b b|
+---+-----+-----+

```

The "X" bit (bit b8) is the parity bit (or omitted bit) in a 7-bit environment. In an 8-bit environment it is 0. For double-byte opcodes the first byte is from column 3 and the second byte is from column 2 or 3 of the code chart. Bits b4 to b1 of the first byte and bits b5 to b1 of the second byte are used to encode the opcode:

```

b8          b1  b8          b1
+---+-----+-----+ +---+-----+-----+
|X|0 1 1|b b b b| |X|0 1|b b b b b|
+---+-----+-----+ +---+-----+-----+

```

The bit combination 3/15, the EXTEND OPCODE SPACE (EOS) allows extension of the basic opcode space (see 5.2).

The basic opcode set, supplied by this encoding technique consists of 496 opcodes, being:

- 16 single-byte opcodes (from column 2);
- 15 x 32 = 480 double-byte opcodes (first byte from column 3 except bit combination 3/15, second byte from column 2 or 3).

5.2 Extension mechanism

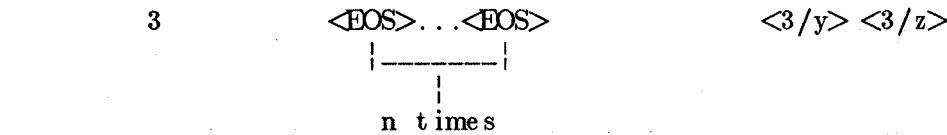
The basic opcode set can be extended with an unlimited number of extension opcode sets by means of the EXTEND OPCODE SPACE code (EOS, 3/15).

The N-th extension opcode set consists of opcodes of the basic opcode set, prefixed with n times the code EOS. The three possible formats of an opcode from the N-th extension opcode set are:

Opcode format	Extension codes	Basic opcode set codes
1	$\langle \text{EOS} \rangle \dots \langle \text{EOS} \rangle$ ----- n times	$\langle 2/x \rangle$
2	$\langle \text{EOS} \rangle \dots \langle \text{EOS} \rangle$ ----- n times	$\langle 3/y \rangle \langle 2/z \rangle$

Extension mechanism

Method of encoding opcodes



- <EOS> = 3/15
- x = 0,1,...,15
- y = 0,1,...,14
- z = 0,1,...,15
- n = 0,1,.....

- n = 0 selects the basic opcode set.
- n = 1 selects the first extension opcode set.
- n = N selects the N-th extension opcode set.

The number of opcodes supplied by this encoding technique (basic opcode set plus extension opcode sets) is $496 \cdot (n+1)$, where n is the number of extension sets. (Each extension set has 496 opcodes — 16 single-byte opcodes plus 480 double-byte codes.)

5.3 Opcode assignments

Table 1 lists the opcode assignments for the CGM elements. All opcodes are from the basic opcode set. They have be organized as follows: single-byte opcodes are assigned to Graphical Primitive elements except for some of the Circular and Elliptical output elements where double-byte opcodes with combination 3/4 for the first byte have been assigned. All other metafile elements have double-byte opcodes with the following bit combinations for the first byte:

3/0	for Delimiter Elements
3/1	for Metafile Descriptor Elements
3/2	for Picture Descriptor Elements
3/3	for Control Elements
3/5 and 3/6	for Attribute Elements
3/7	for Escape and External Elements

Method of encoding opcodes

Opcode assignments

Table 1 — Opcodes for metafile elements.

Opcode	7-Bit coding		8-Bit coding	
BEGIN METAFILE opcode	3/0	2/0	03/0	02/0
END METAFILE opcode	3/0	2/1	03/0	02/1
BEGIN PICTURE opcode	3/0	2/2	03/0	02/2
BEGIN PICTURE BODY opcode	3/0	2/3	03/0	02/3
END PICTURE opcode	3/0	2/4	03/0	02/4
METAFILE VERSION opcode	3/1	2/0	03/1	02/0
METAFILE DESCRIPTION opcode	3/1	2/1	03/1	02/1
VDC TYPE opcode	3/1	2/2	03/1	02/2
INTEGER PRECISION opcode	3/1	2/3	03/1	02/3
REAL PRECISION opcode	3/1	2/4	03/1	02/4
INDEX PRECISION opcode	3/1	2/5	03/1	02/5
COLOUR PRECISION opcode	3/1	2/6	03/1	02/6
COLOUR INDEX PRECISION opcode	3/1	2/7	03/1	02/7
MAXIMUM COLOUR INDEX opcode	3/1	2/8	03/1	02/8
COLOUR VALUE EXTENT opcode	3/1	2/9	03/1	02/9
METAFILE ELEMENT LIST opcode	3/1	2/10	03/1	02/10
BEGIN METAFILE DEFAULTS REPLACEMENT opcode	3/1	2/11	03/1	02/11
END METAFILE DEFAULTS REPLACEMENT opcode	3/1	2/12	03/1	02/12
FONT LIST opcode	3/1	2/13	03/1	02/13
CHARACTER SET LIST opcode	3/1	2/14	03/1	02/14
CHARACTER CODING ANNOUNCER opcode	3/1	2/15	03/1	02/15
SCALING MODE opcode	3/2	2/0	03/2	02/0
COLOUR SELECTION MODE opcode	3/2	2/1	03/2	02/1
LINE WIDTH SPECIFICATION MODE opcode	3/2	2/2	03/2	02/2
MARKER SIZE SPECIFICATION MODE opcode	3/2	2/3	03/2	02/3
EDGE WIDTH SPECIFICATION MODE opcode	3/2	2/4	03/2	02/4
VDC EXTENT opcode	3/2	2/5	03/2	02/5
BACKGROUND COLOUR opcode	3/2	2/6	03/2	02/6
VDC INTEGER PRECISION opcode	3/3	2/0	03/3	02/0
VDC REAL PRECISION opcode	3/3	2/1	03/3	02/1
AUXILIARY COLOUR opcode	3/3	2/2	03/3	02/2
TRANSPARENCY opcode	3/3	2/3	03/3	02/3
CLIP RECTANGLE opcode	3/3	2/4	03/3	02/4
CLIP INDICATOR opcode	3/3	2/5	03/3	02/5
POLYLINE opcode	2/0		02/0	
DISJOINT POLYLINE opcode	2/1		02/1	
POLYMARKER opcode	2/2		02/2	
TEXT opcode	2/3		02/3	
RESTRICTED TEXT opcode	2/4		02/4	
APPEND TEXT opcode	2/5		02/5	
POLYGON opcode	2/6		02/6	
POLYGON SET opcode	2/7		02/7	
CELL ARRAY opcode	2/8		02/8	
GENERALIZED DRAWING PRIMITIVE opcode	2/9		02/9	
RECTANGLE opcode	2/10		02/10	

Table 1 — Opcodes for metafile elements (concluded).

Opcode	7-Bit coding		8-Bit coding	
CIRCLE opcode	3/4	2/0	03/4	02/0
CIRCULAR ARC 3 POINT opcode	3/4	2/1	03/4	02/1
CIRCULAR ARC 3 POINT CLOSE opcode	3/4	2/2	03/4	02/2
CIRCULAR ARC CENTRE opcode	3/4	2/3	03/4	02/3
CIRCULAR ARC CENTRE CLOSE opcode	3/4	2/4	03/4	02/4
ELLIPSE opcode	3/4	2/5	03/4	02/5
ELLIPTICAL ARC opcode	3/4	2/6	03/4	02/6
ELLIPTICAL ARC CLOSE opcode	3/4	2/7	03/4	02/7
LINE BUNDLE INDEX opcode	3/5	2/0	03/5	02/0
LINE TYPE opcode	3/5	2/1	03/5	02/1
LINE WIDTH	3/5	2/2	03/5	02/2
LINE COLOUR opcode	3/5	2/3	03/5	02/3
MARKER BUNDLE INDEX opcode	3/5	2/4	03/5	02/4
MARKER TYPE opcode	3/5	2/5	03/5	02/5
MARKER SIZE opcode	3/5	2/6	03/5	02/6
MARKER COLOUR opcode	3/5	2/7	03/5	02/7
TEXT BUNDLE INDEX opcode	3/5	3/0	03/5	03/0
TEXT FONT INDEX opcode	3/5	3/1	03/5	03/1
TEXT PRECISION opcode	3/5	3/2	03/5	03/2
CHARACTER EXPANSION FACTOR opcode	3/5	3/3	03/5	03/3
CHARACTER SPACING opcode	3/5	3/4	03/5	03/4
TEXT COLOUR opcode	3/5	3/5	03/5	03/5
CHARACTER HEIGHT opcode	3/5	3/6	03/5	03/6
CHARACTER ORIENTATION opcode	3/5	3/7	03/5	03/7
TEXT PATH opcode	3/5	3/8	03/5	03/8
TEXT ALIGNMENT opcode	3/5	3/9	03/5	03/9
CHARACTER SET INDEX opcode	3/5	3/10	03/5	03/10
ALTERNATE CHARACTER SET INDEX opcode	3/5	3/11	03/5	03/11
FILL BUNDLE INDEX opcode	3/6	2/0	03/6	02/0
INTERIOR STYLE opcode	3/6	2/1	03/6	02/1
FILL COLOUR opcode	3/6	2/2	03/6	02/2
HATCH INDEX opcode	3/6	2/3	03/6	02/3
PATTERN INDEX opcode	3/6	2/4	03/6	02/4
EDGE BUNDLE INDEX opcode	3/6	2/5	03/6	02/5
EDGE TYPE opcode	3/6	2/6	03/6	02/6
EDGE WIDTH opcode	3/6	2/7	03/6	02/7
EDGE COLOUR opcode	3/6	2/8	03/6	02/8
EDGE VISIBILITY opcode	3/6	2/9	03/6	02/9
FILL REFERENCE POINT opcode	3/6	2/10	03/6	02/10
PATTERN TABLE opcode	3/6	2/11	03/6	02/11
PATTERN SIZE opcode	3/6	2/12	03/6	02/12
COLOUR TABLE opcode	3/6	3/0	03/6	03/0
ASPECT SOURCE FLAGS opcode	3/6	3/1	03/6	03/1
ESCAPE opcode	3/7	2/0	03/7	02/0
DOMAIN RING opcode	3/7	3/0	03/7	03/0
MESSAGE opcode	3/7	2/1	03/7	02/1
APPLICATION DATA opcode	3/7	2/2	03/7	02/2

6 Method of encoding parameters

The parameter part of a CGM element may contain one or more parameters, each parameter consisting of one or more bytes.

All parameters are coded in columns 4 through 7. (However, the coded representation of a 'string' parameter may include bit combinations from other columns of the code table - see the description of string parameters in 6.9). The general format of a parameter byte is:

```

b8          b1
+-+-----+
|X|1|b b b b b|
+-+-----+

```

The "X" (bit b8) is the parity bit (or omitted bit) in a 7-bit environment. In an 8-bit environment it is 0. Bit b7 is the parameter flag.

Except for 'string' and 'data record' parameters all parameters are encoded using one or both of two formats, Basic format or Bitstream format.

6.1 Basic format

Each Basic format parameter is coded as a sequence of one or more bytes, structured as follows:

```

b8          b1
+++++-----+
|X|1|e|s|b b b b| first byte
+++++-----+

```

⋮

```

b8          b1
+++++-----+
|X|1|e|b b b b b| last byte
+++++-----+

```

The "X" (bit b8) is the parity bit (or omitted bit) in a 7-bit environment. In an 8-bit environment it is 0. Bit b7 is the parameter flag.

"e" (b6 of each byte) is the extension flag. For single byte parameters, the extension flag is 0. In multi-byte parameters, the extension flag is 1 in all bytes except the last byte, where it is 0.

Bits b5 through b1 are the data bits of the parameter.

"s" is the sign bit; if equal to 0 then the integer is non-negative and if equal to 1 then the integer is negative. The number zero shall always be coded as "plus zero" 4/0. The "minus zero" coding is reserved for special usage (see 6.6.3).

The Basic format is used to encode:

- enumerated types (E);
- colour indices (CI);
- indices other than colour indices (IX);
- integers (I);
- real numbers (R);
- non-incremental coordinates.

Basic format**Method of encoding parameters**

The most significant part of the parameter is coded in the first byte. The least significant part of the parameter is coded in the last byte.

6.2 Bitstream format

Each Bitstream format parameter is encoded as a sequence of one or more bytes, structured as follows:

```

b8                b1
+---+-----+
|X|1|b b b b b b| first byte
+---+-----+

```

```

b8                b1
+---+-----+
|X|1|b b b b b b| last byte
+---+-----+

```

The "X" is the parity bit (or omitted bit) in a 7-bit environment. In an 8-bit environment it is 0. Bit b7 is the parameter flag.

Bits b6 through b1 are the data bits of the parameter.

The Bitstream format is used to encode:

- a) incremental mode coordinates (see 6.6.2);
- b) colour direct (see 6.8);
- c) colour index lists (see 6.8).

Bitstream data are packed in consecutive databits starting from high-numbered bits to lower-numbered bits of the first byte for the most significant part of the bitstream data.

The end of a Bitstream format parameter cannot be derived from the Bitstream format itself (the format is not self-delimiting). Instead,

- for incremental mode coordinates, the end of the data (which identifies the end of the Bitstream format parameter) is identified by the <End of Block> code;
- for colour index lists, the number of bits needed to encode the colour index list (which identifies the end of the Bitstream format parameter) is set by the COLOUR INDEX PRECISION element, or by the LOCAL COLOUR PRECISION parameter (for those elements which contain such a parameter);
- for colour direct data, the number of bits needed to encode the data (which identifies the end of the Bitstream format operand) is set by the COLOUR PRECISION element, or by the LOCAL COLOUR PRECISION parameter (for those elements which contain such a parameter).

6.3 Coding integers

Integers are coded as sequences of bytes in the range from 4/0 to 7/15 in the Basic format. If a byte is from columns 4 or 5 of the code table, it is either the last byte in the integer's coded representation or it is a single-byte integer. A multi-byte integer begins with a byte from columns 6 or 7 of the code table.

The structure of integer parameters is as illustrated for Basic format (see 6.1). "bbb..." are bits representing the magnitude of the integer. The most significant part of the parameter is coded in the first byte. The least significant part of the parameter is coded in the last byte.

Any integer can be coded with leading most significant bits which are all zero. For example, 4/3 and 6/0 6/0 4/3 are both valid codings for the integer "+3"; however, efficient metafile generators should avoid such redundant codings.

Method of encoding parameters

Coding integers

The size of integer parameters is limited by the current INTEGER PRECISION value.

Integers in the range of -15 to +15 can be coded as single bytes.

(integer: +1) = 4/1	(integer: -1) = 5/1
(integer: +15) = 4/15	(integer: -15) = 5/15

Larger integers require more bytes.

(integer: +16) = 6/0 5/0	(integer: -16) = 7/0 5/0
(integer: +1034) = 6/1 6/0 4/10	(integer: -1034) = 7/1 6/0 4/10

6.4 Coding real numbers

Each real number is coded as an integer mantissa followed by an optional exponent, both coded in the Basic format. The exponent is the power of two by which the integer mantissa is to be multiplied.

The exponent may be implicitly defined as a default exponent, which is then omitted in the Real format, or the exponent may be coded explicitly as the second part of the Real format:

<real format> = <mantissa part> [<exponent part>]

Depending on the "exponent allowed" parameter of the REAL PRECISION element, one of the bits in the first byte of the mantissa tells whether the exponent follows. If the "exponent follows" bit in the mantissa is zero, or if REAL PRECISION has specified that there is to be no "exponent follows" bit, then the exponent is omitted and a default value is assumed which is set by another parameter of the REAL PRECISION element.

Note that the coding of real VDC coordinates is controlled by VDC REAL PRECISION, not by REAL PRECISION, and that the rules for default exponents in real VDC coordinates are slightly different (see 6.5 and 6.6).

The mantissa is an integer which is coded in the Basic format. The first byte takes one of two forms, depending on whether or not REAL PRECISION has specified that an "exponent follows" bit is to be included. The format is as follows (first byte):

b8	b1	
+++++-----+		{if explicit exponent allowed = 'allowed',
X 1 e s p b b b		i.e. the exponent-follows bit is present}
+++++-----+		

or

b8	b1	
+++++-----+		{if explicit exponent allowed = 'forbidden',
X 1 e s b b b b		i.e. there is no exponent-follows bit}
+++++-----+		

b8	b1	
+++++-----+		
X 1 e b b b b		{last byte}
+++++-----+		

"e" is the extension flag, see 6.1 for a description.

"s" is the sign bit (bit b5 of the first byte): 0 for a positive mantissa, 1 for a negative mantissa.

Here "p" is the exponent-follows bit. If the current "explicit exponent allowed" value of the REAL PRECISION element is 'allowed', bit b4 of the first byte of a mantissa is used as the "exponent follows" bit:

Coding real numbers

Method of encoding parameters

1 if an explicit exponent follows, 0 if no exponent follows the mantissa (then the default exponent set by REAL PRECISION is assumed). If the current "explicit exponent allowed" value of the REAL PRECISION element is 'forbidden', bit b4 of the first byte is used as a databit and the default exponent set by REAL PRECISION is assumed.

The exponent is coded as an integer in Basic format, see 6.3.

Mantissas or exponents of "minus zero" are not allowed and reserved for future use.

For example, suppose that REAL PRECISION has specified that each real parameter is to be coded with an "exponent follows" bit in its mantissa. In that case, the binary numeral +1.110010110011 is coded as follows:

```
(real: binary +1.11 00101 10011)

= (mantissa: +111 00101 10011, "exponent-follows")
  (exponent: -12)

=
+---+---+---+---+---+---+---+---+---+---+---+---+
|X|1|1|0|1|1 1 1| |X|1|1|0 0 1 0 1| |X|1|0|1 0 0 1 1| {mantissa}
+---+---+---+---+---+---+---+---+---+---+---+---+
+---+---+---+---+---+---+
|X|1|0|1|1 1 0 0| {exponent}
+---+---+---+---+---+---+

= 6/15 6/5 5/3 5/12
```

Again, suppose that REAL PRECISION has specified that real parameters are to be coded without exponents and without exponent-follows bits in their mantissas, and that the default exponent is to be -13. In that case, binary +1.110010110011 is coded as follows:

```
(real: binary +1.110010110011)

= (mantissa: +1110 01011 00110)

=
+---+---+---+---+---+---+---+---+---+---+---+---+
|X|1|1|0|1 1 1 0| |X|1|1|0 1 0 1 1| |X|1|0|0 0 1 1 0| {mantissa}
+---+---+---+---+---+---+---+---+---+---+---+---+

= 6/14 6/11 4/6
```

6.5 Coding VDCs and points

A point is a pair of VDC scalars. A VDC scalar is either an integer or real number according to whether VDC TYPE is integer or real.

When VDC TYPE is integer, the encodings of the VDC and point data types are as described in 6.3, Coding Integers. The size of the VDC and point parameters is limited by the current VDC INTEGER PRECISION value.

When VDC TYPE is real, the encodings of the VDC and point data types are as described in 6.4, Coding Real Numbers. The size of the VDC and point parameters is limited by the current VDC REAL PRECISION value. Whether or not "exponent follows" bits are included in the mantissas of VDCs and points is determined by a parameter of VDC REAL PRECISION. The default value for an omitted exponent in a VDC is determined by the "default exponent" parameter in VDC REAL PRECISION. If the exponent is omitted from a point data type, a default exponent is assumed as follows.

- If the point is not in a point list, or is the first point of a point list, the omitted exponent assumes the default value exponents of VDC parameters, as specified by VDC REAL PRECISION.

Method of encoding parameters

Coding VDCs and points

- If the point is in a point list, but is not the first point of that point list, an omitted exponent may assume a different value as follows: if the exponent is omitted from the x-component of a real point, it defaults to the value of the exponent in the preceding x-component; similarly, an exponent omitted from a y-component assumes the value of the exponent in the preceding y-coordinate.

6.6 Coding point list parameters

Point lists may be coded with one or both of the following coding structures. These structures are called 'Displacement Mode' and 'Incremental Mode'. Both formats may be used within a single point list.

Individual points and the first point of each point list are always be coded in Displacement Mode. The displacement values delta-x and delta-y are displacements measured from the origin, i.e. absolute positions. For points after the first point in a point list, each displacement is measured from the preceding point of the point list (this is true regardless of whether the preceding point was coded in Displacement Mode or in Incremental Mode).

6.6.1 Displacement mode

In Displacement Mode, each point is coded as a sequence of two VDCs. The first VDC value gives the x-component of the point's displacement from the preceding point, while the second VDC value specifies the y-component of that displacement.

$$\langle P \rangle = \langle \text{VDC: delta x} \rangle \langle \text{VDC: delta y} \rangle$$

6.6.2 Incremental mode

The Incremental Mode is defined as a Differential Chain Code (DCC). The data in this mode does not reflect actual coordinates, but defines steps (increments) from one coordinate position to another. These increments are identified by points on a Ring. A Ring is a set of points on a square whose centre is the previously identified point. The first centre point is encoded in Displacement Mode.

A Ring is characterized by its Radius (R) in Basic Grid Units (BGU), its Angular resolution (by a factor p) and its Direction (D). The maximum number of points on a Ring is 8R. The actual number of points on a Ring with a given Angular resolution factor p follows from:

$$N = \frac{8R}{2^p}, \quad p=0,1,2,3$$

N is required to be even.

The points on the Ring are numbered, starting at the Direction point D, counter clockwise from 0 to M-1 and clockwise from -1 to -M, with $M=N/2$.

Following are examples of two Rings, both having Radius $R=3$. The Ring on the left has Angular resolution factor $p=0$ and the one on the right has Angular resolution factor $p=1$.

Coding point list parameters

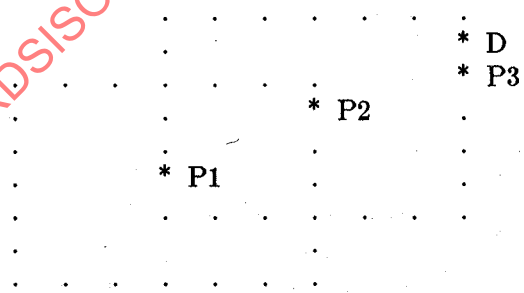
Method of encoding parameters

	8	7	6	5	4		4	3	2	
9	.	.	.	.	.	.	3	.	.	.
10	.					.	2	5	.	1
11	.					.	1			
-12	.		.			.	0	-6	.	0
			centre						centre	
-11	.					.	-1			
-10	.					.	-2	-5	.	-1
-9	.	.	.	.	.	.	-3	.	.	.
	-8	-7	-6	-5	-4		-4	-3	-2	

The Direction of the Ring is given by the position of the point with number ZERO. The initial position of this point is on the positive x-axis of VDC space, while the Cartesian axes are drawn through the centre point of the Ring. The Direction of the Rings following the initial one is dependent on the direction of the increments. This Direction is determined in the following way:

Let P2 be the current centre point and P1 the previous one, i.e., P2 is a point on the Ring with centre in P1. The Direction of the Ring is obtained by drawing a line from P1 through P2; where this direction ray intersects P2's Ring is the ZERO point of that Ring. So the Direction of the Ring is dependent on the direction of the line to be displayed.

In the DCC only the differences between points on the consecutive Rings are coded. The position of point P3, i.e., the increment on the new Ring (centre P2) is described by the difference between the position of point P2 on the previous Ring and the position of the new point P3 on the current Ring, the positions numbered with respect to the previous Ring. As shown in example below, the position of point P3 is defined by the difference: $P3 - P2 = -1$. P3 and P2 being point numbers on the two Rings, numbered as given in the previous figure. The Direction (position of the point with number ZERO) is identified by D.



Change of direction with R=3

The basic Radius of the Ring, as used in Incremental mode, is dependent on the BGU. The BGU is the smallest nonzero value that can be expressed within the precision set by the VDC INTEGER or REAL PRECISION element. For VDC TYPE integer, basic Radius (=BGU) = 1. For VDC TYPE real the size of the BGU is set by a parameter of VDC REAL PRECISION, called smallest-real code: letting "src" stand for smallest-real code, $BGU=2^{src}$. The basic Radius is a multiple of the BGU, i.e. it is a positive integer. For VDC TYPE real the default value for the basic Radius follows from:

Method of encoding parameters

Coding point list parameters

smallest-real code	basic Radius
> g	1
= g	$\frac{1}{2^p}$
< g	BGU

where g is a nominal smallest-real code. A value of $g = -8$ is used in ISO 8632/2.

The basic Radius and the Angular resolution factor may be changed with the DOMAIN RING element. If the basic Radius is set to a value less than ONE, the value ONE is assumed for the basic Radius.

The encoding used in Incremental Mode makes use of the DCC property by using variable length code-words (Huffman Code). The encoding also allows changing of the Radius and the Angular resolution factor. The Radius can have a value of R, 2R, 4R or 8R, where R is the defined Radius. The Angular resolution factor p can be 0, 1, 2, or 3, the default value is 0.

The Huffman Code table (see table 2) used in the Incremental mode is a fixed length table. To allow the encoding of more points on a Ring two Escape codes are defined. With these Escape codes the points outside the Huffman Code table can be addressed. The end of the Incremental mode data is indicated by an End of Block value in the Huffman Code table.

Table 2 — Huffman code table for incremental mode.

Length	Code-word	Point number
2	00	0
2	10	1
2	01	-1
4	1100	2
4	1101	-2
6	111000	3
6	111001	-3
6	111010	4
6	111011	-4
8	11110000	5
8	11110001	-5
8	11110010	6
8	11110011	-6
8	11110100	7
8	11110101	-7
8	11110110	8
8	11110111	-8
10	1111100000	9
10	1111100001	-9
10	1111100010	10
10	1111100011	-10
10	1111100100	11
10	1111100101	-11
10	1111100110	12
10	1111100111	-12
10	1111101000	13
10	1111101001	-13
10	1111101010	14
10	1111101011	-14
10	1111101100	15
10	1111101101	-15
10	1111101110	16
10	1111101111	-16
10	1111110000	17
10	1111110001	-17
10	1111110010	18
10	1111110011	-18
10	1111110100	19
10	1111110101	-19
10	1111110110	C1
10	1111110111	-20
10	1111111000	C2
10	1111111001	C3
10	1111111010	C4
10	1111111011	C5
10	1111111100	C6
10	1111111101	IM-ESC 1
10	1111111110	IM-ESC 2
10	1111111111	End of Block

The <End of Block> code from the Huffman Code table identifies the end of the Incremental mode data. Remaining bits in the last Incremental mode data byte have no meaning; they are required to be zero.

Method of encoding parameters

Coding point list parameters

The Incremental Mode escape codes <IM-ESC 1> and <IM-ESC 2> are used to extend the addressable number of points, e.g. points outside the range -20 to 19. The code <IM-ESC 1> adds +20 or -20 to the following code, depending on the sign of that following point. The code <IM-ESC 2> adds +40 or -40 to the following code, depending on the sign. The escape codes can follow each other in any desired order. The following examples demonstrate some possible combinations, [n] is a point number.

	<IM-ESC 1>	[1]	= point number 21
	<IM-ESC 1>	[-1]	= point number -21
	<IM-ESC 2>	[14]	= point number 54
	<IM-ESC 2>	[-12]	= point number -52
<IM-ESC 1>	<IM-ESC 2>	[6]	= point number (20+40+6) = 66
<IM-ESC 2>	<IM-ESC 1>	[-18]	= point number (-40-20-18) = -78

The codes C1 up to C6 are used to change the parameters that define the Ring to be used. The values of R are taken from the range R0, 2R0, 4R0 and 8R0, where R0 is the value of the Ring Radius before entering the Incremental Mode. The values of p are taken from the range 0, 1, 2 and 3. The function of these codes is as follows:

- C1 Change the Ring parameters, R and p, to the next higher value, e.g. if the Radius is R0 it is set to 2R0, if the Angular resolution is p=0 it is set to p=1. R cannot become greater than 8R0 and p cannot become greater than 3. For example if the current Ring Radius is 8R0 and the current Angular resolution factor 3, the code <C1> has no effect.
- C2 Change the Ring parameters, R and p, to the next lower value. The effect of the code <C2> is the inverse of code <C1>. R cannot become smaller than R0 and p cannot become smaller than 0. For example if the current Radius is R0 and the current Angular resolution factor 0, the code <C2> has no effect.
- C3 Change the Ring Radius R to the next higher value. The code <C3> has no effect if the current Radius is 8R0.
- C4 Change the Angular resolution factor p to the next higher value. The code <C4> has no effect if the current factor is 3.
- C5 Change the Ring Radius R to the next lower value. The code <C5> has no effect if the current Radius is R0.
- C6 Change the Angular resolution factor p to the next lower value. The code <C6> has no effect if the current factor is 0.

In addition, those codes (C1 to C6) set the position of the point with number ZERO on the positive x-axis, while the cartesian axes are drawn through the centre point of the Ring.

6.6.3 Incremental mode encoding

The structure of the incremental mode format is given below:

```

b8          b1
+--+--+--+--+--+--+--+
|X|1|0|1|0|0|0|0| first byte
+--+--+--+--+--+--+--+

+--+--+--+--+--+--+--+
|X|1|0|0|0|0|0|1| second byte
+--+--+--+--+--+--+--+

+--+--+-----+
|X|1|b|b|b|b|b|b| other bytes
+--+--+-----+

```

Coding point list parameters

Method of encoding parameters

The first byte is coded according to the Basic format structure indicating an integer value "minus zero".

The second byte is structured according to the basic format structure. Bit b1 of the second byte is set to 1 to identify the use of Direct Chain Coding (DCC). The bits b5 to b2 are reserved for future use and shall be set to zero. The following bytes are coded according to the Bitstream format (see 6.2).

Because the Incremental mode uses variable length code words, these may not fit in the Incremental mode data bits (bits b6 to b1 of the other bytes). The code words are packed in consecutive bits of the Incremental mode bytes, starting from high numbered bits to lower numbered bits.

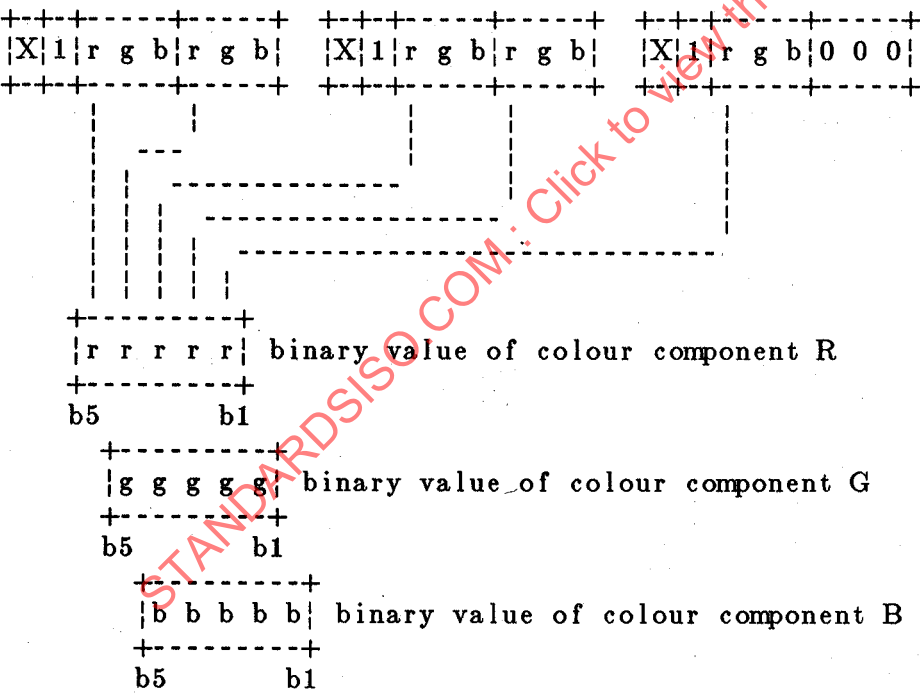
The end of Incremental mode data is identified by the <End of Block> code. Remaining bits in the last Incremental mode data byte have no meaning; they are required to be zero.

6.7 Colour specifiers

Colour specifiers are coded either as colour indexes (if COLOUR SELECTION MODE is 'indexed') or as RGB parameters (if COLOUR SELECTION MODE is 'direct').

A colour index parameter is coded as an integer in the Basic format.

The RGB parameters used if COLOUR SELECTION MODE is 'direct' are coded as a series of bytes in the range of 4/0 to 7/15 (Bitstream format). The six least significant bits in each byte hold binary bits representing the red, green, and blue colour values, starting from high numbered bits to lower numbered bits from the first byte to the last byte. The number of bits needed to encode this data is set by COLOUR PRECISION. If this value is set to N bits, there are 3N colour value bits; i.e. there are only as many bytes in an RGB parameter as are necessary to hold those 3N bits. For example, if COLOUR PRECISION is set to 5 bits, RGB parameters have the following form:



Each byte contains two bits for each component (R,G,B). The first byte contains the most significant bits. If bits b3 to b1 in the last byte are unused they are required to be zero.

6.8 Colour lists

A colour list is a structured data type consisting of a coding type indicator followed by a sequence of colour list elements, each element specifying a colour value (indexed or direct).

Method of encoding parameters

Colour lists

Seven different formats are possible for colour lists, based on the current value of the COLOUR SELECTION MODE and the coding type indicator which precedes the colour values.

This coding type indicator is coded in Basic format and may have values of 0 through 3; these can be viewed as setting two one-bit flags. Bits b5-b3 are reserved for future use, and shall be set to 0. The coding type indicator:

```

      b8                b1
    +--+--+-----+
    |X|1|0|0 0 0 r f|
    +--+--+-----+

```

coding type	runlength(r)	format(f)	"name"
0	0	0	normal
1	0	1	bitstream
2	1	0	runlength
3	1	1	runlength bitstream

Runlength encoding is efficient if many adjacent colour list elements have the same colour value. For each run, the colour of the cells is specified, followed by a count telling how many cells are in the run. This count is always in Basic format; the format of the colour values is described below.

If adjacent colour cells are not the same colour, runlength encoding is less efficient than the normal or bitstream formats.

In the normal formats, colour values are coded as for individual colour parameters (see 6.7).

In the bitstream formats, colour values are packed together as tightly as possible, six bits at a time, in consecutive data bits starting from high-numbered bits to lower-numbered bits, into characters from columns 4 to 7 of the code chart. The parameters are permitted to straddle character boundaries.

Bitstream format parameters are not self-delimiting (see 6.2). Any unused bits in the last character of a Bitstream format parameter have no meaning; they are required to be zero.

6.8.1 Normal format (coding type=0)

When COLOUR SELECTION MODE is 'indexed', the colour list is a sequence of numbers in Basic format, each an individual CI.

When COLOUR SELECTION MODE is 'direct', the colour list is a sequence of bitstreams, each bitstream representing one RGB value.

The normal format is the easiest to produce, and matches the formats used for individual colour values (e.g., LINE COLOUR).

6.8.2 Bitstream format (coding type=1)

When COLOUR SELECTION MODE is 'indexed', the colour list is a single bitstream which is the concatenation of numbers representing CI in the Local Colour Precision. This form is similar to that used for "raster screen dumps" in many applications.

When COLOUR SELECTION MODE is 'direct', the colour list is a single bitstream which is the concatenation of the RGB bitstreams in the Local Colour Precision. This is the most compact form of direct colour list where colours vary greatly within a row.

6.8.3 Runlength format (coding type=2)

When COLOUR SELECTION MODE is 'indexed', the colour list is a sequence of pairs, each pair consisting of [CI in Basic format at Local Colour Precision; number of repetitions in Basic format]. This is

the simplest form of runlength indexed list.

When COLOUR SELECTION MODE is 'direct', the colour list is a sequence of pairs, each pair consisting of [RGB bitstream at Local Colour Precision; number of repetitions in Basic format].

6.8.4 Runlength bitstream format (coding type=3)

When COLOUR SELECTION MODE is 'indexed', the colour list is a sequence of pairs, each pair consisting of [CI in Bitstream format at Local Colour Precision; number of repetitions in Basic format]. The advantage over Runlength format is that the "extend bit" and "sign bit" of the Basic format are available as data bits for the index itself, which means that CI in the range 16..63 may be coded in one character instead of two.

When COLOUR SELECTION MODE is 'direct', this format is identical to the Runlength format.

6.8.5 Examples

The following example shows a colour index list in Runlength coding:

Coding: Runlength
Datatype: Colour Index List
Parameter Values: 2,3,3,3,4

4/2 first byte: r=1,f=0
4/2,4/1 Run Colour 1 (value 2), Run Count 1 (value 1)
4/3,4/4 Run Colour 2 (value 3), Run Count 2 (value 4)
4/4,4/1 Run Colour 3 (value 4), Run count 3 (value 1)

As an example for a colour indexed list, suppose that COLOUR INDEX PRECISION is 5 so that each colour index has 5 bits. Then, the colour indexes are packed into the Bitstream as follows:

```
+--+-----+
|1|A A A A A B|
+--+-----+
+--+-----+
|1|B B B B C C|
+--+-----+
+--+-----+
|1|C C C D D D|
+--+-----+
+--+-----+
|1|D D 0 0 0 0|
+--+-----+
```

Note that AAAAA is the binary numeral for the first colour index, BBBBB is the binary numeral for the second colour index, and so on.

The leftover bits in the last byte have no meaning; they are required to be zero.

The following example shows the Bitstream coding with the same colour index list as used in the Runlength coding:

Coding: Bitstream
Datatype: Colour Index List
Parameter Values: 2,3,3,3,4 (COLOUR INDEX PRECISION = 4)

Method of encoding parameters

Colour lists

- 4/1 first byte : $r=0, f=1$
- 4/8 Colour Index 1 and (half of) 2
- 7/3 Colour Index (half of) 2 and 3
- 4/12 Colour Index 4 and (half of) 5
- 7/4 Colour Index (half of) 5 and 6

In the Basic format the same colour index list would occupy more bytes, as shown in the following example:

Coding: Basic Format
 Datatype: Colour Index List
 Parameter Values: 2,3,3,3,3,4

- 4/0 first byte : $r=0, f=0$
- 4/2 Colour Index 1 (value 2)
- 4/3 Colour Index 2 (value 3)
- 4/3 Colour Index 3 (value 3)
- 4/3 Colour Index 4 (value 3)
- 4/3 Colour Index 5 (value 3)
- 4/4 Colour Index 6 (value 4)

6.9 String parameters

6.9.1 Overall string parameter format

Strings are coded as sequences of bytes, starting with START OF STRING (SOS) and terminated by STRING TERMINATOR (ST).

SOS is represented by the C1 control 9/8. In a 7-bit environment SOS is coded ESC 5/8 (where ESC is the bit combination 1/11). ST is coded as ESC 5/12 in a 7-bit environment, and 9/12 in an 8-bit environment.

	7-bit environment	8-bit environment
SOS	1/11 5/8	09/08
ST	1/11 5/12	09/12

6.9.2 Bit combinations permitted within string parameters of text elements

The following bit combinations are allowed as bytes in string parameters in a conforming metafile:

- a) Bit combinations from columns 2 through 7 of a 7-bit or 8-bit code chart. If the CHARACTER CODING ANNOUNCER has selected an 8-bit coding technique, then bit combinations from columns 10 through 15 of an 8-bit code chart are also permitted within string parameters.
- b) The following C0 control characters: NUL (0/0), BS (0/8), HT (0/9), LF (0/10), VT (0/11), FF (0/12), CR (0/13), SO (0/14), SI (0/15) and ESC (1/11). The NUL control is permitted, but has no effect on the meaning of the string. The bit combinations 0/8 to 0/13 are permitted, but have no standardized effect. The SO, SI, and ESC controls are permitted within conforming metafiles, dependent on the CHARACTER CODING ANNOUNCER. If the bit combinations 0/14 or 0/15 are present, they shall have the meanings (SO or LS1, and SI or LS0) specified for them in ISO 2022, unless a private coding technique has been selected by CHARACTER CODING ANNOUNCER. Other C0 controls are reserved for future standardization.
- c) The ESCAPE control (bit combination 1/11 with the exception of the bit combinations 1/11 5/8 and 1/11 5/12 for SOS and ST), and escape sequences formed according to the rules specified in ISO 2022. Those escape sequences for which meanings are specified in ISO 2022 shall have those specified meanings. ESCAPE sequences are only permitted if the CHARACTER CODING

String parameters

Method of encoding parameters

ANNOUNCER has selected a 2022-compatible technique. (If a private value of CHARACTER CODING ANNOUNCER is selected, ESCAPE may be used for escape sequences in a manner agreed upon by the interchanging parties. However this is not recommended as it will decrease the portability of the metafile.)

- d) By agreement between interchanging parties, C1 controls, with the exception of the bit combinations SOS and ST for the string delimiter. (In an 8-bit environment, these are bit combinations from columns 8 and 9 of the code chart.) In a 7-bit environment, they are ESC Fe sequences, where ESC is the bit combination 1/11 and Fe is a bit combination from column 4 or 5 of the code chart. The bit combinations 08/14 and 08/15 (or ESC 4/14 and ESC 4/15), if present, shall have the meanings SS2 (SINGLE SHIFT TWO) and SS3 (SINGLE SHIFT THREE) as defined in ISO 2022 when the extended 7-bit or extended 8-bit mode has been selected using CHARACTER CODING ANNOUNCER.

The bit combinations 0/1 through 0/7, 1/0 through 1/10, and 1/12 through 1/15 are reserved for future standardization.

6.9.3 C0 control within string parameters

Table 3 — C0 control set.

Column 0			Column 1		
0/0	NUL	(NULL-ignored)	1/0	DLE	(reserved)
0/1	SOH	(reserved)	1/1	DC1	(reserved)
0/2	STX	(reserved)	1/2	DC2	(reserved)
0/3	ETX	(reserved)	1/3	DC3	(reserved)
0/4	EOT	(reserved)	1/4	DC4	(reserved)
0/5	ENQ	(reserved)	1/5	NAK	(reserved)
0/6	ACK	(reserved)	1/6	SYN	(reserved)
0/7	BEL	(reserved)	1/7	ETB	(reserved)
0/8	BS	(no standardized effect)	1/8	CAN	(reserved)
0/9	HT	(no standardized effect)	1/9	EM	(reserved)
0/10	LF	(no standardized effect)	1/10	SUB	(reserved)
0/11	VT	(no standardized effect)	1/11	ESC	(ESCAPE)
0/12	FF	(no standardized effect)	1/12	IS4	(reserved)
0/13	CR	(no standardized effect)	1/13	IS3	(reserved)
0/14	SO	(SHIFT OUT)	1/14	IS2	(reserved)
0/15	SI	(SHIFT IN)	1/15	IS1	(reserved)

6.9.4 Using G-sets in string parameters

The G-sets (G0, G1, G2, and G3) are coded character sets of 94 or 96 bit combinations. They may be invoked into columns 2 through 7 of a 7-bit code chart, or columns 02 through 07 and 10 through 15 of an 8-bit code chart. This encoding of the CGM uses G-sets within string parameters, as follows:

6.9.4.1 String parameters of TEXT, APPEND TEXT, and RESTRICTED TEXT

For the string parameters of TEXT, APPEND TEXT, and RESTRICTED TEXT, the CHARACTER SET INDEX element designates a particular character set (from the list established by CHARACTER SET LIST) as the G0 set. Likewise, ALTERNATE CHARACTER SET INDEX designates a particular character set as both the G1 and G2 sets. These attributes may also apply to string parameters within data records of GDP elements.

At the start of each picture, BEGIN PICTURE invokes the G0 set into columns 2 through 7 of a 7-bit or 8-bit code chart. In an 8-bit environment, BEGIN PICTURE invokes the G1 set into columns 10 through 15 of the 8-bit code chart.

Method of encoding parameters**String parameters**

The ISO 2022 C0 or C1 controls or escape sequences to designate character sets as G0, G1, G2, or G3, or to invoke G-sets into the 7-bit or 8-bit code chart ("in-use table") may be included within the string parameters of the graphical primitive elements identified in 6.9.4.1 (with the exception of the string delimiter, listed in 6.9.1), if the CHARACTER CODING ANNOUNCER has selected an extended 2022 technique, whereupon they assume the meanings specified in ISO 2022.

6.9.4.2 String parameters of other metafile elements

The CHARACTER SET INDEX and ALTERNATE CHARACTER SET INDEX elements apply only to the string parameters of the graphical primitive elements identified in 6.9.4.1. String parameters of other metafile elements are not part of the pictures stored in the metafile. ISO 8632/2 does not specify the effect, if any, of using characters from code chart columns 10 through 15 in the string parameters of metafile elements other than the graphical primitive elements identified in 6.9.4.1, nor will any future revision of ISO 8632/2. ISO 8632/2 does not specify the effect of ISO 2022 designating and invoking controls when they occur within the string parameters of metafile elements other than the graphical primitive elements identified in 6.9.4.1.

6.10 Enumerated parameters

Enumerated parameters represent choices within a fixed set of standardized options. All of the enumerated types may have private values. They are coded as integers (Basic format). Private (non-standard) values of enumerated parameters shall use negative integers. Some but not all of these have been shown in the encodings in clause 8 below.

6.11 Index parameters

Indexes are coded as integers (Basic format), at INDEX PRECISION. Private (non-standard) values of index parameters are all coded using negative integers.

6.12 Data record parameters

Data record parameters are coded identically to string parameters in this encoding.

7 Character substitution

To accommodate systems in which it is inconvenient or impossible to include C0 control characters, the SPACE character (2/0), or the DELETE character (7/15) in the metafile, this CGM encoding includes a "character substitution" option. Characters in the range of 0/0 to 1/15, the characters 2/0, 7/14, and 7/15 may be replaced by 2-byte sequences provided each such 2-byte sequence is declared in the first parameter of the BEGIN METAFILE element.

Although the first parameter, 'substitution strings', of the BEGIN METAFILE element is of data type "string" the START OF STRING (SOS) introducer is omitted in this special case. This is because the SOS contains characters for which character substitution may be required. The STRING TERMINATOR (ST), however, is included in this parameter (character substitution, if selected, will already be in effect by the end of the string parameter).

Table 4 shows the characters that may be replaced by such 2-byte sequences, and the 2-byte sequences with which they are replaced. Note that all the 2-byte sequences begin with 7/14. Therefore, if the character substitution option is used at all, that character (7/14) shall be declared as one of the characters that is being replaced with a 2-byte sequence. This is done by including its replacement sequence (7/14 3/14) in the first parameter of BEGIN METAFILE.

Once a character (or rather, its 2-byte replacement) has been declared in the BEGIN METAFILE element, "character substitution" is in effect immediately for that character and will remain in effect until the end of the metafile (that is, until the END METAFILE element). The metafile interpreter would ignore any characters for which character substitution is in effect.

Example:

Supposing that it is desirable to avoid using the characters SPACE, TILDE, and DELETE in the metafile, and for the metafile interpreter to ignore those characters if they are inadvertently inserted (for example, by a host operating system or some process other than the metafile generator).

In that case, the metafile generator declares "character substitution" for the above two characters and for the TILDE character, 7/14. It does this in the first parameter of BEGIN METAFILE as follows:

```
BEGIN METAFILE
= 3/0 2/0 {BEGIN METAFILE opcode}

7/14 6/0 7/14 3/14 7/14 3/15 { string: ~">~?}
1/11 5/12 { STRING TERMINATOR (ST)}

1/11 5/8 { START OF STRING (SOS)}
metafile name
1/11 5/12 { ST}
```

Throughout the metafile, wherever the metafile generator would otherwise put a SPACE, TILDE (7/14), or DELETE character, it substitutes the 2-byte sequences 7/14 6/0, 7/14 3/14, or 7/14 3/15, respectively.

In the preceding example, wherever the metafile interpreter encounters the character 7/14, it interprets it as the first character of a 2-byte sequence representing one of these characters. The metafile interpreter would then ignore the characters 2/0 (SPACE), and 7/15 (DELETE).

Character substitution

Table 4 — Character substitution.

The character:		May be replaced with:		ISO 646 char.
		7-bit	8-bit	
0/0	(NUL)	7/14 4/0	07/14 04/0	(~@)
0/1	(SOH)	7/14 4/1	07/14 04/1	(~A)
0/2	(STX)	7/14 4/2	07/14 04/2	(~B)
0/3	(ETX)	7/14 4/3	07/14 04/3	(~C)
0/4	(EOT)	7/14 4/4	07/14 04/4	(~D)
0/5	(ENQ)	7/14 4/5	07/14 04/5	(~E)
0/6	(ACK)	7/14 4/6	07/14 04/6	(~F)
0/7	(BEL)	7/14 4/7	07/14 04/7	(~G)
0/8	(BS)	7/14 4/8	07/14 04/8	(~H)
0/9	(HT)	7/14 4/9	07/14 04/9	(~I)
0/10	(LF)	7/14 4/10	07/14 04/10	(~J)
0/11	(VT)	7/14 4/11	07/14 04/11	(~K)
0/12	(FF)	7/14 4/12	07/14 04/12	(~L)
0/13	(CR)	7/14 4/13	07/14 04/13	(~M)
0/14	(SO)	7/14 4/14	07/14 04/14	(~N)
0/15	(SI)	7/14 4/15	07/14 04/15	(~O)
1/0	(DLE)	7/14 5/0	07/14 05/0	(~P)
1/1	(DC1)	7/14 5/1	07/14 05/1	(~Q)
1/2	(DC2)	7/14 5/2	07/14 05/2	(~R)
1/3	(DC3)	7/14 5/3	07/14 05/3	(~S)
1/4	(DC4)	7/14 5/4	07/14 05/4	(~T)
1/5	(NAK)	7/14 5/5	07/14 05/5	(~U)
1/6	(SYN)	7/14 5/6	07/14 05/6	(~V)
1/7	(ETB)	7/14 5/7	07/14 05/7	(~W)
1/8	(CAN)	7/14 5/8	07/14 05/8	(~X)
1/9	(EM)	7/14 5/9	07/14 05/9	(~Y)
1/10	(SUB)	7/14 5/10	07/14 05/10	(~Z)
1/11	(ESC)	7/14 5/11	07/14 05/11	(~[)
1/12	(FS or IS4)	7/14 5/12	07/14 05/12	(~\)
1/13	(GS or IS3)	7/14 5/13	07/14 05/13	(~])
1/14	(RS or IS2)	7/14 5/14	07/14 05/14	(~^)
1/15	(US or IS1)	7/14 5/15	07/14 05/15	(~_)
2/0	(SPACE)	7/14 6/0	07/14 06/0	(~`)
7/14	(TILDE, ~)	7/14 3/14	07/14 03/14	(~>)
7/15	(DELETE)	7/14 3/15	07/14 03/15	(~?)

In a 7-bit or an 8-bit CGM coding environment the "GL" part of the code table (columns 02 through 07), table 4 may be summarized as follows: each character from decimal 0 (0/0) to decimal 32 (2/0) may be replaced by a 2-byte sequence in which the first byte is decimal 126 (7/14) and the decimal equivalent of the second byte is obtained by adding 64, modulo 128, to the decimal equivalent of the original character.

8 Representation of each element

For convenience, the 7-bit opcode is given for each element described in the remainder of this clause. To determine the 8-bit opcode, a zero is added in front of the column specification (for example, 02/1 instead of 2/1). A list of opcodes is given in table 1.

Notation Used:

<code><symbol>*</code>	=	0 or more occurrences
<code><symbol>+</code>	=	1 or more occurrences
<code><symbol>o</code>	=	optional, 0 or more occurrences
<code>{comment}</code>	=	explanation of a production
<code><x:y></code>	=	construct x with meaning y

STANDARDSISO.COM : Click to view the full PDF of ISO 8632-2:1987

Representation of each element

Delimiter elements

8.1 Delimiter elements

8.1.1 BEGIN METAFILE

<BEGIN-METAFILE-opcode: 3/0 2/0>
 <string: substitution-codes>
 <string: metafile-identifier>

The BEGIN METAFILE and END METAFILE elements respectively mark the beginning and the end of a metafile. (More than one metafile can be recorded in a single computer file provided each metafile is delimited by the elements BEGIN METAFILE and END METAFILE.)

The first parameter, <string: substitution-codes>, lists those 2-character substitution codes that will be used in the metafile to represent C0 control characters (bit combinations in the range of 0/0 to 1/15) or the SPACE, TILDE, and DELETE characters (bit combinations 2/0, 7/14, and 7/15, respectively). Character substitution will be in effect only for those bit combinations whose substitution codes are listed in the first <string> parameter. If the first <string> parameter is empty (represented by a single ST), character substitution is not to be used in this metafile.

NOTE - Substitution codes are not opened by an SOS introducer, see clause 7.

8.1.2 END METAFILE

<END-METAFILE-opcode: 3/0 2/1>

8.1.3 BEGIN PICTURE

<BEGIN-PICTURE-opcode: 3/0 2/2>
 <string: picture-identifier>

8.1.4 BEGIN PICTURE BODY

<BEGIN-PICTURE-BODY-opcode: 3/0 2/3>

8.1.5 END PICTURE

<END-PICTURE-opcode: 3/0 2/4>

8.2 Metafile descriptor elements

8.2.1 METAFILE VERSION

<METAFILE-VERSION-opcode: 3/1 2/0>
 <integer: version>

8.2.2 METAFILE DESCRIPTION

<METAFILE-DESCRIPTION-opcode: 3/1 2/1>
 <string: description-of-the-metafile>

8.2.3 VDC TYPE

<VDC-TYPE-opcode: 3/1 2/2>
 <enumerated: VDC-type>

<enumerated:VDC-type> = <integer: 0> {integer}
 | <integer: 1> {real}

8.2.4 INTEGER PRECISION

<INTEGER-PRECISION-opcode: 3/1 2/3>
 <integer: largest-integer-code + 1 >

The largest-integer-code tells how many bits occur in the largest possible magnitude for an integer. For example, if integers in the metafile can range from -32767 to +32767, the largest-integer-code is 15. One additional bit is required for the sign, and so is added to obtain the proper precision. Thus in this example the parameter would be 16.

8.2.5 REAL PRECISION

<REAL-PRECISION-opcode: 3/1 2/4>
 <integer: largest-real-code+1>
 <integer: smallest-real-code>
 <integer: default-exponent-for-reals>
 <enumerated: exponents-allowed>

<enumerated: exponents-allowed> = <integer: 0> {allowed}
 | <integer: 1> {forbidden}

The largest-real-code and smallest-real-code together specify the maximum number of bits of precision that can occur in the coded representations of real numbers. The largest-real-code specifies the largest possible magnitude of positive or negative real numbers, while the smallest-real-code specifies the granularity of the real number space.

The largest-real-code tells how many bits can occur to the left of the (binary) radix point in the largest possible real number. For example, if the largest possible real number is binary +1111111111111111111.111, the largest-real-code would be 20.

The smallest-real-code shows the granularity of the real number space by indicating how small a non-zero real number can be. It does this by specifying the smallest exponent that is permitted in

Representation of each element**Metafile descriptor elements**

the coded representation of a real number. For example, if all real numbers are integer multiples of $1/64$ (2 raised to the power -6), the smallest-real-code would be -6 . This indicates that there are six bits of precision to the right of the binary radix point and that the smallest allowable exponent is -6 .

It is possible for the smallest-real-code to be positive. For example, if the only real numbers to be coded are multiples of 8 , smallest-real-code would be $+3$, since 8 , or binary 1000 , is 2 raised to the power $+3$.

The largest-real-code shall be greater than the smallest-real-code. The difference, largest-real-code minus smallest-real-code, gives the number of bits of precision which are required in order to store real numbers with full precision.

The default-exponent-for-reals is the exponent that is to be assumed if an exponent is omitted from the coded representation of a real number. This parameter shall be greater than or equal to the smallest-real-code.

The exponents-allowed parameter specifies whether or not an "exponent follows" bit is included in the mantissa of the coded representation of each number. If this parameter is 0 , exponents are allowed in real parameters, and the exponent-follows bit is included in their mantissas. If this parameter is 1 , exponents are not allowed in real parameters and the mantissas of real parameters do not include the exponent-follows bit. In that case, the omitted exponents of all real parameters are deemed to have the value determined by the default-exponent-for-reals parameter.

8.2.6 INDEX PRECISION

<INDEX-PRECISION-opcode: 3/1 2/5>

<integer: largest-index-code + 1>

The largest-index-code tells how many bits occur in the largest possible magnitude for an index. For example, if indexes in the metafile can range from -511 to $+511$, the largest-integer-code is 9 . One additional bit is required for the sign, and so is added to obtain the proper precision. Thus in this example the parameter would be 16 .

8.2.7 COLOUR PRECISION

<COLOUR-PRECISION-opcode: 3/1 2/6>

<integer: largest-component code>

The largest-component-code tells how many bits occur in the largest possible magnitude for a direct colour component, i.e., for each of the red, green, and blue components. For example, if the largest-component-code is 10 , then direct colour components in the range $0-1023$ can be handled.

8.2.8 COLOUR INDEX PRECISION

<COLOUR-INDEX-PRECISION-opcode: 3/1 2/7>

<integer: largest-colour-index-code+1>

The parameter specifies the number of bits used in the binary numerals that represent colour indexes. For example, if the parameter is 4 , then the largest-colour-index-code is 3 , and there are 8 possible colour indexes: binary 0000 (decimal 0) to binary 0111 (decimal 7).

8.2.9 MAXIMUM COLOUR INDEX

<MAXIMUM-COLOUR-INDEX-opcode: 3/1 2/8>

<integer: maximum-colour-index>

8.2.10 COLOUR VALUE EXTENT

<COLOUR-VALUE-EXTENT-opcode: 3/1 2/9>
 <RGB: minimum-colour-value>
 <RGB: maximum-colour-value>

8.2.11 METAFILE ELEMENT LIST

<METAFILE-ELEMENT-LIST-opcode: 3/1 2/10>
 <string: element-list>

<string: element-list> = SOS <element names>* ST

<element names> = <element opcode>
 | <enumerated: element set>

<enumerated: element set> = <integer: 0> {DRAWING SET}
 | <integer: 1> {DRAWING PLUS CONTROL SET}
 | <integer: negative> {privately defined set}

8.2.12 METAFILE DEFAULTS REPLACEMENT

<BEGIN-METAFILE-DEFAULTS-opcode: 3/1 2/11>
 <element-that-sets-a-default>+
 <END-METAFILE-DEFAULTS-opcode: 3/1 2/12>

The <element-that-sets-a-default> list is given in part 1, clause 6. The coding format for each of the elements that appear in METAFILE DEFAULTS REPLACEMENT is exactly as described in this clause, Representation of Each Element. The parameter values express the default values to be used. For example, in a 7-bit encoding, the default values for CLIP INDICATOR and HATCH INDEX would be as follows:

3/1 2/11 {BEGIN-METAFILE-DEFAULTS-opcode}
 3/3 2/5 {CLIP-INDICATOR-opcode}
 integer: 1 {clip-indicator-on}
 3/6 3/3 {HATCH-INDEX-opcode}
 integer: 2 {default index equal to 2}
 3/1 2/12 {END-METAFILE-DEFAULTS-opcode}

8.2.13 FONT LIST

<FONT-LIST-opcode: 3/1 2/13>
 <font-declaration>+

<font-declaration> = <string: name-of-font>

The FONT LIST element declares the character fonts that may be named in subsequent TEXT FONT INDEX elements and establishes the font index value that is associated with each such character font.

The first <font-declaration> in the list names the character font whose font index value is to be 1. Likewise, the second, third, fourth, etc., <font-declaration>s name the fonts whose index values are to be 2, 3, 4, etc. The <font-declaration>s are separated from each other by the string delimiter characters.

8.2.14 CHARACTER SET LIST

<CHARACTER-SET-LIST-opcode: 3/1 2/14>
<character-set-declaration>+

```

<character-set-declaration>      =  <enumerated: character-set-type>
                                   <string: end-of-escape-sequence-to-designate-set>

```

```

<enumerated: character-set-type>      =  <integer: 0> {94-character G-set}
                                         <integer: 1> {96-character G-set}
                                         <integer: 2> {multibyte 94-character G-set}
                                         <integer: 3> {multibyte 96-character G-set}
                                         <integer: 4> {"complete code" character set}

```

The CHARACTER SET LIST element declares the character sets that can be named in subsequent CHARACTER SET INDEX and ALTERNATE CHARACTER SET INDEX elements and establishes the character set index value that is associated with each of these character sets.

The first `<character-set-declaration>` in the list names the character set whose character set index value is to be 1. Likewise, the second, third, fourth, etc., `<character-set-declaration>`s name the character sets whose index values are to be 2, 3, 4, etc.

Each `<character-set-declaration>` has two parts: an enumerated `<integer>` and a short `<string>`. The `<integer>` specifies which type of character set is being declared (that is, which type of ISO 2022 designating escape sequence is associated with that character set). The `<string>` consists of the character that forms the "tail end" of such designating escape sequences for that character set.

There are five types of character sets: 94-character G-sets, 96-character G-sets, 94-character multi-byte G-sets, 96-character multibyte G-sets, and character sets intended to be designated as "complete codes". Examples given below are for an 8-bit environment.

8.2.14.1 94-character G-sets

A CHARACTER SET LIST element could specify that the U.K. character set is to be referred to by character set index 1, and the French character set by character set index 2, as follows:

```
<CHARACTER-SET-LIST-opcode>
<integer: 0> SOS 4/1 ST
<integer: 0> SOS 6/6 ST
```

= 03/1 02/14 04/0 09/8 04/1 09/12 04/0 09/8 06/6 09/12

8.2.14.2 96-character G-sets

The following CHARACTER SET LIST element establishes the U.K. 94-character G-set, the French 94-character G-set, and a private 96-character G-set as the character sets named by character set indexes 1, 2, and 3, respectively:

```
<CHARACTER-SET-LIST-opcode>
<integer: 0> SOS 4/1 ST
<integer: 0> SOS 6/6 ST
<integer: 1> SOS 3/0 ST
```

$$= \begin{matrix} 03/1 & 02/14 & 04/0 & 09/8 & 04/1 & 09/12 \\ & 04/0 & 09/8 & 06/6 & 09/12 & 04/1 & 09/8 & 03/0 & 09/12 \end{matrix}$$

8.2.14.3 94-character multibyte G-sets

A Japanese 2-byte character set of 6802 graphic characters has been registered in the International Register Of Character Sets To Be Used With Escape Sequences, and its designating escape sequences have the form shown above, with the final character <F> being 4/0. Thus, the following

Metafile descriptor elements

Representation of each element

CHARACTER SET LIST element could be used to specify that this 2-byte Japanese character set is to be referred to by character set index 2:

```
<CHARACTER-SET-LIST-opcode>
<integer: 2> SOS 4/0 ST

= 03/1 02/14 04/2 09/8 04/0 09/12
```

8.2.14.4 96-character multibyte G-sets of graphic characters

It is not possible to designate a 96-character multibyte G-set as a G0 set.

The <character-set-declaration> for a 96-character multibyte G-set consists of <integer: 3> followed by a <string> consisting only of the final character <F> in the character set's ISO 2022 designating escape sequence.

So far, no 96-character multibyte G-sets have been registered in the International Register of Character Sets To Be Used with Escape Sequences.

8.2.14.5 Character sets intended to be designated as complete codes

The following CHARACTER SET LIST element would declare the French character set to have character set index 1 and that 8-bit private code to have character set index 2:

```
<CHARACTER-SET-LIST-opcode>
<integer: 0> SOS 6/6 ST
<integer: 4> SOS 2/0 3/0 ST

= 03/1 02/14 04/0 09/8 06/6 09/12 04/4 09/8 02/0 03/0 09/12
```

8.2.15 CHARACTER CODING ANNOUNCER

```
<CHARACTER-CODING-ANNOUNCER-opcode: 3/1 2/15>
<enumerated: coding-technique>
```

```
<enumerated: coding-technique> =
|   <integer: 0>           {basic 7-bit}
|   <integer: 1>           {basic 8-bit}
|   <integer: 2>           {extended 7-bit}
|   <integer: 3>           {extended 8-bit}
|   <integer: negative>    {private}
```

Representation of each element

Picture descriptor elements

8.3 Picture descriptor elements

8.3.1 SCALING MODE

<SCALING-MODE-opcode: 3/2 2/0>
 <enumerated: scaling-mode>
 <real: metric-scale-factor>

<enumerated: scaling-mode>	=	<integer: 0>	{abstract}
		<integer: 1>	{metric}

8.3.2 COLOUR SELECTION MODE

<COLOUR-SELECTION-MODE-opcode: 3/2 2/1>
 <enumerated: colour-selection-mode>

<enumerated: colour-selection-mode>	=	<integer: 0>	{indexed}
		<integer: 1>	{direct}

8.3.3 LINE WIDTH SPECIFICATION MODE

<LINE-WIDTH-SPECIFICATION-MODE-opcode: 3/2 2/2>
 <enumerated: specification-mode>

<enumerated: specification-mode>	=	<integer: 0>	{absolute}
		<integer: 1>	{scaled}

8.3.4 MARKER SIZE SPECIFICATION MODE

<MARKER-SIZE-SPECIFICATION-MODE-opcode: 3/2 2/3>
 <enumerated: specification-mode>

<enumerated: specification-mode>	=	<integer: 0>	{absolute}
		<integer: 1>	{scaled}

8.3.5 EDGE WIDTH SPECIFICATION MODE

<EDGE-WIDTH-SPECIFICATION-MODE-opcode: 3/2 2/4>
 <enumerated: specification-mode>

<enumerated: specification-mode>	=	<integer: 0>	{absolute}
		<integer: 1>	{scaled}

8.3.6 VDC EXTENT

<VDC-EXTENT-opcode: 3/2 2/5>
 <point: first-corner>
 <point: second-corner>

Picture descriptor elements

Representation of each element

8.3.7 BACKGROUND COLOUR

<BACKGROUND-COLOUR-opcode: 3/2 2/6>
<RGB: background colour>

STANDARDSISO.COM : Click to view the full PDF of ISO 8632-2:1987

8.4 Control elements

8.4.1 VDC INTEGER PRECISION

<VDC-INTEGER-PRECISION-opcode: 3/3 2/0>
 <integer: largest-integer-code+1>

The largest-integer-code tells how many bits occur in the largest possible magnitude for an integer point. For example, if integer points in the metafile can range from -32767 to +32767, the largest-integer-code is 15. One additional bit is required for the sign, thus an additional bit is added to obtain the appropriate field width. Thus in this example the parameter would be 16.

8.4.2 VDC REAL PRECISION

<VDC-REAL-PRECISION-opcode: 3/3 2/1>
 <integer: largest-real-code+1>
 <integer: smallest-real-code>
 <integer: default-exponent-for-vdc-reals>
 <enumerated: exponents-allowed>

<enumerated: exponents-allowed> = <integer: 0> {allowed}
 | <integer: 1> {forbidden}

The largest-real-code and smallest-real-code (or granularity code) together specify the maximum number of bits of precision that can occur in the coded representations of real VDC space coordinates and VDC scalars. The largest-real-code specifies the largest possible magnitude of positive or negative real VDC coordinates and VDC scalars, while the smallest-real-code specifies the granularity of VDC space.

The largest-real-code tells how many bits can occur to the left of the (binary) radix point in the largest possible VDC scalar. For example, if the largest possible real number is binary +1111111111111111111.111, the largest-real-code would be 20.

The smallest-real-code shows the granularity of VDC space by indicating how small a non-zero VDC scalar can be. It does this by specifying the smallest exponent that is permitted in the coded representation of a VDC scalar, thus determining the Basic Grid Unit: letting "src" stand for smallest-real-code, $BGU=2^{src}$. For example, if all VDC scalars are integer multiples of 1/64 (2 raised to the power -6), the smallest-real-code would be -6. This indicates that there are six bits of precision to the right of the binary radix point and that the smallest allowable exponent is -6.

It is possible for the smallest-real-code to be positive. For example, if all VDC scalars to be coded are multiples of 8, smallest-real-code would be +3, since 8, or binary 1000, is 2 raised to the power +3.

The largest-real-code shall be greater than the smallest-real-code. The difference, largest-real-code minus smallest-real-code, gives the number of bits of precision which are required in order to store VDC coordinates with full precision. It is especially important to use all these bits of precision when coding the point list parameters of the POLYLINE, DISJOINT POLYLINE, POLYMARKER, POLYGON, POLYGON SET, and GDP.

The default-exponent-for-vdc-reals is the exponent that is to be assumed if an exponent is omitted from the coded representation of a real VDC coordinate or VDC scalar. This parameter shall be greater than or equal to the smallest-real-code.

The exponents-allowed parameter specifies whether or not an "exponent follows" bit is included in the mantissa of the coded representation of a real VDC coordinate or VDC scalar. If this parameter is 0, exponents are allowed in real VDC coordinates and VDC scalars, and the exponent-follows bit is

Control elements

Representation of each element

included in their mantissas (see 6.5.) If this parameter is 1, exponents are not allowed in real VDC coordinates and VDC scalars; in that case the mantissas of real VDC coordinates and VDC scalar do not include the exponent-follows bit. In that case, the omitted exponents of all real parameters are deemed to have the value determined by the default-exponent-for-reals parameter.

8.4.3 AUXILIARY COLOUR

<AUXILIARY-COLOUR-opcode: 3/3 2/2>
<colour-specifier: auxiliary-colour>

<colour-specifier> = <integer: colour-index> {if COLOUR SELECTION MODE is indexed}
 | <RGB> {if COLOUR SELECTION MODE is direct}

8.4.4 TRANSPARENCY

<TRANSPARENCY-opcode: 3/3 2/3>
<enumerated: transparency indicator>

<enumerated: transparency indicator> =<integer: 0> {off}
 | <integer: 1> {on}

8.4.5 CLIP RECTANGLE

<CLIP-RECTANGLE-opcode: 3/3 2/4>
<point: first-corner>
<point: second-corner>

8.4.6 CLIP INDICATOR

<CLIP-INDICATOR-opcode: 3/3 2/5>
<enumerated: clip-indicator>

<enumerated: clip-indicator> = <integer: 0> {off}
 | <integer: 1> {on}

STANDARDSISO.COM : Click to view the full PDF of ISO 8632-2:1987

Representation of each element

Graphical primitive elements

8.5 Graphical primitive elements

8.5.1 POLYLINE

<POLYLINE-opcode: 2/0>
 <point-list>

<point-list> = <point><point>+

8.5.2 DISJOINT POLYLINE

<DISJOINT-POLYLINE-opcode: 2/1>
 <point-list>

8.5.3 POLYMARKER

<POLYMARKER-opcode: 2/2>
 <point-list>

8.5.4 TEXT

<TEXT-opcode: 2/3>
 <point: starting-point>
 <enumerated: final-or-not-final>
 <string: text-to-be-displayed>

<enumerated: final-or-not-final> = <integer: 0> {not final}
 | <integer: 1> {final}

The string parameter may contain characters from G sets together with the ISO 2022 designating and/or invoking controls to select those G-sets as described below. Other C0 and C1 controls (except as noted in 6.9.2) may be included by agreement between exchanging parties but have no standardized meaning.

Within the string parameter, characters from columns 2 through 7 are displayed using the GL character set selected by the most recent CHARACTER SET INDEX element, the default character set, or by the most recent locking shift. If the character set selected by CHARACTER SET INDEX (or by default) is a 94-character G-set, the bit combinations 2/0 and 7/15 represent SPACE and DELETE, respectively.

Characters from columns 10 through 15 are displayed using the GR character set selected by the most recent ALTERNATE CHARACTER SET INDEX or locking shift right.

The ISO 2022 controls may occur within the string parameter provided their use is declared with the CHARACTER CODING ANNOUNCER element.

8.5.5 RESTRICTED TEXT

<RESTRICTED-TEXT-opcode: 2/4>
 <VDC: delta-width>
 <VDC: delta-height>
 <point: starting-point>

Graphical primitive elements

Representation of each element

<enumerated: final-or-not-final>
 <string: text-to-be-displayed>

<enumerated: final-or-not-final> = <integer: 0> {not final}
 | <integer: 1> {final}

See 8.5.4, TEXT, for a description of the string parameter for RESTRICTED TEXT.

8.5.6 APPEND TEXT

<APPEND-TEXT-opcode: 2/5>
 <enumerated: final-or-not-final>
 <string: text-to-be-displayed>

<enumerated: final-or-not-final> = <integer: 0> {not final}
 | <integer: 1> {final}

See 8.5.4, TEXT, for a description of the parameters for APPEND TEXT.

8.5.7 POLYGON

<POLYGON-opcode: 2/6>
 <point-list>

8.5.8 POLYGON SET

<POLYGON-opcode: <2/7>
 <flagged point-list>

<flagged point-list> = <flagged-point>+

where the first coordinate is coded as an absolute and subsequent coordinates as incremental, as in the point list parameter type.

<flagged-point> = <coordinate> <enumerated: edge-out>

<enumerated: edge out> = <integer: 0> {invisible}
 | <integer: 1> {visible}
 | <integer: 2> {close invisible}
 | <integer: 3> {close visible}

8.5.9 CELL ARRAY

<CELL-ARRAY-opcode: 2/8>
 <P: point>
 <Q: point>
 <R: point>
 <integer: nx>
 <integer: ny>
 <local-colour-precision>
 <colour-list>

<local-colour-precision> = <default-colour-precision-indicator>

Representation of each element

Graphical primitive elements

| <number-of-bits>

<default-colour-precision-indicator> = <integer: 0>
 <number-of-bits> = <integer: positive>

The value 0 for the local-colour-precision is the 'default colour precision indicator'. It indicates that the colour specifications of the CELL ARRAY element are in the precision of the COLOUR INDEX PRECISION or the COLOUR PRECISION (depending upon the value of the COLOUR SELECTION MODE). If positive, then it is the 'number of bits' of the colour specifications of the CELL ARRAY element (as in COLOUR PRECISION and COLOUR INDEX PRECISION).

<colour-list> = <indexed colour list>
 | <direct colour list>

<indexed colour list> = <indexed normal colour list>
 | <indexed bitstream colour list>
 | <indexed runlength colour list>
 | <indexed runlength bitstream colour list>

<direct colour list> = <direct normal colour list>
 | <direct bitstream colour list>
 | <direct runlength colour list>
 | <direct runlength bitstream colour list>

<indexed normal colour list> = <integer: 0>
 <basic format>+

<indexed bitstream colour list> = <integer: 1>
 <bitstream: concatenation of colour indexes>

<indexed runlength colour list> = <integer: 2>
 <indexed run>+

<indexed run> = <integer: colour index>
 <integer: number of repetitions>

<indexed runlength bitstream colour list> = <integer: 3>
 <indexed bitstream run>+

<indexed bitstream run> = <bitstream: colour index>
 <integer: number of repetitions>

<direct normal colour list> = <integer: 0>
 <RGB: colour value>+

<direct bitstream colour list> = <integer: 1>
 <bitstream: concatenation of colour values>

<direct runlength colour list> = <integer: 2>
 <direct run>+

<direct run> = <RGB: colour value>
 <integer: number of repetitions>

<direct runlength bitstream colour list> = <integer: 3>

<direct run>+

8.5.10 GENERALIZED DRAWING PRIMITIVE

<GDP-opcode: 2/9>
 <integer: GDP-identifier>
 <point-list>
 <data-record: data-record>

8.5.11 RECTANGLE

<RECTANGLE-opcode: 2/10>
 <point: first-corner>
 <point: second-corner>

8.5.12 CIRCLE

<CIRCLE-opcode: 3/4 2/0>
 <point: centre-of-circle>
 <VDC: radius-of-circle>

8.5.13 CIRCULAR ARC 3 POINT

<CIRCULAR-ARC-3-POINT-opcode: 3/4 2/1>
 <point: starting-point>
 <point: intermediate-point>
 <point: ending-point>

8.5.14 CIRCULAR ARC 3 POINT CLOSE

<CIRCULAR-ARC-3-POINT-CLOSE-opcode: 3/4 2/2>
 <point: starting-point>
 <point: intermediate-point>
 <point: ending-point>
 <enumerated: close-type>

<enumerated: close-type>	=	<integer: 0>	{pie}
		<integer: 1>	{chord}

8.5.15 CIRCULAR ARC CENTRE

<CIRCULAR-ARC-CENTRE-opcode: 3/4 2/3>
 <point: centrepoint>
 <VDC: DX_start>
 <VDC: DY_start>
 <VDC: DX_end>
 <VDC: DY_end>
 <VDC: radius>

Representation of each element

Graphical primitive elements

8.5.16 CIRCULAR ARC CENTRE CLOSE

```

<CIRCULAR-ARC-CENTRE-CLOSE-opcode: 3/4 2/4>
<point: centrepoint>
<VDC: DX_start>
<VDC: DY_start>
<VDC: DX_end>
<VDC: DY_end>
<VDC: radius>
<enumerated: close-type>

<enumerated: close-type>      =  <integer: 0>  {pie}
                                |  <integer: 1>  {chord}

```

8.5.17 ELLIPSE

```

<ELLIPSE-opcode: 3/4 2/5>
<point: centrepoint>
<point: endpoint of first conjugate diameter>
<point: endpoint of second conjugate diameter>

```

8.5.18 ELLIPTICAL ARC

```

<ELLIPTICAL-ARC-opcode: 3/4 2/6>
<point: centrepoint>
<point: endpoint of first conjugate diameter>
<point: endpoint of second conjugate diameter>
<VDC: DX_start>
<VDC: DY_start>
<VDC: DX_end>
<VDC: DY_end>

```

8.5.19 ELLIPTICAL ARC CLOSE

```

<ELLIPTICAL-ARC-CLOSE-opcode: 3/4 2/7>
<point: centrepoint>
<point: endpoint of first conjugate diameter>
<point: endpoint of second conjugate diameter>
<VDC: DX_start>
<VDC: DY_start>
<VDC: DX_end>
<VDC: DY_end>
<enumerated: close-type>

<enumerated: close-type>      =  <integer: 0>  {pie}
                                |  <integer: 1>  {chord}

```

8.6 Attribute elements

8.6.1 LINE BUNDLE INDEX

<LINE-BUNDLE-INDEX-opcode: 3/5 2/0>
<integer: line-bundle-index>

<integer: line-bundle-index> = <positive integer>

8.6.2 LINE TYPE

<LINE-TYPE-opcode: 3/5 2/1>
<index: line-type>

<index: line-type>	=	<integer: 1>	{solid}
		<integer: 2>	{dash}
		<integer: 3>	{dot}
		<integer: 4>	{dash-dot}
		<integer: 5>	{dash-dot-dot}
		<integer: negative>	{private line type}

8.6.3 LINE WIDTH

<LINE-WIDTH-opcode: 3/5 2/2>
<line-width-specifier>

<line-width-specifier>	=	<real: line width scale factor>
		{if LINE WIDTH SPECIFICATION MODE is scaled}
		<VDC: line width>
		{if LINE WIDTH SPECIFICATION MODE is absolute}

8.6.4 LINE COLOUR

<LINE-COLOUR-opcode: 3/5 2/3>
<colour-specifier>

<colour-specifier>	=	<integer: colour-index>	{if COLOUR SELECTION MODE is indexed}
		<RGB>	{if COLOUR SELECTION MODE is direct}

<integer: colour-index> = <non-negative integer>

8.6.5 MARKER BUNDLE INDEX

<MARKER-BUNDLE-INDEX-opcode: 3/5 2/4>
<integer: marker-bundle-index>

<integer: marker-bundle-index> = <positive integer>

Representation of each element

Attribute elements

8.6.6 MARKER TYPE

<MARKER-TYPE-opcode: 3/5 2/5>
 <index: marker-type>

<index: marker-type>	=	<integer: 1>	{dot}
		<integer: 2>	{plus}
		<integer: 3>	{asterisk}
		<integer: 4>	{circle}
		<integer: 5>	{cross (x)}
		<integer: negative>	{private marker type}

8.6.7 MARKER SIZE

<MARKER-SIZE-opcode: 3/5 2/6>
 <marker-size-specifier>

<marker-size-specifier>	=	<real: marker size scale factor>	{if MARKER SIZE SPECIFICATION MODE is 'scaled'}
		<VDC: marker size>	{if MARKER SIZE SPECIFICATION MODE is 'absolute'}

8.6.8 MARKER COLOUR

<MARKER-COLOUR-opcode: 3/5 2/7>
 <colour-specifier: marker-colour>

<colour-specifier>	=	<integer: colour-index>	{if COLOUR SELECTION MODE is 'indexed'}
		<RGB>	{if COLOUR SELECTION MODE is 'direct'}

<integer: colour-index>	=	<non-negative integer>
-------------------------	---	------------------------

8.6.9 TEXT BUNDLE INDEX

<TEXT-BUNDLE-INDEX-opcode: 3/5 3/0>
 <integer: text-bundle-index>

<integer: text-bundle-index>	=	<positive integer>
------------------------------	---	--------------------

8.6.10 TEXT FONT INDEX

<TEXT-FONT-INDEX-opcode: 3/5 3/1>
 <integer: font-index>

<integer: text-font-index>	=	<positive integer>
----------------------------	---	--------------------

8.6.11 TEXT PRECISION

<TEXT-PRECISION-opcode: 3/5 3/2>
 <enumerated: TEXT-PRECISION>

Attribute elements

Representation of each element

<enumerated: text-precision>

=

<integer: 0> {string}

|

<integer: 1> {character}

|

<integer: 2> {stroke}

8.6.12 CHARACTER EXPANSION FACTOR

<CHARACTER-EXPANSION-FACTOR-opcode: 3/5 3/3>

<real: expansion-factor>

<real: expansion-factor>

=

<non-negative real>

8.6.13 CHARACTER SPACING

<CHARACTER-SPACING-opcode: 3/5 3/4>

<real: character-spacing>

8.6.14 TEXT COLOUR

<TEXT-COLOUR-opcode: 3/5 3/5>

<colour-specifier>

<colour-specifier>

=

<integer: colour-index> {if COLOUR SELECTION MODE is 'indexed'}

|

<RGB> {if COLOUR SELECTION MODE is 'direct'}

<integer: colour-index>

=

<non-negative integer>

8.6.15 CHARACTER HEIGHT

<CHARACTER-HEIGHT-opcode: 3/5 3/6>

<VDC: character-height>

<VDC: character-height>

=

<non-negative VDC>

8.6.16 CHARACTER ORIENTATION

<CHARACTER-ORIENTATION-opcode: 3/5 3/7>

<VDC: x-component of up vector>

<VDC: y-component of up vector>

<VDC: x-component of base vector>

<VDC: y-component of base vector>

8.6.17 TEXT PATH

<TEXT-PATH-opcode: 3/5 3/8>

<enumerated: text-path>

<enumerated: text-path>

=

<integer: 0> {right}

|

<integer: 1> {left}

|

<integer: 2> {up}