



© British Standards Institution. No part of this publication may be photocopied or otherwise reproduced without the prior permission in writing of BSI

Draft for Development

Computer graphics: Metafile for the storage and transfer of picture description information

Part 3. Binary encoding (ISO/DIS 8632/3)

Projet à développer
Infographie
Métafichier graphique pour la communication et l'archivage d'information graphique
Partie 3. Codage binaire

Vornorm
Graphische Datenverarbeitung
Metadatei für die Speicherung und Übertragung von Informationen zur Bildbeschreibung
Teil 3. Binärcodierung

Contents

	Page		Page
National foreword	2	7.9 External elements	41
Committees responsible	Back cover	8 Defaults	42
		9 Conformance	43
Draft for Development		Annexes	
0 Introduction	3	A Formal grammar	45
0.1 Purpose of the binary encoding	3	B Examples	47
0.2 Objectives	3	B.1 Example 1: BEGIN METAFILE 'Example 1'	47
0.3 Relationship to other standards	4	B.2 Example 2: BEGIN PICTURE 'Test'	47
0.4 Status of annexes	4	B.3 Example 3: POLYLINE from 0,2 to 1,3 to 2,1 to 0,2	48
1 Scope and field of application	5	B.4 Example 4: TEXT 'Hydrogen' at 0,1	48
1.1 Scope	5	B.5 Example 5: Partitioned POLYLINE with 50 points	49
1.2 Field of application	5	B.6 Example 6: METAFILE DEFAULT REPLACEMENT linewidth 0.5	50
2 References	6	B.7 Example 7: Application Data # 655 with 10K octets (chars) of data	50
3 Notational conventions	7	C List of binary encoding metafile element codes	51
4 Overall structure	8		
4.1 General form of metafile	8	Tables	
4.2 General form of pictures	8	1 Representation of abstract data types	18
4.3 General structure of the binary metafile	8	2 List of element class codes	21
4.4 Structure of the command header	9	3 Encoding of delimiter elements	22
5 Primitive data forms	12	4 Encoding of metafile descriptor elements	23
5.1 Signed integer	12	5 Encoding of picture descriptor element	26
5.2 Unsigned integer	13	6 Encoding of control elements	28
5.3 Character	14	7 Encoding of graphical primitive elements	30
5.4 Fixed point real	14	8 Encoding of attribute elements	34
5.5 Floating point	15	9 Encoding of escape element	40
6 Representations of abstract parameter types	17	10 Encoding of external elements	41
7 Representation of each element	21	Figures	
7.1 Method of presentation	21	1 Format of a short-form command header	10
7.2 Delimiter elements	22	2 Format of a long-form command header	10
7.3 Metafile descriptor elements	23		
7.4 Picture descriptor element	26		
7.5 Control elements	28		
7.6 Graphical primitive elements	30		
7.7 Attribute elements	34		
7.8 Escape element	40		

This publication is not to be regarded as a British Standard.

It is being issued in the Draft for Development series of publications and is of a provisional nature because the subject matter is in the process of ongoing development within the International Standard Committee (ISO/TC 97/ SC 21 Information Retrieval, Transfer and Management for Open Systems Interconnection).

It should be applied on this provisional basis, so that information and experience of its practical application may be obtained.

A review of this Draft for Development will be carried out not later than two years after its publication. Notification

of the start of the review period, with a request for the submission of comments from users of this Draft for Development will be made in an announcement in the appropriate issue of *BSI News*. Observations which it is felt should receive attention before the official call for comments will be welcomed.

According to the replies received, the responsible BSI Committee will judge whether the Draft for Development can be converted into a British Standard or what other action should be taken.

National foreword

This Part of DD 107 has been prepared under the direction of the Office and Information Standards Committee.

The primary objective of this Part of DD 107 is to give advance guidance to product and systems designers concerned with computer graphics.

The text of this Part of DD 107 is identical with Draft International Standard ISO/DIS 8632/3-1985, issued by the International Organization for Standardization (ISO).

For ease of reproduction, the ISO text has been reproduced unchanged.

The UK is actively participating through BSI Technical Committee IST/21* in the development of the ISO International Standard. The Technical Committee will review any agreed amendment or addenda to the ISO text and will, where appropriate, update the text of this Draft for Development by means of an amendment or revision. When the ISO draft reaches an appropriate level of agreement and stability, this Part of DD 107 may then be converted to a British Standard.

The four Parts of ISO/DIS 8632 are being published as corresponding Parts of DD 107, as follows.

ISO/DIS 8632/1-1985 is identical with
DD 107 : Part 1 Functional specification

ISO/DIS 8632/2-1985 is identical with
DD 107 : Part 2 Character encoding

ISO/DIS 8632/3-1985 is identical with
DD 107 : Part 3 Binary encoding

ISO/DIS 8632/4-1985 is identical with
DD 107 : Part 4 Clear text encoding

All four Parts are currently at the Draft International Standard stage.

A brochure giving details of existing and projected British Standards publications for Open Systems Interconnection

(OSI) may be obtained from the Sales Department, BSI Linford Wood, Milton Keynes, MK14 6LE, at no charge.

Press releases for new OSI, data communications and computer graphics British Standards publications will be issued to the computing press, and details of all new publications will be recorded in full in *BSI News*.

Terminology and conventions. The text of the Draft International Standard has been approved as suitable for publication as a Draft for Development without deviation. Some terminology and certain conventions are not identical with those used in Drafts for Development; attention is drawn especially to the following.

Wherever the word 'Standard' appears, referring to this Draft for Development, it should be read as 'Draft for Development'.

Wherever page numbers are quoted, they are ISO page numbers.

Cross-references

International Standard	Corresponding British Standard
ISO 646-1983	BS 4730 : 1985 Specification for UK 7 bit data code (Identical, exercising national options)
ISO 7942-1985	BS 6390 : 1985 Specification for a set of functions for computer graphics programming, the Graphical Kernel System (GKS) (Identical)

ISO 2022-1982, to which reference is made in the text, is currently being revised, and this revision may be implemented as a British Standard. BS 4953, which was technically equivalent to ISO 2022-1973, was withdrawn in June 1984.

*Previously OIS/121.

0 Introduction

0.1 Purpose of the Binary Encoding

The Binary Encoding of the Computer Graphics Metafile (CGM) provides a representation of the Metafile syntax that can be optimized for speed of generation and interpretation, while still providing a standard means of interchange among computer systems. The encoding uses binary data formats that are much more similar to the data representations used within computer systems than the data formats of the other encodings.

Some of the data formats may exactly match those of some computer systems. In such cases processing is reduced very much relative to the other standardized encodings. On most computer systems processing requirements for the Binary Encoding will be substantially lower than for the other encodings.

In cases where a computer system's architecture does not match the standard formats used in the Binary Encoding, and where absolute minimization of processing requirements is critical, and where interchange among dissimilar systems does not matter, it may be more appropriate to use a private encoding, conforming to the rules specified in clause 7 of part 1 of this Standard.

0.2 Objectives

This encoding has the following features.

- a. Partitioning of parameter lists: metafile elements are coded in the Binary Encoding by one or more partitions (see clause 4); the first (or only) partition of an element contains the opcode (Element Class plus Element Id).
- b. Alignment of elements: every element begins on a word boundary. Alignment of partitions that require an odd number of octets is effected by padding with an octet with all bits zero. A no-op element is available in this encoding; it is ignored. It may be used to align data on machine-dependent record boundaries for speed of processing.
- c. Uniformity of format: all elements have an associated parameter length value. The length is specified as an octet count. As a result, it is possible to scan the metafile, without interpreting it, at high speed.
- d. Alignment of coordinate data: at default precisions and by virtue of alignment of elements, coordinate data always start on word boundaries. This minimizes processing by ensuring, on a wide class of computing systems, that single coordinates do not have to be assembled from pieces of multiple computer words.
- e. Efficiency of encoding integer data: other data such as indexes, colour and characters are encoded as one or more octets. The precision of every parameter is determined by the appropriate precision as given in the METAFILE DESCRIPTOR.
- f. Order of bit data: in each word, or unit within a word, the bit with the highest number is the most significant bit. Likewise, when data word are accessed sequentially, the least significant word follows the most significant.
- g. Extensibility: the arrangement of Element Class and Element Id values has been designed to allow future growth, such as new graphical elements.
- h. Format of real data: real numbers are encoded using either IEEE floating point representation or a metafile fixed-point representation.

Objectives**Introduction**

- i. Run length encoding: if many adjacent cells have the same colour (or colour index) efficient encoding is possible. For each run, the colour (or colour index) is specified, followed by a cell count.
- j. Packed list encoding: if adjacent colour cells do not have the same colour (or colour index) the metafile provides bit-stream lists in which the values are packed as closely as possible.

0.3 Relationship to Other Standards

The floating point representation of real data in this part of the Standard is that in IEEE standard P754.

The representation of character data in this part of the Standard 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 part 1, 4.11.)

0.4 Status of Annexes

The annexes do not form an integral part of the Standard but are included to provide extra information and explanation.

[ISO page 2]

This publication is a Draft for Development and is therefore not to be regarded as a British Standard. See page 1.

1 Scope and Field of Application

1.1 Scope

This part of this Standard specifies a binary encoding of the Computer Graphics Metafile. For each of the elements specified in part 1 of this Standard, an encoding is specified in terms of a data type. For each of these data types, an explicit representation in terms of bits, octets and words is specified. For some data types, the exact representation is a function of the precisions being used in the metafile, as recorded in the METAFILE DESCRIPTOR.

1.2 Field of Application

This encoding of the Computer Graphics Metafile will, in many circumstances, minimize the effort required to generate and interpret the metafile.

2 References

The publications referred to in this part of the Standard are as follows.

- ISO 646 Information Processing - 7-bit coded character set for information interchange
- ISO 2022 Information Processing - ISO 7-bit and 8-bit coded character sets - Code Extension techniques
- IEEE P754 Standard for Binary Floating Point Arithmetic

[ISO page 4]

This publication is a Draft for Development and is therefore not to be regarded as a British Standard. See page 1

Documents

Document Sharing Hub

3 Notational Conventions

"Command Header" is used throughout this part to refer to that portion of a Binary-Encoded element that contains the opcode (element class plus element id) and parameter length information (see clause 4).

Within this part, the terms "octet" and "word" have specific meanings. These meanings may not match those of a particular computer system on which this encoding of the metafile is used.

An octet is an 8-bit entity. All bits are significant. The bits are numbered from 7 (most significant) to 0 (least significant).

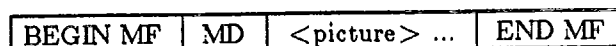
A word is a 16-bit entity. All bits are significant. The bits are numbered from 15 (most significant) to 0 (least significant).

4 Overall Structure

4.1 General Form of Metafile

All elements in the metafile are encoded using a uniform scheme. The elements are represented as variable length data structures, each consisting of opcode information (element class plus element id) designating the particular element, the length of its parameter data and finally the parameter data (if any).

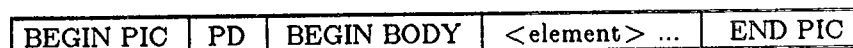
The structure of the metafile is as follows. (For the purposes of this diagram only, MF is used as an abbreviation for METAFILE.)



The BEGIN METAFILE element is followed by the METAFILE DESCRIPTOR (MD). After this the pictures follow, each logically independent of each other. Finally the Metafile is ended with an END METAFILE element.

4.2 General Form of Pictures

Apart from the BEGIN METAFILE, END METAFILE and Metafile Descriptor elements, the metafile is partitioned into pictures. All pictures are mutually independent. A picture consists of a BEGIN PICTURE element, a PICTURE DESCRIPTOR (PD) element, a BEGIN PICTURE BODY element, an arbitrary number of control, graphical and attribute elements and finally an END PICTURE element. (For the purpose of this diagram only, PIC is used as an abbreviation for PICTURE and BEGIN BODY for BEGIN PICTURE BODY.)



4.3 General Structure of the Binary Metafile

The binary encoding of the metafile is a logical data structure consisting of a sequential collection of bits. For convenience in describing the length and alignment of metafile elements, fields of two different sizes are defined within the structure. These fields are used in the remainder of this part of the Standard for illustrating the contents and structure of elements and parameters.

For measuring the lengths of elements the metafile is partitioned into octets, which are 8-bit fields.

The structure is also partitioned into 16-bit fields called words (these are logical metafile words). To optimize processing of the binary metafile on a wide collection of computers, metafile elements are constrained to start on word boundaries within the binary data structure (this alignment may necessitate padding an element with bits to a word boundary if the parameter data of the element does not fill to such a boundary).

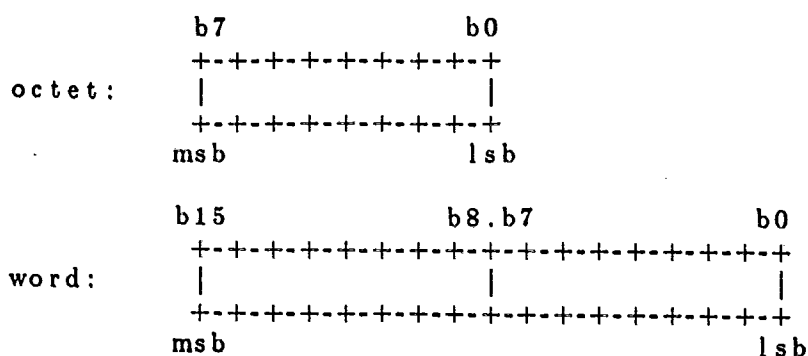
The octet is the fundamental unit of organization of the binary metafile.

The bits of an octet are numbered 7 to 0, with 7 being the most significant bit. The bits of a word are numbered 15 to 0, with 15 being the most significant bit.

[ISO page 6]

This publication is a Draft for Development and is therefore not to be regarded as a British Standard. See page 1.

General Structure of the Binary Metafile



metafile bit number	octet bit number	word bit number
1	b7/octet1	b15/word1
.	.	.
.	.	.
8	b0/octet1	b8/word1
9	b7/octet2	b7/word1
.	.	.
.	.	.
16	b0/octet2	b0/word1
17	b7/octet3	b15/word2
.	.	.
.	.	.
24	b0/octet3	b8/word2
25	b7/octet4	b7/word2
.	.	.
.	.	.

The forms differ in the format of the Command Header that precedes the parameter list. The command form for an element (short or long) is established by the first word of the element. For the short-form, the Command Header consists of a single word divided into three fields: element class, element id and parameter list length.

Structure of the Command Header

Overall Structure

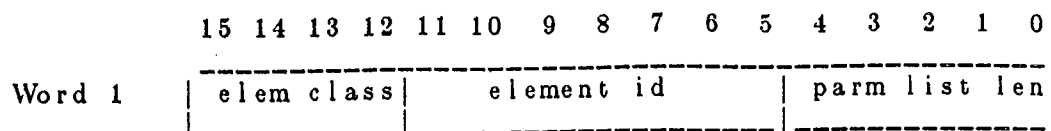


Figure 1. Format of a short-form Command Header.

The fields in the short-form Command Header are as follows.

- bits 15-12 element class (value range 0-15)
- bits 11-5 element id (value range 0-127)
- bits 4-0 parameter list length: the number of octets of parameter data that follow for this command (value range 0-30)

This Command Header is then followed by the parameter list.

The first word of a long-form command is identical in structure to the first word of a short-form command. The presence of the value 11111 binary (decimal 31) in parameter list length field indicates that the command is a long-form command. The Command Header for the long-form command consists of two words. The second word contains the actual parameter list length. The two header words are then followed by the parameter list.

In addition to allowing longer parameter lists, the long-form command allows the parameter list to be partitioned. Bit 15 of the second word indicates whether the given data complete the element or more data follow. For subsequent data partitions of the element, the first word of the long-form Command Header (containing element class and element id) is omitted; only the second word, containing the parameter list length, is given. The parameter list length for each partition specifies the length of that partition, not the length of the complete element. The final partition of an element is indicated by bit 15 of the parameter list length word being zero.

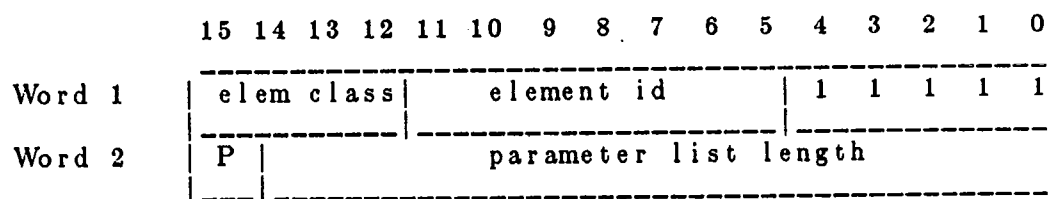


Figure 2. Format of a long-form Command Header.

The fields in the long-form Command Header are as follows.

Word 1

- bits 15-12 element class (value range 0-15)
- bits 11-5 element id (value range 0-127)
- bits 4-0 binary value 11111 (decimal 31) indicating long-form

Word 2

- bit 15 partition flag
 - 0 for 'last' partition
 - 1 for 'not-last' partition
- bits 14-0 parameter list length: the number of octets of parameter data that follow for this command or partition (value range 0-32767).

The parameter values follow the parameter list length for either the long-form or short-form commands. The number of values is determined from the parameter list length and the type and precision of the

Overall Structure

Structure of the Command Header

operands. These parameter values have the format illustrated in clause 5 of this part. The parameter type for coordinates is indicated in the Metafile Descriptor. For non-coordinate parameters, the parameter type is as specified in clause 5 of part 1. If the parameter type is encoding dependent, its code is specified in the coding tables of clause 7 of this part. Unless otherwise stated, the order of parameters is as listed in clause 5 of part 1.

Every command is constrained to begin on a word boundary. This necessitates padding the command with a single null octet at the end of the command if the command contains an odd number of octets of parameter data. In addition, in elements with parameters whose precisions are shorter than one octet (i.e., those containing a 'local colour precision' parameter) it is necessary to pad the last data-containing octet with null bits if the data do not fill the octet. In all cases, the parameter list length is the count of octets actually containing parameter data -- it does not include the padding octet if one is present. It is only at the end of a command that padding is performed, with the single exception of the CELL ARRAY element.

The purpose of this command alignment constraint is to optimize processing on a wide class of computers. At the default metafile precisions, the parameters which are expected to occur in greatest numbers (coordinates, etc) will align on 16-bit boundaries, and Command Headers will align on 16-bit boundaries. Thus, at the default precisions the most frequently parsed entities will lie entirely within machine words in a large number of computer designs. The avoidance of assembling single metafile parameters from pieces of several computer words will approximately halve the amount of processing required to recover element parameters and command header fields from a binary metafile data stream.

This optimization may be compromised or destroyed altogether if the metafile precisions are changed from default. Commands are still constrained to begin on 16-bit boundaries, but the most frequently expected parameters may no longer align on such boundaries as they do at the default precisions.

[ISO page 9]

5 Primitive Data Forms

The Binary Encoding of the CGM uses five primitive data forms to represent the various abstract data types used to describe parameters in the part 1 of this Standard.

The primitive data forms and the symbols used to represent them are as follows.

SI	Signed Integer
UI	Unsigned Integer
C	Character
FX	Fixed Point Real
FP	Floating Point Real

Each of these primitive forms (except Character) can be used in a number of precisions. The definitions of the primitive data forms in sub-clauses 5.1 to 5.5 show the allowed precisions for each primitive data form. The definitions are in terms of 'metafile words' which are 16-bit units.

The following terms are used in the following diagrams when displaying the form of numeric values.

msb	most significant bit
lsb	least significant bit
S	sign bit

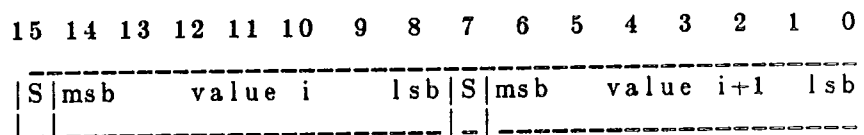
The data types in the following data diagrams are illustrated for the case that the parameter begins on a metafile word boundary. In general, parameters may begin on odd or even octet boundaries within commands, because they may be preceded by an odd or even number of octets of other parameter data.

5.1 Signed Integer

Signed integers are represented in "two's complement" format. Four precisions may be specified for signed integers: 8-bit, 16-bit, 24-bit and 32-bit. (Integer coordinate data encoded with this primitive data form do not use the 8-bit precision.)

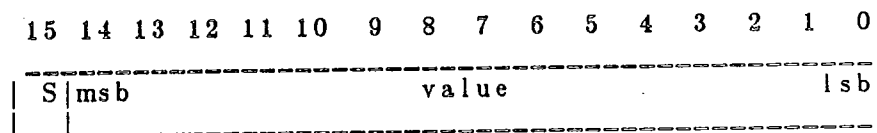
5.1.1 Signed Integer at 8-bit precision

Each value occupies half a metafile word (one octet).



5.1.2 Signed Integer at 16-bit precision

Each value occupies one metafile word.

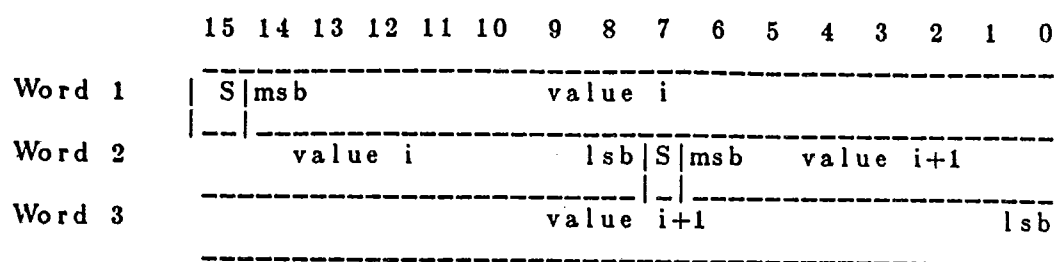


5.1.3 Signed Integer at 24-bit precision

Each value straddles two successive metafile words.

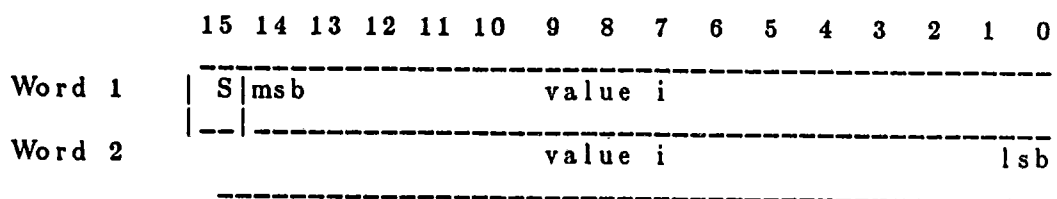
Primitive Data Forms

Signed Integer



5.1.4 Signed Integer at 32-bit precision

Each value fills two complete metafile words.

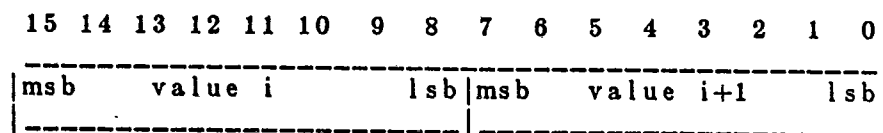


5.2 Unsigned Integer

Four precisions may be specified for unsigned integers: 8-bit, 16-bit, 24-bit and 32-bit.

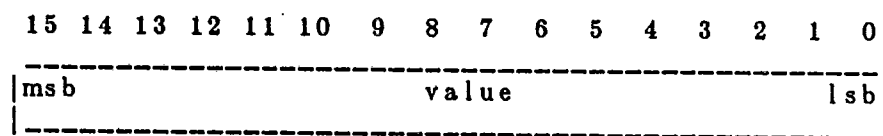
5.2.1 Unsigned Integers at 8-bit precision

Each value occupies half a metafile word.



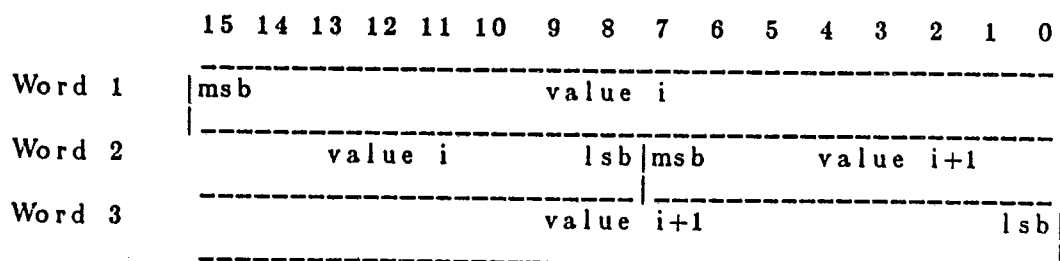
5.2.2 Unsigned Integers at 16-bit precision

Each value occupies one metafile word.



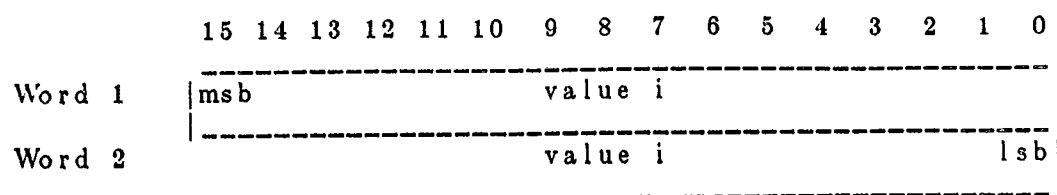
5.2.3 Unsigned Integers at 24-bit precision

Each value straddles two successive metafile words.

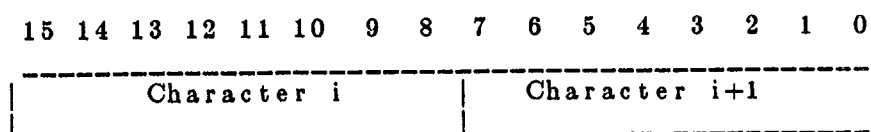


Unsigned Integer**Primitive Data Forms****5.2.4 Unsigned Integers at 32-bit precision**

Each value fills two complete metafile words.

**5.3 Character**

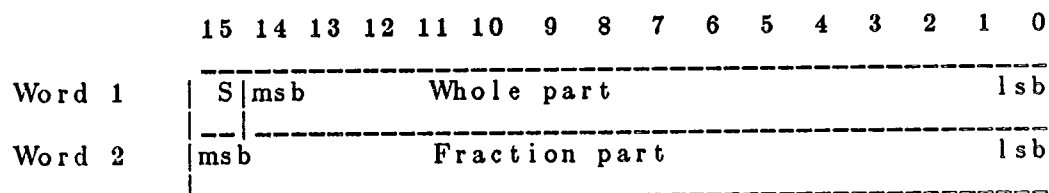
Each character is stored in an octet.

**5.4 Fixed Point Real**

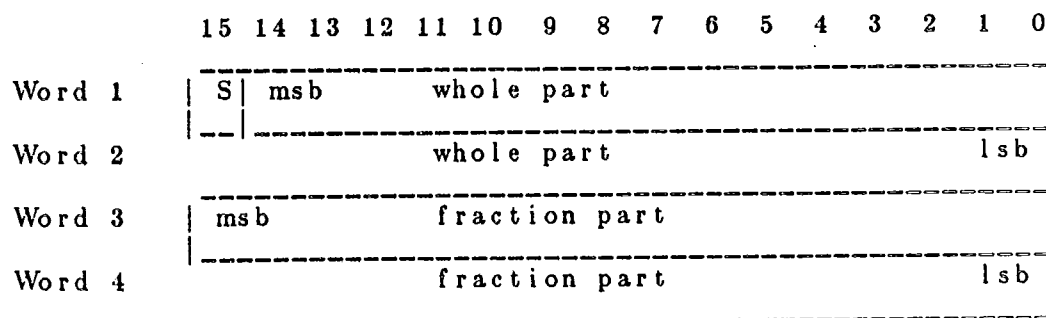
Fixed point real values are stored as two integers; the first represents the "whole part" and has the same form as a Signed Integer (SI; see 5.1); the second represents the "fractional part" and has the same form as an Unsigned Integer (UI; see sub-clause 5.2). Two precisions may be specified for Fixed Point Reals: 32-bit or 64-bit.

5.4.1 Fixed Point Real at 32-bit precision

Each Fixed Point Real occupies 2 complete metafile words; the first has the form of a 16-bit Signed Integer and the second the form of a 16-bit Unsigned Integer.

**5.4.2 Fixed Point Real at 64-bit precision**

Each Fixed Point Real occupies 4 complete metafile words; the first has the form of a 32-bit Signed Integer and the second the form of a 32-bit Unsigned Integer.



Primitive Data Forms

Fixed Point Real

5.4.3 Value of Fixed Point Reals

The values of the represented real numbers are given by:

$$\text{for 32 bits: } \text{real_value} = SI + (\text{sign } SI) \left(\frac{UI}{2^{16}} \right)$$

$$\text{for 64 bits: } \text{real_value} = SI + (\text{sign } SI) \left(\frac{UI}{2^{32}} \right)$$

SI stands for the "whole part" and UI stands for the "fractional part" in these equations.

5.5 Floating Point

Floating Point Real values are represented in the floating point format of IEEE standard P754. This format contains three parts:

- a sign bit ('s')
- a biased exponent part ('e')
- a fraction part ('f')

The value is a function of these three values ('s', 'e' and 'f'). If 's' is '0', the value is positive; if 's' is '1', the value is negative. Two precisions may be specified for Floating Point Reals: 32-bit or 64-bit. The magnitude of the value is calculated as follows for 32-bit representation.

- a. If $e = 255$ and $f \neq 0$, then the value is undefined.
- b. If $e = 255$ and $f = 0$, then the value is as large a positive ($s=0$) or negative ($s=1$) value as possible.
- c. If $0 < e < 255$, then the magnitude of the value is $(1.f)(2^{e-127})$.
- d. If $e = 0$ and $f \neq 0$, then the magnitude of the value is $(0.f)(2^{-126})$.
- e. If $e = 0$ and $f = 0$, then the value is 0.

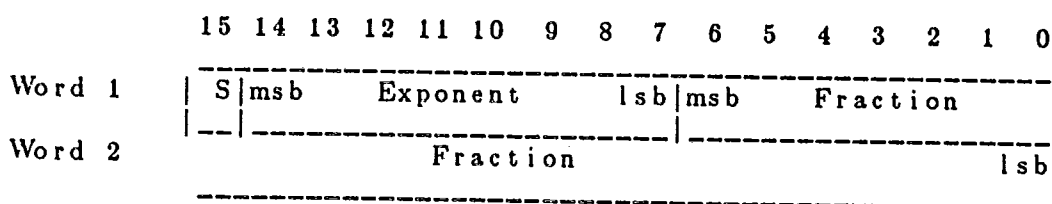
The magnitude of the value is calculated as follows for 64-bit representation.

- a. If $e = 2047$ and $f \neq 0$, then the value is undefined.
- b. If $e = 2047$ and $f = 0$, then the value is as large a positive ($s=0$) or negative ($s=1$) value as possible.
- c. If $0 < e < 2047$, then the magnitude of the value is $(1.f)(2^{e-1023})$.
- d. If $e = 0$ and $f \neq 0$, then the magnitude of the value is $(0.f)(2^{-1022})$.
- e. If $e = 0$ and $f = 0$, then the value is 0.

5.5.1 Floating Point Real at 32-bit precision

Each Floating Point Real value occupies 2 metafile words. The size of each field in the value is as follows.

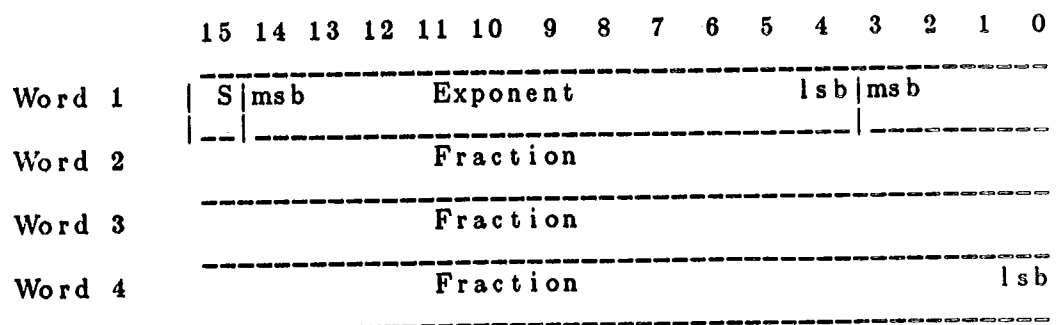
sign	1 bit
exponent	8 bits
fraction	23 bits



Floating Point**Primitive Data Forms****5.5.2 Floating Point Real at 64-bit precision**

Each Floating Point Real value occupies 4 metafile words. The size of each field in the value is as follows.

sign 1 bit
exponent 11 bits
fraction 52 bits



[ISO page 14]

This publication is a Draft for Development and is therefore not to be regarded as a British Standard. See page 1.