INTERNATIONAL STANDARD

ISO/IEC 9637-2

First edition
1992-12-01

Information technology — Computer graphics — Interfacing techniques for dialogues with graphical devices (CGI) — Data stream binding —

Part 2:
Binary encoding

Technologies de l'information — Infographie — Interfaces pour l'infographie — Interface du flux de données CGI —

Partie 2: Codage binaire

Reference number
ISO/IEC 9637-2:1992(E)

CONTENTS

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

International Standard ISO/IEC 9637-2 was prepared by Joint Technical Committee ISO/IEC JTC1, *Information technology.*

ISO/IEC 9637 consists of the following parts, under the general title *Information technology — Computer graphics — Interfacing techniques for dialogues with graphical devices (CGI) — Data stream binding*

— *Part 1: Character encoding*

— *Part 2: Binary encoding*

Annex A forms an integral part of this part of ISO/IEC 9637. Annex B is for information only.

Introduction

Purpose

The Binary Encoding of the Computer Graphics Interface (CGI), ISO/IEC 9636, provides a data stream representation of the CGI function 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 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. 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 ISO/IEC 9636-1.

Objectives

This encoding has the following features:

a) Partitioning of parameter lists: function/response representations are coded in the Binary Encoding by one or more partitions (see clause 4); the first (or only) partition of a representation contains the opcode (class code and id code);

b) Alignment of function representations and response representations: every function/response representation begins on a 16-bit boundary. Alignment of representations which follow partitions that require an odd number of 8-bit entities may require a partition to be padded with an 8-bit entity with all bits zero;

c) Uniformity of format: all function representations and response data records have an associated parameter length value. As a result, it is possible to ignore function representations which are not supported by the interpreter;

d) Efficiency of encoding parameter data: parameter data such as coordinates, indexes and colours are encoded as one or more 8-bit entities. The precision of every parameter is determined by the appropriate default precision or as set by a precision setting CGI function;

e) Extensibility: the arrangement of opcode class and id values has been designed to allow future growth;

f) Format of real data: real numbers are encoded using either IEEE floating point representation or a fixed-point representation;

g) Run length encoding option: if many adjacent colours have the same value, efficient encoding is possible. For each run a cell count is specified followed by the colour (or colour index);

h) Packed list encoding option: if adjacent colours do not have the same value, bit-stream lists are provided in which the values are packed as closely as possible;

i) Encoding of soliciting functions: the assignment of opcodes to functions which require a response has been designed so that all such functions can be recognized by a CGI interpreter;

j) Response Data: responses to soliciting functions have been assigned different opcodes from their associated soliciting functions. However, the response opcode can be derived in a straightforward manner from the soliciting function opcode;

k) Lists of data: there is a standard technique for representing lists of any type of data (with a few specific exceptions);

Relationship to other standards

This encoding is guided by the same objectives as the Computer Graphics Metafile Binary Encoding, ISO/IEC 8632-3:1992. For each CGI function which is identical in both semantics and parameterization to a CGM element, the encoding will be identical. That is, the opcodes will be identical and the parameters will use the same data type and appear in the same order. The extension mechanism defined in this encoding is also compatible with the CGM Binary Encoding.

The floating point representation of real data in this part of the Standard is that in ANSI/IEEE 754-1986.

The representation of character data in this part of the Standard follows the rules of ISO 646 and ISO 2022.

For certain functions and response data, the CGI defines parameter value ranges as being reserved for registration. The values and their meanings will be defined using the procedures established in ISO TR 9973.

Information technology — Computer graphics — Interfacing techniques for dialogues with graphical devices (CGI) — Data stream binding —

Part 2:
Binary encoding

1 Scope

This part of ISO/IEC 9637 specifies a Binary Encoding of the Computer Graphics Interface (CGI) data stream. For each of the function syntaxes in clause 5 and clause 6 of ISO/IEC 9636-2, ISO/IEC 9636-3, ISO/IEC 9636-4, ISO/IEC 9636-5, and ISO/IEC 9636-6, an encoding is specified in terms of an opcode and a sequence of parameters of specified data types. For each of these data types, an explicit representation in terms of bits, 8-bit and 16-bit entities is specified. For some data types, the exact representation depends on a type and/or precision for the data as used in the data stream.

The Binary Encoding of the Computer Graphics Interface data stream will, in many circumstances, reduce the effort required to generate and interpret the data stream as compared to other encodings.

2 Normative references

The following standards contain provisions which, through reference in this text, constitute provisions of this part of ISO/IEC 9637. At the time of publication, the editions indicated were valid. All standards are subject to revisions, and parties to agreements based on this part of ISO/IEC 9637 are encouraged to investigate the possibility of applying the most recent editions of the standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO/IEC 646:1991, *Information technology – ISO 7-bit coded character set for information interchange.*

ISO 2022:1986, *Information processing – ISO 7-bit and 8-bit coded character sets – Code extension techniques.*

ISO/IEC 7942:1985/Amd.1:1991, *Information processing systems – Computer graphics – Graphical Kernel System (GKS) functional description – Amendment 1*

ISO 8632-1:1992, *Information technology – Computer graphics – Metafile for the storage and transfer of picture description information – Part 1: Functional specification.*

ISO 8632-3:1992, *Information technology – Computer graphics – Metafile for the storage and transfer of picture description information – Part 3: Binary encoding.*

ISO/IEC 9636-1:1991, *Information technology – Computer graphics – Interfacing techniques for dialogues with graphical devices(CGI) – Functional specification – Part 1: Overview, profiles and conformance.*

ISO/IEC 9636-2:1991, *Information technology – Computer graphics – Interfacing techniques for dialogues with graphical devices(CGI) – Functional specification – Part 2: Control.*

ISO/IEC 9636-3:1991, *Information technology – Computer graphics – Interfacing techniques for dialogues with graphical devices(CGI) – Functional specification – Part 3: Output.*

ISO/IEC 9636-4:1991, *Information technology – Computer graphics – Interfacing techniques for dialogues with graphical devices(CGI) – Functional specification – Part 4: Segments.*

ISO/IEC 9636-5:1991, *Information technology – Computer graphics – Interfacing techniques for dialogues with graphical devices(CGI) – Functional specification – Part 5: Input and echoing.*

ISO/IEC 9636-6:1991, *Information technology – Computer graphics – Interfacing techniques for dialogues with graphical devices(CGI) – Functional specification – Part 6: Raster.*

ISO/IEC 9637-1:1992, *Information technology – Computer graphics – Interfacing techniques for dialogues with graphical devices(CGI) – Data stream binding – Part 1: Character encoding.*

ISO/IEC TR 9973:1988, *Information technology – Computer graphics – Procedures for registration of graphical items.*

ANSI/IEEE 754, *Standard for Binary Floating Point Arithmetic.*

3 Definitions

3.1 representation: Portion of a binary-encoded function or response representation that contains the opcode (function/response class plus function/response id) and parameter length information. (See 4.3.4.)

3.2 octet: 8-bit entity in which all bits are significant. The bits are numbered from 7 (most significant) to 0 (least significant).

3.3 word: 16-bit entity in which all bits are significant. The bits are numbered from 15 (most significant) to 0 (least significant).

3.4 word-aligned: An entity is word-aligned when it begins on a word (16-bit) boundary within the data stream.

NOTE - Within this part of ISO/IEC 9637, the terms "octet", "word", and "word-aligned" have specific meanings. These meanings may not match those of a particular computer system on which this encoding of the data stream is used.

4 Overall structure

This encoding specifies representations for each of the CGI functions of ISO/IEC 9636 as well as any associated responses. A function representation is the encoded representation of a function with its *In* parameters. A response representation is the encoded representation of the *Out* parameters of a soliciting function.

4.1 General form of the data stream

All function representations in the data stream are encoded using a uniform scheme. These are represented as variable length data structures, each consisting of opcode information (function class plus function id) designating the particular function representation, the length of its parameter data and finally the parameter data itself (if any).

All response representations in the return data stream are encoded using the same uniform scheme. These are represented as variable length data structures, each consisting of opcode information (response class plus response id) designating the particular response representation, the length of its return parameter data and finally the data itself.

4.2 General structure of the Binary Encoding

The octet is the fundamental unit of organization of the binary data stream. The Binary Encoding of the CGI data stream is a logical data structure consisting of a sequential collection of octets. Fields of two different sizes are defined within the Binary Encoding structure. These two sizes correspond to the octet, an 8-bit field, and the word, a 16-bit field. These fields are used in the remainder of this part of ISO/IEC 9637 for illustrating the contents and structure of function representations and parameters. The parameter list length of a function or response representation is expressed as a number of octets.

To optimize processing of the binary data stream on a wide collection of computers, data stream function and response representations are required to consist of an even number of octets. This forces the alignment of representations in the Binary Encoding data stream to word boundaries. It is necessary to pad a representation with null octets or bits to the word boundary if the parameter data does not fill to such a boundary. This padding of an extra octet or bits does not affect any parameter list length counts within a representation.

Parameter data can be organized into subgroups called partitions. Partitions are used to accommodate parameter data larger than that supported by the basic long-form representation, as described below. Partitions need not begin on a word boundary.

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

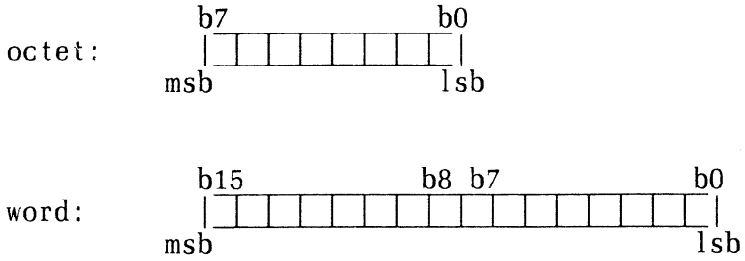


Figure 1 - Bit numbering for octets and words

If the consecutive bits of the binary data structure are numbered 1..*N*, the consecutive octets are numbered 1..*N*/8 (rounded to next integer), and the consecutive words are numbered 1..*N*/16 (rounded to next integer), then the logical correspondence of bits, octets, and words in the binary data structure is as illustrated in table 1.

Table 1 - Binary data structure

Data stream 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
.	.	.
.	.	.

4.3 Encoding functions

The function descriptions in clause 5 and clause 6, as well as the formal grammars, of ISO/IEC 9636-2, ISO/IEC 9636-3, ISO/IEC 9636-4, ISO/IEC 9636-5, and ISO/IEC 9636-6, provide the basic syntax needed to encode each CGI function. They contain the function name and the input and output parameters, along with their abstract data types and value ranges. The order in which the parameters are specified in clause 5 and clause 6 is significant. This is the exact order in which they will be encoded in a representation with very few exceptions. These special parameter encoding cases are specifically detailed. (See 6.1.)

The Binary Encoding scheme has two basic components: the representation header and the parameter data. The representation header provides information concerning the function opcode (function class and function id) and the amount (in octets) of parameter data that is being supplied in the function representation. There are one or more function representation headers for each function representation in the data stream. There may be no parameter data, a fixed amount of parameter data or an indefinite amount of parameter data depending on the function being represented.

CGI functions and their *In* parameters, if any, are encoded as function representations consisting of one or more representation headers followed by any *In* parameter data. Functions with *Out* parameters are called soliciting functions. Their *Out* parameters are considered response data and are encoded in a separate response representation, consisting of one or more representation headers followed by the *Out* parameter data. (See 4.5.)

The formats of the various function and response representations and the rules for their encoding into the binary data stream are described below.

4.3.1 General structure of the representation header

Representations in the Binary Encoding have four forms - basic short-form representations, basic long-form representations, extended short-form representations and extended long-form representations. The forms differ in format with regard to the amount of parameter data accommodated and the number of function class and function id values accommodated. Specifically, the differences are:

-- the short-form representation always contains the complete function representation, including its parameter list data. The long-form representation is used to supply an indefinite amount of parameter data through the use of data partitions;

-- the short-form representation only accommodates parameter list data up to 30 octets in length. The long-form representation accommodates lengths up to 32767 octets per data partition with an indefinite number of partitions.

-- the basic form representations accommodate only 14 function classes and 127 function ids. The extended form representations accommodate an unlimited number of function classes and function ids.

The representation forms also differ in the format of the representation headers. There are specific fields, unique to each type of representation header, that identify the type of representation header. There are also differences in how the function opcode (function class and function id) and the parameter list length are encoded.

4.3.2 Basic short-form representation header

For the basic short-form, the representation header consists of a single word divided into three fields: function class, function id and parameter list length.

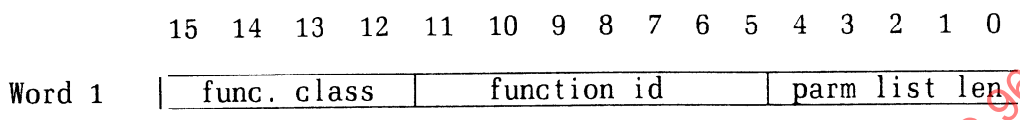


Figure 2 - Format of a basic short-form representation header.

The fields in the basic short-form representation header are as follows:

bits 15-12 function class (value range 0-14)

bits 11-5 function 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)

4.3.3 Basic long-form representation header

The representation header of the basic long-form representation consists of two words. The first word of the basic long-form representation has the same structure as the first word of the basic short-form representation with the difference that the parameter list length field contains the binary value 11111 (decimal 31). This value indicates that the header is a basic long-form representation header rather than a basic short-form representation header. The second word contains the length of the following parameter data partition and a flag that indicates if the parameter data partition is followed by another data partition or not.

An indefinite number of parameter partitions can be accommodated by the long-form representation. When the partition flag, bit 15 of the second word of the header has the value 1, it indicates that there will be another parameter data partition to follow. Each subsequent data partition of the function representation is preceded by a word composed of a partition flag and the partition's parameter list length in the same format as the second word of the representation header. The final data partition of a function representation is indicated by the partition flag being set to zero.

The parameter list length supplied in each of the parameter data partitions specifies the length of that partition in octets and not the length of the complete function representation data. Partitions need not start on a word boundary, therefore no padding octet needs to be appended to the data in a data partition which contains an odd number of octets for its parameter list length.

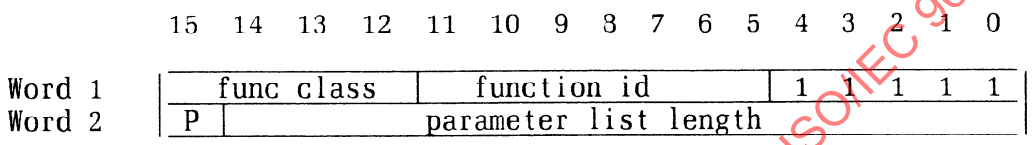


Figure 3 - Format of a basic long-form representation header.

The fields in the basic long-form representation header are as follows:

Word 1

bits 15-12 function class (value range 0-14)

bits 11-5 function representation id (value range 0-127)

bits 4-0 binary value 11111 (decimal 31) indicating long-form

Word 2

bit 15 P, partition flag
- 0 if 'final' partition
- 1 if 'not final' partition

bits 14-0 parameter list length: the number of octets of parameter data that follow for this partition (value range 0-32767).

4.3.4 Extended-form representation header

The extended-form representation header is composed of one or more extender headers followed by either a basic short-form or a basic long-form representation header. If the last header is a basic short-form header, the representation header is referred to as an extended short-form representation header. If the last header is a basic long-form header, the representation header is referred to as an extended long-form representation header.

The first extender header is recognized by having a binary value of 1111 (decimal 15) in the field normally containing the function class code in a basic header, bits 12 through 15. Each succeeding extender header also has this value in this field. The extender header is also composed of an extension field selector (FS), a final extension header flag (FE), extension data, and a six bit constant with the value of zero. A value of 1 for FE, bit 10 of the extender header, indicates that there are more extender headers to follow. A value of 0 for FE indicates that this is the final extender header and the next word in the data stream will be one of the basic representation headers. A value of 0 for FS, bit 11 of the extender header, indicates that the extension data is to be used in constructing the function class code. A value of 1 for FS indicates that the extension data is to be used in constructing the function id code.

An extended function class is constructed from the extender headers by appending the bits in the extension data to the bits in the class code already accumulated. As each extender header that composes a particular extended function class is encountered in the data stream, the extension data bits are accumulated. That is, the higher order bits are encoded in the extender headers encountered earlier in the data stream. The bits in the class code field of the short-form or long-form basic function respresentation header trailing the last extender header are appended to the accumulated function class value to form the final function class code. An extended function id code is constructed in the same manner.

Extender headers for constructing function class and function id codes can be transmitted in any order relative to one another as long as the higher-order-bits-first rule is followed. That is, it does not matter if the extender headers forming a function class code are transmitted intermixed with the extender headers forming a function id code.

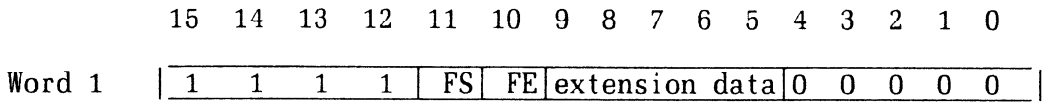


Figure 4 - Format of an extender header.

The fields in the extender header are as follows.

bits 15-12 binary value 1111 (decimal 15) indicating an extender header

bit 11 FS, extension field selector
- 0 if extension data applies to class code
- 1 if extension data applies to id code

bit 10 FE, final extension header flag
- 0 if 'final' extender header
- 1 if 'not final' extender header

bits 9-5 extension data

bits 4-0 binary value 00000

Note that while the function class code field of a representation header normally uses the binary value 1111 to indicate an extender header, an exception is made when the value of this field is in the basic header representation that follows a set of extender headers. This is the only means of specifying a function class with a binary value of 1111 (decimal 15) in bits 15-12. (See clause B.8.)

Also note that the use of extender headers to specify the class and id codes will not necessarily result in a unique Binary Encoding of a particular representation opcode. Refer to annex B for examples.

Theoretically, this extension technique is infinitely extensible, however, for this part of ISO/IEC 9637, a CGI interpreter will be expected to parse only a limited number of extender headers. Specifically, no CGI interpreter will be expected to accumulate more than two extension data fields for extending the class code. This allows for 16384 possible class code values. Likewise, no CGI interpreter will be expected to accumulate more than two extension data fields for extending the id code and no assigned id code will exceed 65535 (16-bits in size).

4.4 Encoding parameter data

When present in a function or response representation, the parameter values follow the parameter list length portion of a representation header for each representation form. The parameter list length is determined by the number of parameters and the type and precision of the parameters. Parameter values have the format illustrated in clause 5 of this part of ISO/IEC 9637. The possible parameter types are as specified in 5.2.10 of ISO/IEC 9636-1. Parameters are encoded in a function representation in exactly the same order as they are listed in a CGI function description in clause 5 of ISO/IEC 9636-2, ISO/IEC 9636-3, ISO/IEC 9636-4, ISO/IEC 9636-5, and ISO/IEC 9636-6. There are a few exceptions to this rule. These special cases are described in 6.1.

Every representation is constrained to begin on a word boundary. This necessitates padding a representation with a single null octet at the end if it contains an odd number of octets. In addition, in function representations with parameters whose precisions are shorter than one octet (i.e. those containing a 'local colour precision' parameter) it is necessary to pad the low order bits of the last data-containing octet with null bits if the data does 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 representation that padding is performed, with the exception of the function representations and response representations which encode their colour specifier list using the run length list mode or the packed list mode. (See 6.1.7.)

4.5 Encoding function response data

CGI functions which have *Out* parameters are called soliciting functions. These functions return the *Out* parameter values in associated response representations. The function id code for a soliciting function and its associated response representation are identical. The class code for each of these differs only in bit 4 of the accumulated class code. In the response representation, this bit has a value of 1; in the function representation, this bit has a value of 0. The response class code is derived by adding decimal 16 to the soliciting function class code.

All soliciting CGI functions have as their first return parameter a response validity value. When the response validity is INVALID, no additional return parameters need to be encoded in the response representation. If any other return parameters are present in the response representation, they should be regarded as undefined.

4.5.1 Structure of response representations

Response representations have exactly the same structure and encoding rules for their headers and parameter data as function representations. Both soliciting function representations and response representations can always be recognized as such by their assigned class codes. All soliciting functions will have bit 5=1 and bit 4=0 in their class code. All responses will have bit 5=1 and bit 4=1 in their class code. (See annex A.)

5 Binary Encoding primitive data forms

The Binary Encoding uses five primitive data forms to represent the various abstract data types used to describe parameters in ISO/IEC 9636-2 through 6. The primitive data forms and the symbols used to represent them are as follows.

SI	Signed Integer
UI	Unsigned Integer
O	Octet
FX	Fixed Point Real
FP	Floating Point Real

Each of these primitive data forms (except Octet) may be used in a variable number of precisions. CGI functions have been defined to effect changes in these precisions. (See 6.1.1.)

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 (0 = non-negative; 1 = negative)

where the sign bit (S) is the high order bit for signed integer values.

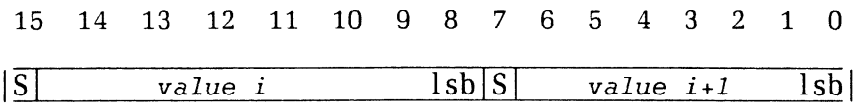
The primitive data forms in the following diagrams are illustrated for the case that the parameter begins on a data stream word boundary. In general, parameters may align on odd or even octet boundaries because they can be preceded by an odd or even number of octets of other parameter data. Function representations containing a local colour precision parameter may have parameters shorter than one octet. It is possible in such cases that the individual parameters will not align on octet boundaries. (See 6.1.7.)

5.1 Signed Integer

Signed Integer data forms are represented in two's complement format. Four precisions are available for Signed Integers: 8-bit, 16-bit, 24-bit and 32-bit.

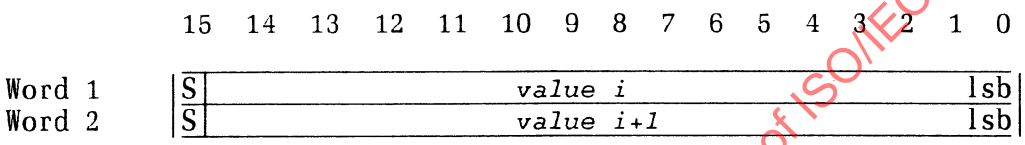
5.1.1 Signed Integer at 8-bit precision

Each value occupies one octet. The range is $-128 \leq value \leq 127$. The sign bit is 1 for negative values.



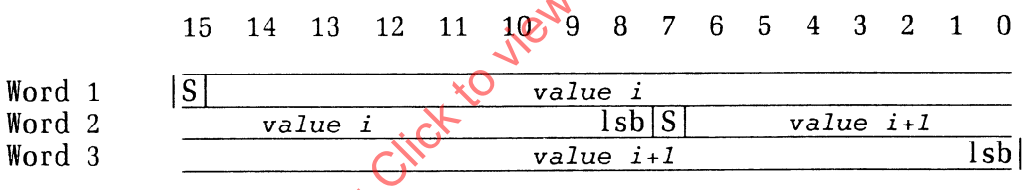
5.1.2 Signed Integer at 16-bit precision

Each value occupies two successive octets.
The range is $-32,768 \leq value \leq 32,767$. The sign bit is 1 for negative values.



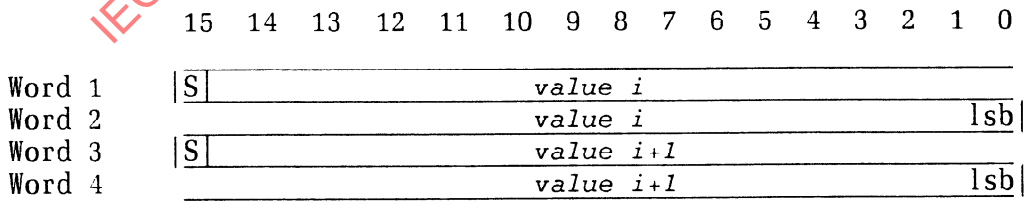
5.1.3 Signed Integer at 24-bit precision

Each value occupies three successive octets.
The range is $-8,388,608 \leq value \leq 8,388,607$. The sign bit is 1 for negative values.



5.1.4 Signed Integer at 32-bit precision

Each value occupies four successive octets.
The range is $-2,147,483,648 \leq value \leq 2,147,483,647$. The sign bit is 1 for negative values.

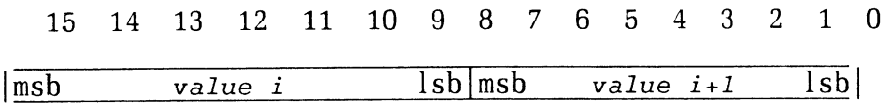


5.2 Unsigned Integer

Four precisions may be specified for Unsigned Integer data forms: 8-bit, 16-bit, 24-bit and 32-bit.

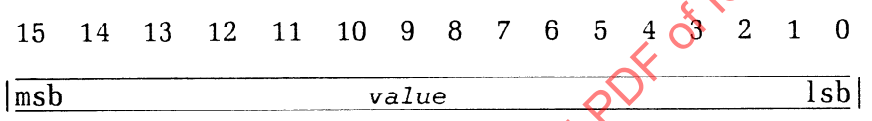
5.2.1 Unsigned Integer at 8-bit precision

Each value occupies one octet. The range is $0 \leq value \leq 255$.



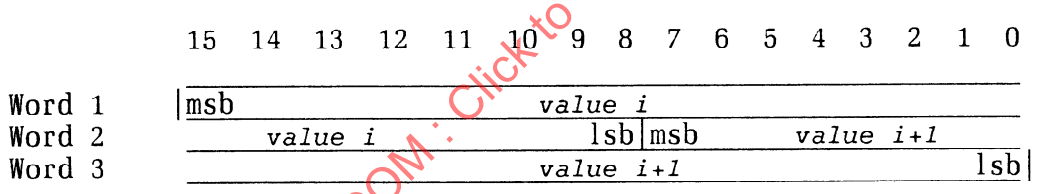
5.2.2 Unsigned Integer at 16-bit precision

Each value occupies two successive octets.
The range is $0 \leq value \leq 65,535$.



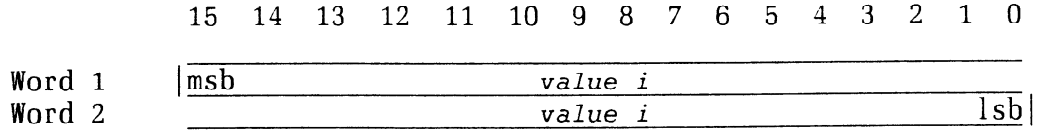
5.2.3 Unsigned Integer at 24-bit precision

Each value occupies three successive octets.
The range is $0 \leq value \leq 16,777,215$.



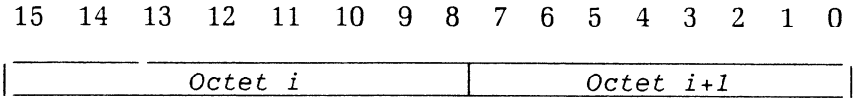
5.2.4 Unsigned Integer at 32-bit precision

Each value occupies four successive octets.
The range is $0 \leq value \leq 4,294,967,295$.



5.3 Octet

Each Octet primitive data form occupies a single octet.

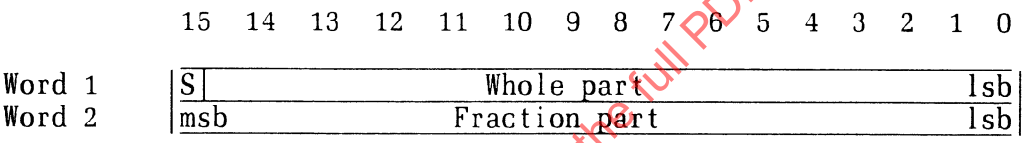


5.4 Fixed Point Real

Fixed Point Real data forms are composed of a single Signed Integer data form followed by a single Unsigned Integer data form. The first integer represents the whole part and the second integer represents the fractional part. Two precisions may be specified for Fixed Point Reals: 32-bit or 64-bit.

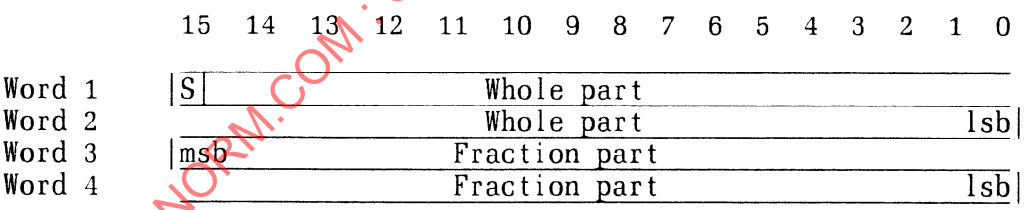
5.4.1 Fixed Point Real at 32-bit precision

Each value occupies 4 successive octets. The first 2 octets form a 16-bit Signed Integer data form. The remaining 2 octets form a 16-bit Unsigned Integer data form.



5.4.2 Fixed Point Real at 64-bit precision

Each value occupies 8 successive octets. The first 4 octets form a 32-bit Signed Integer data form. The remaining 4 octets form a 32-bit Unsigned Integer data form.



5.4.3 Values of Fixed Point Real

The values of the represented real numbers are given by:

$$\text{for 32 bits:} \quad \text{real value} = SI + \frac{\{UI\}}{\{2^{16}\}}$$

$$\text{for 64 bits:} \quad \text{real value} = SI + \frac{\{UI\}}{\{2^{32}\}}$$

SI stands for the whole part and UI stands for the fractional part in these equations. SI, the whole part, is the largest integer less than or equal to the real number being represented.

5.5 Floating Point Real

Floating Point Real data forms are represented in the floating point format of ANSI/IEEE standard 754. This format contains three parts:

- a sign bit;
- a biased exponent part;
- a fraction part.

Two precisions may be specified for Floating Point Reals: 32-bit or 64-bit. The 32-bit precision occupies 4 successive octets. The 64-bit precision occupies 8 successive octets.

6 Representation of abstract parameter data types

Table 2 shows, for each of the CGI basic data types, how it is represented in the Binary Encoding of the CGI in terms of primitive data forms. Table 3 shows, for each of the CGI abstract data types, how it is represented in terms of the CGI basic data types or other abstract data types. The columns of tables 2 and 3 are as follows:

1) The symbol for the data type, as it is specified in clause 5, ISO/IEC 9636-1;

2) The way the parameter type is constructed in terms of the primitive data forms, the CGI basic data types or other CGI abstract data types, at the appropriate precisions. The order in which the primitive data forms and basic or abstract data types are constructed is important. The precisions are those defined in clause 5, ISO/IEC 9636-2. Refer to clause 8 for the default precisions for this encoding;

3) The formula for computing the number of octets required to represent one instance (occurrence) of the given parameter, at the given precision.

In table 2, the following sample notation will be used:

- *n*O denotes '*n*' octets;
- 3UI denotes 3 unsigned integers;
- E,I,R denotes an enumerated value, an integer and a real;
- VDC,VDC denotes a pair of VDC values.

Representation of abstract parameter data types

Table 2 – Representation of basic data types

Basic data type symbol	Parameter construction from primitive forms	Formula for computing number of octets per parameter data type	
CD	3UI at colour precision (dcp)	3*dcp/8	(Note 1,2)
CI	UI at colour index precision (cip)	cip/8	(Note 2)
CSN	SI at client specified name precision (csnp)	csnp/8	
D	UI,*n*O	(*n*+1) or (*n*+3)	(Note 3)
E	SI at fixed precision (16-bit)	2	(Note 4)
I	SI at integer precision (ip)	ip/8	
ICO			(Note 5)
IF8	SI at fixed precision (8-bit)	1	
IF16	SI at fixed precision (16-bit)	2	
IF32	SI at fixed precision (32-bit)	4	
IX	SI at index precision (ixp)	ixp/8	
R	FP or FX at real precision (rp)	sum(rp)/8	(Note 6)
S	UI,*n*O	(*n*+1) or (*n*+3)	(Note 8)
SF	UI,*n*O	(*n*+1) or (*n*+3)	(Note 9)
VDC	SI at VDC integer precision (vip) or FP or FX at VDC real precision (vrp)	vip/8 or sum(vrp)/8	(Note 7) (Note 6)

NOTES

1 A direct colour is a triple of values, each of which is abstractly a real in the range [0,1]. This value is normalized onto the unsigned range $[0,2^{dcp}-1]$.

2 Certain CGI functions, which transmit large numbers of CD or CI values, from and to the client, make provision for local colour precision. (See 6.1.7 to 6.1.9 inclusive.)

3 Data records are encoded as a count (unsigned integer) followed by octets of data. The symbols UI,nO in the table refer to the octet count followed by 'n' octets. Note that this formula only applies to data within a single partition. (See 6.1.2.)

Table 2 - Representation of basic data types - (concluded)

NOTES

4 The general rule for assigning integer values to an enumerated is based on the order in which they appear in enumerated type lists in the function specification in clauses 5 and 6 of ISO/IEC 9636-2, ISO/IEC 9636-3, ISO/IEC 9636-4, ISO/IEC 9636-5, and ISO/IEC 9636-6. The first enumerated in a list is assigned value 0, the second enumerated is assigned value 1, and so on. Special enumerated assignments have been made for the INHERITANCE FILTER filter selection list parameter. (See 6.1.4.) Private (non-standard) values of enumerated parameters shall use negative integers.

5 The abstract data type ICO, input colour, obtains its representation and precision based on the Colour and Bits Per Colour entries in the Raster LID State List. When Colour is YES, realization is identical to CD with local colour precision = Bits Per Colour. When Colour is NO, realization is UI at local colour precision = Bits Per Colour. When Colour is YES, realization is 3UI at local colour precision = Bits Per Colour. (See 6.1.8.2.)

6 The REAL PRECISION function is encoded as an indicator (fixed or floating point) and two precision components. The symbol "sum(rp)" in the table indicates the sum of the number of bits specified in the two components. The same considerations apply to the encoding of the VDC REAL PRECISION function and the symbol "sum(vrp)" in the table. The VDC REAL PRECISION control function representation may cause 'vrp' to be updated in the body of data stream. (See 6.1.1.)

7 The abstract data type VDC, a single VDC value, is either a real or an integer, depending on the VDC TYPE in effect.

8 Strings are encoded as a count (unsigned integer) followed by octets of character data. The symbols UI,nO in the table refer to the octet count followed by 'n' octets. Each character obtains its representation based on the character set and character coding announcer controls in effect. This may result in characters which occupy several octets such as those in the multi-byte G-set for Japanese characters. Note that this formula only applies to data within a single partition. (See 6.1.2.)

9 Strings with fixed representation are encoded as a count (unsigned integer) followed by octets of character data. In this case, since the strings use the fixed character set ISO/IEC 646 and are not subject to interpretation according to the character coding announcer, each character occupies a single octet. The symbols UI,nO in the table refer to the octet count followed by 'n' octets. Note that this formula only applies to data within a single partition. (See 6.1.2.)

Table 3 - Representation of abstract data types
(Listed alphabetically)

Abstract parameter data type symbol	Parameter construction from primitive forms and abstract data types	Formula for computing number of octets per parameter data type	
ASN	CSN	csnp/8	
BN	CSN	csnp/8	
CO	CI or CD	cip/8 or 3*dcp/8	(Note 1,2) (Note 2,3)
CS	E,S	2 + (*n*+1) or (*n*+3)	(Note 4)
DC	IF16	2	
DP	DC,DC	4	
EI	IN,IN	4	
ER	EI,IF16 or EI,FN	6	
EV	D		(Note 4,5)
EVL	D		(Note 4,6)
FN	IN	2	
IN	IF16	2	
ISC	IF16	2	
ISP	ISC,ISC	4	
P	VDC,VDC	2*vip/8 or 2*sum(vrp)/8	 (Note 7)
PN	CSN	csnp/8	
PRN	IN	2	
PV	SN,PN	2*csnp/8	
SN	CSN	csnp/8	
SS	R or VDC	sum(rp/8) or 2*vip/8 or 2*sum(vrp)/8	 (Note 7,8)
VC	R or I	sum(rp/8) or ip/8	(Note 7) (Note 9)
VP	VC,VC	2*sum(rp/8) or 2*ip/8	(Note 7) (Note 9)

NOTES

1 The abstract data type CO, a colour value, is either direct colour (CD) or indexed colour (CI), depending on the parameter value of the function COLOUR SELECTION MODE.

2 Certain CGI functions,which transmit large numbers of CD or CI values, from to to the client, make provision for local colour precision. (See 6.1.7 to 6.1.9 inclusive.)

Table 3 – Representation of abstract data types – (concluded)

NOTES

3 A direct colour is a triple of values, each of which is abstractly a real in the range [0,1]. This value is normalized onto the unsigned range $[0,2^{dcp}-1]$.

4 Data records are encoded as a count (unsigned integer) followed by octets of data. The symbols UI,nO in the table refer to the octet count followed by 'n' octets. Note that this formula only applies to data within a single partition. (See 6.1.2.)

5 The abstract data type EV, event report, follows the rules for encoding data records. Its contents and structure depends upon the input class associated with the event. (See 6.1.2.)

6 The abstract data type EVL is a list of events of abstract data type EV. The entire list is treated as a one data record and thus follows the rules for encoding data records. (See 6.1.2.)

7 The REAL PRECISION function is encoded as an indicator (fixed or floating point) and two precision components. The symbol "sum(rp)" in the table indicates the sum of the number of bits specified in the two components. The same considerations apply to the encoding of the VDC REAL PRECISION function and the symbol "sum(vrp)" in the tables. The VDC REAL PRECISION control function representation may cause 'vrp' to be updated in the body of data stream. (See 6.1.1.)

8 The abstract data type SS, specification mode value is either a real or a VDC depending on the EDGE WIDTH SPECIFICATION MODE, LINE WIDTH SPECIFICATION MODE and MARKER SIZE SPECIFICATION MODE in effect. Context determines which specification mode is relevant.

9 The abstract data type VC, a single viewport coordinate, is either a real or an integer, depending on the DEVICE VIEWPORT SPECIFICATION MODE in effect.

6.1 Special Binary Encoding rules

The majority of the CGI functions and responses can be encoded in a straight forward manner by following the general Binary Encoding rules (see 4.3) and utilizing the opcode and parameter data type assignment tables (see tables 2, 3 and 8). There are, however, some functions/responses, parameters and parameter types which require special definition.

6.1.1 Encoding the precision functions

The CGI Binary Encoding contains functions which are not specified in ISO/IEC 9636. These encoding-specific functions are the precision setting functions. They exist in order to specify the details of encoding the data stream passed over the communications link between a CGI Generator/Interpreter pair with respect to CGI data types which admit variable precision. If these functions are not present in a CGI Binary Encoding data stream, the default precisions are assumed. (See clause 8.)

The generation and use of these precision functions is directly dependent on the client's requirements for numerical precision in representing and communicating information in CGI data streams. The client's requirements are specified in the CGI numerical precision requirement functions. These requirements are either expressed in terms of base 2 logarithms or number of bits, depending on the particular function. Using the client's requirements, a Generator of a CGI Binary Encoding makes an independent decision regarding which Binary Encoding specific precision will satisfy the client's needs. Note that the Generator need not match the client's requirements exactly; the Generator need only use a precision which satisfies the client's requirements. Once the Generator has decided upon a precision to use, the Generator shall set the precision by encoding the appropriate Binary Encoding precision function into the CGI data stream (if it differs from the currently set precision). If the Generator decides that the client's precision requirements must be satisfied by a precision which differs from the currently set precision, then the corresponding Binary Encoding specific precision function representation shall be placed into the data stream immediately after the precision requirement function representation.

NOTE - A numerical precision requirement function invoked by a CGI client is always encoded into the CGI data stream.

Table 4 lists the relationship between the CGI variable precision basic data types, the CGI abstract numerical precision requirement functions and the CGI Binary Encoding specific precision functions.

Table 4 - Data types and matching precision functions

Data type	Precision requirement function	Precision function
VDC (integer)	VDC INTEGER PRECISION REQUIREMENT	VDC INTEGER PRECISION
VDC (real)	VDC REAL PRECISION REQUIREMENTS	VDC REAL PRECISION
CD	COLOUR PRECISION REQUIREMENT	COLOUR PRECISION
CI	COLOUR INDEX PRECISION REQUIREMENT	COLOUR INDEX PRECISION
CSN	CLIENT SPECIFIED NAME PRECISION REQUIREMENT	CLIENT SPECIFIED NAME PRECISION
I	INTEGER PRECISION REQUIREMENT	INTEGER PRECISION
IX	INDEX PRECISION REQUIRMENT	INDEX PRECISION
R	REAL PRECISION REQUIREMENTS	REAL PRECISION

6.1.1.1 Binary Encoding precision functions

This subclause defines the abstract functional specifications for the CGI Binary Encoding specific precision functions. Each function shall be encoded according to the general Binary Encoding rules (see 4.3), utilizing the opcode and parameter basic data type assignment tables (see tables 2 and 8).

Each function description consists of:

-- function name;
-- input parameters, along with their abstract data types (note, *In* indicates that the parameter values are provided by the CGI Generator);
-- effect of the function.

VDC INTEGER PRECISION

Parameters:

In	VDC integer precision specification	(8,16,24 or 32)	I

Effect:

The VDC Integer Precision value in the CGI Generator and CGI Interpreter state information is set to the value specified. It applies to the operands of basic data type VDC for subsequent transactions in both directions when the VDC type is INTEGER.

VDC REAL PRECISION

Parameters:

In	form of representation for VDC real	(FLOAT_PT, FIXED_PT)	E
In	field width for exponent or whole part(including sign bit)	(9,12,16 or 32)	I
In	field width for fraction or fractional part	(23,52,16 or 32)	I

Effect:

The VDC Real Precision value in the CGI Generator and CGI Interpreter state information is set to the value specified. It applies to the operands of basic data type VDC for subsequent transactions in both directions when the VDC type is REAL.

The field widths are specified in number of bits. The legal combinations of parameter values for the resulting real precisions are:

Form	Exponent Field Width	Fraction Field Width	Resulting Real Precision
0	9	23	32-bit floating point
0	12	52	64-bit floating point
1	16	16	32-bit fixed point
1	32	32	64-bit fixed point

COLOUR PRECISION

Parameters:

In	colour precision specification	(8,16,24 or 32)	I

Effect:

The Colour Precision value in the CGI Generator and CGI Interpreter state information is set to the value specified and applies to the operands of basic data type CD (colour direct) for subsequent transactions in both directions.

COLOUR INDEX PRECISION

Parameters:

In	colour index precision specification	(8,16,24 or 32)	I

Effect:

The Colour Index Precision value in the CGI Generator and CGI Interpreter state information is set to the value specified and applies to the operands of basic data type CI (colour index) for subsequent transactions in both directions.

CLIENT SPECIFIED NAME PRECISION

Parameters:

In	client specified name precision specification	(8,16,24 or 32)	I

Effect:

The Client Specified Name Precision value in the CGI Generator and CGI Interpreter state information is set to the value specified and applies to the operands of the basic data type CSN (client specified name) for subsequent transactions in both directions.

INTEGER PRECISION

Parameters:

In	integer precision specification	(8,16,24 or 32)	I

Effect:

The Integer Precision value in the CGI Generator and CGI Interpreter state information is set to the value specified and applies to the operands of data type I (integer) for subsequent transactions in both directions.

INDEX PRECISION

Parameters:

In	index precision specification	(8,16,24 or 32)	I

Effect:

The Index Precision value in the CGI Generator and CGI Interpreter state information is set to the value specified and applies to the operands of basic data type IX (index) for subsequent transactions in both directions.

REAL PRECISION

Parameters:

In	form of representation for real	(FLOAT_PT, FIXED_PT)	E
In	field width for exponent or whole part(including sign bit)	(9,12,16 or 32)	I
In	field width for fraction or fractional part	(23,52,16 or 32)	I

Effect:

The Real Precision value in the CGI Generator and CGI Interpreter state information is set to the value specified and applies to the operands of basic data type R (real) for subsequent transactions in both directions.

The field widths are specified in number of bits. The legal combinations of parameter values for the resulting real precisions are:

Form	Exponent Field Width	Fraction Field Width	Resulting Real Precision
0	9	23	32-bit floating point
0	12	52	64-bit floating point
1	16	16	32-bit fixed point
1	32	32	64-bit fixed point

6.1.2 Encoding strings, fixed strings and data records

String, fixed string, and data record parameter data are encoded as an octet count followed by the data. The encoding of the count is similar to the encoding of the parameter list length for function and response representations in that there is a short-form and a long-form. The short-form only accommodates strings and data records up to 254 octets in length; the long-form accommodates lengths up to 32767 octets per string or data partition with an indefinite number of partitions.

The short-form representation consists of an octet containing the count in the range 0...254 decimal followed by the data. The long-form consists of an octet containing the decimal value 255 followed by two successive octets (forming the count word) which contain the length of the following string or data partition and a flag that indicates if the partition is followed by another string partition.

NOTE - All octets, including count words and string data, are packed into successive octets. Refer to annex B for examples.

An indefinite number of string or data partitions can be accommodated by the long-form string representation. Each partition is preceded by a count word composed of a partition flag and the partition's octet length in the same format. The partition flag values are listed below:

0	denotes 'final' partition;	this partial string or data record completes the string or data record parameter data.
1	denotes 'not final' partition;	this partial string or data record will be followed by another partial string or data record.

6.1.2.1 Encoding data record contents

Data records use the basic data types, plus data type VC and binary encoding specific data type CL, and have internal structure. The content of a data record is encoded as an ordered set of sub-sequences of parameters of a given data type. Each such sub-sequence is introduced by a header consisting of two items: the first is of data type Index (IX), indicating the data type for the parameters in the sub-sequence; the second is an Integer (I) count for the number of following parameters of that type. The header is encoded in accordance with the current precision for Index (IX) and Integer (I). The following parameters are encoded in the usual manner according to the encoding rules for parameters of the specified type. In particular, data types subject to type, precision, or a specification mode are encoded in accordance with the relevant controls in effect.

Table 5 - Data type index assignments within data records

Value	Data type	
1	D (Data Record)	
2	CI (Colour Index)	
3	CD (Direct Colour Value)	
4	CSN (Client Specified Name)	(Note 1)
5	E (Enumerated)	
6	I (Integer)	
7	ICO (Input Colour)	
8	IF8 (Fixed 8-bit Integer)	
9	IF16 (Fixed 16-bit Integer)	
10	IF32 (Fixed 32-bit Integer)	
11	IX (Index)	
12	R (Real)	
13	S (String)	
14	SF (Fixed String)	
15	VC (Viewport Coordinate)	
16	VDC (Virtual Device Coordinate)	
17	<reserved>	(Note 2)
18	<reserved>	(Note 2)
19	<reserved>	(Note 2)
20	<reserved>	(Note 2)
21	CL (Colour Specifier List)	

NOTES

1 Data type CSN (Client Specified Name) in ISO/IEC 9636 is the same as data type N (Name) in ISO/IEC 8632

2 <reserved> data type indices are assigned to ISO/IEC 8632 data types.

6.1.2.2 String parameters and character sets

The following CGI functions have one or more string parameters of data type S (String):

MESSAGE	ECHO REQUEST STRING
TEXT	DEQUEUE STRING EVENT
APPEND TEXT	AWAIT EVENT QUEUE TRANSFER
RESTRICTED TEXT	PUT CURRENT STRING MEASURE
REQUEST STRING	SET STRING DEVICE DATA
SAMPLE STRING	UPDATE STRING ECHO OUTPUT

These string parameters, whether output from the generator, or returned by the interpreter, are affected by the CHARACTER SET INDEX function, which designates a particular character set (from the list established by CHARACTER SET LIST functions) as the G0 set. Likewise, ALTERNATE CHARACTER SET INDEX function designates a particular character set as both the G1 and G2 sets.

CGI functions which have parameters of data type SF (Fixed String) are:

FONT LIST
INQUIRE LIST OF AVAILABLE TEXT FONTS
INQUIRE FONT CAPABILITIES

These parameters always use the fixed character set ISO/IEC 646 and are not subject to control by the CHARACTER CODING ANNOUNCER.

6.1.3 Encoding <input class> input functions

Some of the input functions are described in ISO/IEC 9636 in a generic fashion. They differ depending on the logical input device (LID) class, one of: LOCATOR, STROKE, VALUATOR, CHOICE, PICK, STRING, RASTER or GENERAL. In this encoding, these functions will be encoded as separate function representations. Likewise, any responses for functions of this sort will be encoded as separate response representations.

6.1.4 INHERITANCE FILTER enumerated values

The general rule for assigning an integer value to an enumerated is not applicable to the filter list parameter of the function INHERITANCE FILTER since the list is specified in tables by category. The integer values are assigned to the filter list categories in the prescribed order of:

a) all Individual ASF Names;
b) all Individual Attribute Names;
c) all ASF Group Names;
d) all Attribute Group Names.

The formal grammar in ISO/IEC 9636-4 lists the enumerated data types in this prescribed order resulting in the assigned values listed in Table 6 below.

Table 6 – Inheritance filter selection list enumerated values

Value	Filter selection list enumerated
0	LINE TYPE ASF
1	LINE WIDTH ASF
2	LINE COLOUR ASF
3	MARKER TYPE ASF
4	MARKER SIZE ASF
5	MARKER COLOUR ASF

Table 6 - Inheritance filter selection list enumerated values - (continued)

Value	Filter selection list enumerated
6	TEXT FONT INDEX ASF
7	TEXT PRECISION ASF
8	CHARACTER EXPANSION FACTOR ASF
9	CHARACTER SPACING ASF
10	TEXT COLOUR ASF
11	INTERIOR STYLE ASF
12	FILL COLOUR ASF
13	HATCH INDEX ASF
14	PATTERN INDEX ASF
15	FILL BITMAP ASF
16	EDGE TYPE ASF
17	EDGE WIDTH ASF
18	EDGE COLOUR ASF
19	EDGE VISIBILITY ASF
20	LINE BUNDLE INDEX
21	LINE TYPE
22	LINE WIDTH
23	LINE COLOUR
24	LINE CLIPPING MODE
25	MARKER BUNDLE INDEX
26	MARKER TYPE
27	MARKER SIZE
28	MARKER COLOUR
29	MARKER CLIPPING MODE
30	TEXT BUNDLE INDEX
31	TEXT FONT INDEX
32	TEXT COLOUR
33	CHARACTER EXPANSION FACTOR
34	CHARACTER SPACING
35	CHARACTER HEIGHT
36	CHARACTER ORIENTATION
37	TEXT PRECISION
38	TEXT PATH
39	TEXT ALIGNMENT
40	FILL BUNDLE INDEX
41	INTERIOR STYLE
42	FILL COLOUR
43	HATCH INDEX
44	PATTERN INDEX
45	FILL BITMAP

Table 6 – Inheritance filter selection list enumerated values – (concluded)

Value	Filter selection list enumerated
46	EDGE BUNDLE INDEX
47	EDGE TYPE
48	EDGE WIDTH
49	EDGE COLOUR
50	EDGE VISIBILITY
51	EDGE CLIPPING MODE
52	FILL REFERENCE POINT
53	PATTERN SIZE
54	AUXILIARY COLOUR
55	TRANSPARENCY
56	DRAWING MODE
57	PICK IDENTIFIER
58	LINE ASFS
59	MARKER ASFS
60	TEXT ASFS
61	FILL ASFS
62	EDGE ASFS
63	ALL ASFS
64	LINE ATTRIBUTES
65	MARKER ATTRIBUTES
66	LOCAL TEXT ATTRIBUTES
67	GLOBAL TEXT ATTRIBUTES
68	FILL ATTRIBUTES
69	EDGE ATTRIBUTES
70	PATTERN ATTRIBUTES
71	OUTPUT CONTROL
72	PICK ATTRIBUTES
73	ALL ATTRIBUTES
74	ALL

6.1.5 Encoding point lists

Most point list representations do not explicitly contain the number of points in the list. The number of points is implicitly known from the current VDC precision value and the parameter length value.

The exception is when the point list is not the last parameter of the function, as in the case of the GENERALIZED DRAWING PRIMITIVE function. In this special case, the point list is represented by the number of points in the list explicitly encoded as parameter type I followed immediately by the actual points themselves. Explicitly encoding the number of points is required to determine where the coordinate data ends and the next parameter begins.

6.1.6 Encoding transformation matrix components

The functions SEGMENT TRANSFORMATION and COPY SEGMENT have as their second *In* parameter, a transformation consisting of a scaling and rotation portion (2x2R) and a translation portion (2x1VDC). In the Binary Encoding, the transformation will be expressed as a 3x2 matrix of the form:

P2	P3	P6
P4	P5	P7

where: P2 through P7 are the second through seventh encoded parameter values for these functions
and
P2, type R, is the x scale component
P3, type R, is the x rotation component
P4, type R, is the y rotation component
P5, type R, is the y scale component
P6, type VDC, is the x translation component
P7, type VDC, is the y translation component

6.1.7 Encoding local colour precision

The functions CELL ARRAY, PATTERN TABLE and PIXEL ARRAY which include a colour specifiers list *In* parameter, also include a local colour precision requirement *In* parameter. This parameter value specifies the client's precision requirement for the colour specifiers thus enabling the compaction of the colour specifiers within the data stream. The Binary Encoding of this local colour precision requirement is an exception to the general encoding rules (see 4.3) because it is not encoded into the data stream. The client's requirement value expresses the minimum number of bits the client requires in the data stream to specify a colour index or the components of an RGB triple for each colour specifier, according to the COLOUR SELECTION MODE in effect. The CGI Generator of the Binary Encoding uses the client's requirement value to decide independently which local colour precision will satisfy the client's needs. The Generator then encodes this value into the data stream in the place of the client-specified local colour precision requirement parameter.

A local colour precision requirement *In* parameter is also specified in the soliciting functions INQUIRE PATTERN DIMENSIONS and GET PIXEL ARRAY DIMENSIONS. In these functions, the local colour precision requirement parameter value is encoded into the data stream as specified by the client. The value specifies the client's precision requirement on which to base the returned *Out* local colour precision parameter value.

6.1.7.1 Encoding CELL ARRAY and PATTERN TABLE local colour precision

The local colour precision is encoded into these function representations as a value of data type I (integer) in place of the client-specified local colour requirement parameter.

The local colour precision may be encoded as one of the following values:
1, 2, 4, 8, 16, 24 or 32 bits
or 0 to represent the current CI or CD precision.

If the current COLOUR SELECTION MODE is DIRECT, the integer local colour precision value specifies the precision in which each of the RGB components of the colour specifiers list is encoded. If the value is zero, then the COLOUR PRECISION indicates the precision for the RGB components.

If the current COLOUR SELECTION MODE is INDEXED, the integer local colour precision value specifies the precision in which the colour indices of the colour specifiers list are encoded. If the value is zero, then the COLOUR PRECISION indicates the precision for the colour indices.

The specification of the local colour precision parameter for CELL ARRAY and PATTERN TABLE is:

In	local colour precision	(0,1,2,4,8,16,24, or 32)	I

6.1.7.2 Encoding PIXEL ARRAY local colour precision

The local colour precision is encoded into this function representation as a triple of data type I (integer) in place of the client-specified local colour precision requirement parameter.

Each integer in the local colour precision triple may be encoded as one of the following values:
1 to 32 bits
or 0 to represent the current CI or CD precision.

If the current COLOUR SELECTION MODE is DIRECT, the three integer local colour precision values specify the precisions in which each of the RGB components of the colour specifiers list are encoded respectively. If any of the values is zero, then the COLOUR PRECISION indicates the precision for the corresponding RGB components.

If the current COLOUR SELECTION MODE is INDEXED, three integer local colour precision values will be encoded but only the first integer value will specify the precision in which the colour indices of the colour specifiers list are encoded. If the first integer value is zero, then the COLOUR INDEX PRECISION indicates the precision for the colour indices.

The specification of the local colour precision parameter for PIXEL ARRAY:

In	local colour precision	(0,1-32)	3I

6.1.8 Colour specifier lists

A colour specifier list is encoded in the Binary Encoding data stream as a structured data type consisting of a coding representation mode indicator followed by a compacted list of colour specifiers.

A colour specifier list shall be used to represent a parameter which is an array or list of colour specifiers or input colour specifiers in most, but not all, relevant CGI functions.

This form of colour specifier list encoding shall therefore be used for the following functions and parameters:

Function	Parameter
CELL ARRAY	cell colour specifiers
PUT CURRENT RASTER MEASURE	list of input colour values
REQUEST RASTER	list of input colour values
SAMPLE RASTER	list of input colour values
ECHO REQUEST RASTER	list of input colour values
DEQUEUE RASTER EVENT	list of input colour values
PIXEL ARRAY	colour specifiers
GET PIXEL ARRAY	colour specifiers

A raster input measure within an event input report shall also be encoded as a colour specifier list.

This form of colour specifier list encoding is not used for the following functions and parameters:

Function	Parameter
COLOUR TABLE	colour list
PATTERN TABLE	pattern colour specifiers
INQUIRE PATTERN	list of colour specifiers
INQUIRE LIST OF COLOUR TABLE ENTRIES	list of direct colour specifiers

The colour list parameter of COLOUR TABLE and the list of direct colour specifiers parameter of INQUIRE LIST OF COLOUR TABLE ENTRIES are encoded as lists of direct colour values according to the general encoding rules with no compaction of the colour data. Refer to 6.1.9 for the special rules for encoding the pattern colour specifiers in the functions PATTERN TABLE and INQUIRE PATTERN TABLE.

6.1.8.1 Encoding colour specifier lists

The Binary Encoding of the colour specifier lists allows for the compaction of large colour specifier lists. This is accomplished in one of two possible modes at the local colour precision (see 6.1.7) for either indexed or direct colour, according to the COLOUR SELECTION MODE in effect. Two different coding representation modes are supported, packed list mode or run length list mode. The representation mode is encoded into the data stream as a value of data type E (enumerated). It follows the local colour precision parameter (see 6.1.7) and precedes the actual compacted colour specifier values in the data stream. The representation mode may take one of two values:

0 run length list mode
1 packed list mode

For packed list mode, the colour values are represented by rows of colour specifiers packed together without padding. There may, however, be padding at the end of a row since each row is word-aligned, that is starts on a word (16-bit) boundary. No row length information is stored since all rows are the same length. The colour data thus occupies $2n_y([[(p)(n_x)-1)/16+1])$ octets, where n_x is the number of colours per row, n_y is the number of rows, p is the total number of bits per colour, and $[x]$ denotes "the greatest integer in x".

For run length list mode, the data for each row begins on a word (16-bit) boundary and consists of run-length-lists for runs of constant colour value. Each run-length-list consists of two components: a count of type I (Integer) of the number of consecutive constant colours and the representation of that single colour. With the exception of the first run-length-list of a row, the integer count immediately follows the colour representation of the preceding run-length-list with no intervening padding.

The specification of the coding representation mode is:

In	coding representation mode	(RUN_LENGTH, PACKED)	E

This parameter immediately precedes the colour specifiers parameter announcing how the colour specifiers are encoded.

These rules apply to the encoding of colour specifiers list parameters of CELL ARRAY, PIXEL ARRAY and INQUIRE PIXEL ARRAY functions as defined above with no modifications.

6.1.8.2 Encoding lists of input colour values and local colour precision

Lists of input colour values of data type ICO obtain their precision in a special manner. These parameters of RASTER input functions are encoded in one of two coding representation modes: packed list mode or run length list mode, in a manner identical to the encoding rules for colour specifier lists. (See 6.1.8.1.) The difference is in the determination of the local colour precision. That is, the colour specifiers are encoded in a local colour precision which depends on both the Colour and Bits Per Colour entries in the Raster LID State List. Specifically, the local colour precision = Bits Per Colour. It will be encoded into the data stream as data type integer (I) directly preceding the input colour specifiers. When Colour = NO, each ICO is represented by Bits Per Colour number of bits. In this case, Bits Per Colour specifies the number of bits of grey-scale information in the input colour specifier. When Colour = YES, each ICO is represented by three times Bits Per Colour number of bits. In this second case, Bits Per Colour specifies the number of bits of each red, green, blue color component in the input colour specifier.

The specification of the local colour precision for input colour values parameter is:

In/Out	local colour precision	(>0)	I

6.1.9 Encoding PATTERN TABLE and INQUIRE PATTERN colour specifiers

The pattern colour specifier parameters of the PATTERN TABLE and INQUIRE PATTERN functions are encoded in the local colour precision into a packed list. The packing of the colour specifiers extends across rows. That is, there is no word alignment at the beginning of each row.

The colour specifier data occupies $[((n_x)(n_y)(p) + 7)/8]$ octets, where n_x is the number of colours per row, n_y is the number of rows, p is the number of bits per colour, and $[x]$ denotes "the greatest integer in x".

7 Representation of each function and response

7.1 Opcode assignments

Each function and response representation has an assigned opcode which consists of a class code and an id code.

7.1.1 Class code assignments

CGI function and response representations are grouped according to their class; there are currently 256 classes allocated. These classes are grouped into five different categories: non-soliciting function classes, soliciting function classes, response classes, private classes and classes reserved for future standardization. Refer to annex A for class code assignment rules.

Table 7 - List of function/response class codes

Non-soliciting function classes:

Class	Type of representations
0	Virtual Device Management Functions
1	Virtual Device Control Functions
2	Picture Descriptor Functions
3	Control and General Attribute Functions
4	Graphical Primitive Functions
5	Attribute Functions
6	ESCAPE
7	External Functions
8	Segment Functions
9	Input Control Functions
10	Raster Functions
11	<reserved>
12	<reserved>
13	<reserved>
14	<reserved>
15	<reserved for extender code>
16-31	<reserved>

Soliciting function classes:

Class	Type of representations
32	Description Table Inquiry Functions
33	State List Inquiry Functions
34	Non-Inquiry Soliciting Functions
35-46	<reserved>
47	GET ESCAPE

Response classes:

Class	Type of representations
48	Responses for Description Table Inquiry Functions
49	Responses for State List Inquiry Functions
50	Responses for Non-Inquiry Soliciting Functions
51-62	<reserved>
63	GET ESCAPE Response

Private Class:

Class	Type of representations
96	Private Functions

Classes 64 and greater are reserved for future allocation.

7.1.2 Function id code assignments

Each function and response representation has an assigned id within its assigned class. All soliciting functions have the same id assignment as their associated response. Whenever possible, the opcodes list is arranged in the order of the CGI functions defined in ISO/IEC 9636-2, ISO/IEC 9636-3, ISO/IEC 9636-4, ISO/IEC 9636-5, and ISO/IEC 9636-6. Binary encoding opcodes have already been assigned to CGM and GKS metafile elements. CGI functions share these opcodes only when a CGI function is functionally equivalent to a CGM element or GKS metafile element. In these cases, the CGI functional representation encoding will be identical to the CGM or GKS element encoding. It should be noted that the CGI Binary Encoding specific function CLIENT SPECIFIED NAME PRECISION is identical in syntax, semantics and opcode assignment to the CGM element NAME PRECISION. In addition, the opcode assignment for CGI function DRAW ALL SEGMENTS is identical to the GKS metafile element REDRAW ALL SEGMENTS.

Table 8 - List of function/response representation opcodes (Class and id)

Virtual Device Management Functions: Class 0

Class code	Id code	Function name
0	1	<reserved>
0	2	<reserved>
0	3	<reserved>
0	4	<reserved>
0	5	<reserved>
0	6	<reserved>
0	7	<reserved>
0	8	BEGIN FIGURE
0	9	END FIGURE
0	10	<reserved>
0	11	<reserved>
0	12	<reserved>
0	13	<reserved>
0	14	<reserved>
0	15	<reserved>
0	16	<reserved>
0	17	<reserved>
0	18	<reserved>
0	19	<reserved>
0	20	<reserved>
0	21	INITIALIZE
0	22	TERMINATE
0	23	EXECUTE DEFERRED ACTIONS
0	24	DEFERRAL MODE
0	25	PREPARE DRAWING SURFACE
0	26	END PAGE

Table 8 - List of function/response representation opcodes - (continued)

Virtual Device Control Functions: Class 1

Class code	Id code	Function name
1	1	<reserved>
1	2	<reserved>
1	3	VDC TYPE
1	4	INTEGER PRECISION (*)
1	5	REAL PRECISION (*)
1	6	INDEX PRECISION (*)
1	7	COLOUR PRECISION (*)
1	8	COLOUR INDEX PRECISION (*)
1	9	<reserved>
1	10	COLOUR VALUE EXTENT
1	11	<reserved>
1	12	<reserved>
1	13	FONT LIST
1	14	CHARACTER SET LIST
1	15	CHARACTER CODING ANNOUNCER
1	16	CLIENT SPECIFIED NAME PRECISION (*)
1	17	<reserved>
1	18	<reserved>
1	19	<reserved>
1	20	<reserved>
1	21	<reserved>
1	22	<reserved>
1	23	<reserved>
1	24	<reserved>
1	25	<reserved>
1	26	<reserved>
1	27	<reserved>
1	28	<reserved>
1	29	<reserved>
1	30	INTEGER PRECISION REQUIREMENT
1	31	REAL PRECISION REQUIREMENTS
1	32	INDEX PRECISION REQUIREMENT
1	33	COLOUR PRECISION REQUIREMENT
1	34	COLOUR INDEX PRECISION REQUIREMENT
1	35	CLIENT SPECIFIED NAME PRECISION REQUIREMENT
1	36	ERROR HANDLING CONTROL

(*) - Binary Encoding specific function not defined in ISO/IEC 9636.

Table 8 – List of function/response representation opcodes – (continued)

Picture Descriptor Functions: Class 2

Class code	Id code	Function name
2	1	<reserved>
2	2	COLOUR SELECTION MODE
2	3	LINE WIDTH SPECIFICATION MODE
2	4	MARKER SIZE SPECIFICATION MODE
2	5	EDGE WIDTH SPECIFICATION MODE
2	6	VDC EXTENT
2	7	<reserved>
2	8	DEVICE VIEWPORT
2	9	DEVICE VIEWPORT SPECIFICATION MODE
2	10	DEVICE VIEWPORT MAPPING
2	11	LINE REPRESENTATION
2	12	MARKER REPRESENTATION
2	13	TEXT REPRESENTATION
2	14	<reserved>
2	15	<reserved>
2	16	<reserved>
2	17	<reserved>
2	18	<reserved>
2	19	<reserved>
2	20	FILL REPRESENTATION
2	21	EDGE REPRESENTATION
2	22	DELETE BUNDLE REPRESENTATION

Table 8 - List of function/response representation opcodes - (continued)

Control and General Attribute Functions: Class 3

Class code	Id code	Function name
3	1	VDC INTEGER PRECISION
3	2	VDC REAL PRECISION
3	3	AUXILIARY COLOUR
3	4	TRANSPARENCY
3	5	CLIP RECTANGLE
3	6	CLIP INDICATOR
3	7	LINE CLIPPING MODE
3	8	MARKER CLIPPING MODE
3	9	EDGE CLIPPING MODE
3	10	NEW REGION
3	11	<reserved>
3	12	<reserved>
3	13	<reserved>
3	14	<reserved>
3	15	<reserved>
3	16	<reserved>
3	17	<reserved>
3	18	<reserved>
3	19	<reserved>
3	20	<reserved>
3	21	<reserved>
3	22	<reserved>
3	23	<reserved>
3	24	<reserved>
3	25	<reserved>
3	26	DRAWING SURFACE CLIP RECTANGLE
3	27	DRAWING SURFACE CLIP INDICATOR
3	28	VDC INTEGER PRECISION REQUIREMENT
3	29	VDC REAL PRECISION REQUIREMENTS
3	30	STATE LIST INQUIRY SOURCE
3	31	BACKGROUND COLOUR
3	32	SAVE PRIMITIVE ATTRIBUTES
3	33	RESTORE PRIMITIVE ATTRIBUTES
3	34	DELETE PRIMITIVE ATTRIBUTE SAVE SET
3	35	DELETE PATTERN

Table 8 - List of function/response representation opcodes - (continued)

Graphical Primitive Functions: Class 4

Class code	Id code	Function name
4	1	POLYLINE
4	2	DISJOINT POLYLINE
4	3	POLYMARKER
4	4	TEXT
4	5	RESTRICTED TEXT
4	6	APPEND TEXT
4	7	POLYGON
4	8	POLYGON SET
4	9	CELL ARRAY
4	10	GENERALIZED DRAWING PRIMITIVE
4	11	RECTANGLE
4	12	CIRCLE
4	13	CIRCULAR ARC 3 POINT
4	14	CIRCULAR ARC 3 POINT CLOSE
4	15	CIRCULAR ARC CENTRE
4	16	CIRCULAR ARC CENTRE CLOSE
4	17	ELLIPSE
4	18	ELLIPTICAL ARC
4	19	ELLIPTICAL ARC CLOSE
4	20	CIRCULAR ARC CENTRE REVERSED
4	21	CONNECTING EDGE

Table 8 - List of function/response representation opcodes - (continued)

Attribute Functions: Class 5

Class code	Id code	Function name
5	1	LINE BUNDLE INDEX
5	2	LINE TYPE
5	3	LINE WIDTH
5	4	LINE COLOUR
5	5	MARKER BUNDLE INDEX
5	6	MARKER TYPE
5	7	MARKER SIZE
5	8	MARKER COLOUR
5	9	TEXT BUNDLE INDEX
5	10	TEXT FONT INDEX
5	11	TEXT PRECISION
5	12	CHARACTER EXPANSION FACTOR
5	13	CHARACTER SPACING
5	14	TEXT COLOUR
5	15	CHARACTER HEIGHT
5	16	CHARACTER ORIENTATION
5	17	TEXT PATH
5	18	TEXT ALIGNMENT
5	19	CHARACTER SET INDEX
5	20	ALTERNATE CHARACTER SET INDEX
5	21	FILL BUNDLE INDEX
5	22	INTERIOR STYLE
5	23	FILL COLOUR
5	24	HATCH INDEX
5	25	PATTERN INDEX
5	26	EDGE BUNDLE INDEX
5	27	EDGE TYPE
5	28	EDGE WIDTH
5	29	EDGE COLOUR
5	30	EDGE VISIBILITY
5	31	FILL REFERENCE POINT
5	32	PATTERN TABLE
5	33	PATTERN SIZE
5	34	COLOUR TABLE
5	35	ASPECT SOURCE FLAGS
5	36	PICK IDENTIFIER

Table 8 – List of function/response representation opcodes – (continued)

ESCAPE Function: Class 6

Class code	Id code	Function name
6	1	ESCAPE

External Functions: Class 7

Class code	Id code	Function name
7	1	MESSAGE
7	2	<reserved>

Segment Functions: Class 8

Class code	Id code	Function name
8	1	COPY SEGMENT
8	2	INHERITANCE FILTER
8	3	CLIPPING INHERITANCE
8	4	SEGMENT TRANSFORMATION
8	5	SEGMENT HIGHLIGHTING
8	6	SEGMENT DISPLAY PRIORITY
8	7	SEGMENT PICK PRIORITY
8	8	DELETE SEGMENT
8	9	RENAME SEGMENT
8	10	<reserved>
8	11	SEGMENT VISIBILITY
8	12	SEGMENT DETECTABILITY
8	13	<reserved>
8	14	<reserved>
8	15	<reserved>
8	16	<reserved>
8	17	<reserved>
8	18	CREATE SEGMENT
8	19	REOPEN SEGMENT
8	20	CLOSE SEGMENT
8	21	DELETE ALL SEGMENTS
8	22	DRAW ALL SEGMENTS
8	23	IMPLICIT SEGMENT REGENERATION MODE
8	24	RESET REGENERATION PENDING

Table 8 - List of function/response representation opcodes - (continued)

Input Control Functions: Class 9

Class code	Id code	Function name
9	1	INITIALIZE LOGICAL INPUT DEVICE
9	2	RELEASE LOGICAL INPUT DEVICE
9	3	ECHO CONTROLS
9	4	PUT CURRENT LOCATOR MEASURE
9	5	PUT CURRENT STROKE MEASURE
9	6	PUT CURRENT VALUATOR MEASURE
9	7	PUT CURRENT CHOICE MEASURE
9	8	PUT CURRENT PICK MEASURE
9	9	PUT CURRENT STRING MEASURE
9	10	PUT CURRENT RASTER MEASURE
9	11	PUT CURRENT GENERAL MEASURE
9	12	ECHO DATA
9	13	LOCATOR DEVICE DATA
9	14	STROKE DEVICE DATA
9	15	VALUATOR DEVICE DATA
9	16	CHOICE DEVICE DATA
9	17	PICK DEVICE DATA
9	18	STRING DEVICE DATA
9	19	RASTER DEVICE DATA
9	20	GENERAL DEVICE DATA
9	21	ASSOCIATE TRIGGERS
9	22	SAMPLING STATE
9	23	INITIALIZE ECHO REQUEST
9	24	RELEASE EVENT QUEUE
9	25	ENABLE EVENTS
9	26	DISABLE EVENTS
9	27	EVENT QUEUE BLOCK CONTROL
9	28	FLUSH EVENTS
9	29	FLUSH DEVICE EVENTS
9	30	INITIALIZE ECHO OUTPUT
9	31	RELEASE ECHO OUTPUT
9	32	ECHO OUTPUT CONTROLS
9	33	PERFORM ACKNOWLEDGEMENT
9	34	UPDATE LOCATOR ECHO OUTPUT
9	35	UPDATE STROKE ECHO OUTPUT
9	36	UPDATE VALUATOR ECHO OUTPUT
9	37	UPDATE CHOICE ECHO OUTPUT
9	38	UPDATE PICK ECHO OUTPUT
9	39	UPDATE STRING ECHO OUTPUT
9	40	UPDATE RASTER ECHO OUTPUT
9	41	UPDATE GENERAL ECHO OUTPUT
9	42	ECHO OUTPUT DATA

Table 8 – List of function/response representation opcodes – (continued)

Raster Functions: Class 10

Class code	Id code	Function name
10	1	CREATE BITMAP
10	2	DELETE BITMAP
10	3	DRAWING BITMAP
10	4	DISPLAY BITMAP
10	5	MAPPED BITMAP FOREGROUND COLOUR
10	6	MAPPED BITMAP BACKGROUND COLOUR
10	7	TRANSPARENT COLOUR
10	8	DRAWING MODE
10	9	FILL BITMAP
10	10	PIXEL ARRAY
10	11	SOURCE DESTINATION BITBLT
10	12	TILE THREE OPERAND BITBLT

Table 8 - List of function/response representation opcodes - (continued)

Description Table Inquiry Functions: Class 32

Class code	Id code	Function name
32	1	INQUIRE DEVICE IDENTIFICATION
32	2	INQUIRE DEVICE DESCRIPTION
32	3	LOOKUP FUNCTION SUPPORT
32	4	LOOKUP PROFILE SUPPORT
32	5	INQUIRE LIST OF PROFILE SUPPORT INDICATORS
32	6	INQUIRE SUPPORTED VDC TYPES
32	7	INQUIRE DEVICE CONTROL CAPABILITY
32	8	LOOKUP ESCAPE SUPPORT
32	9	LOOKUP GET ESCAPE SUPPORT
32	10	INQUIRE PRIMITIVE SUPPORT LEVELS
32	11	LOOKUP GDP SUPPORT
32	12	INQUIRE GDP ATTRIBUTES
32	13	INQUIRE LINE CAPABILITY
32	14	INQUIRE LIST OF AVAILABLE LINE TYPES
32	15	INQUIRE LIST OF AVAILABLE SCALED LINE WIDTHS
32	16	INQUIRE MARKER CAPABILITY
32	17	INQUIRE LIST OF AVAILABLE MARKER TYPES
32	18	INQUIRE LIST OF AVAILABLE SCALED MARKER SIZES
32	19	INQUIRE TEXT CAPABILITY
32	20	INQUIRE LIST OF AVAILABLE CHARACTER SETS
32	21	INQUIRE LIST OF AVAILABLE TEXT FONTS
32	22	INQUIRE FONT CAPABILITIES
32	23	INQUIRE LIST OF AVAILABLE CHARACTER EXPANSION FACTORS
32	24	INQUIRE LIST OF AVAILABLE CHARACTER SPACINGS
32	25	INQUIRE LIST OF AVAILABLE CHARACTER HEIGHTS
32	26	INQUIRE LIST OF AVAILABLE CHARACTER ORIENTATIONS
32	27	INQUIRE FILL CAPABILITY
32	28	INQUIRE LIST OF AVAILABLE HATCH STYLES
32	29	INQUIRE EDGE CAPABILITY
32	30	INQUIRE LIST OF AVAILABLE EDGE TYPES
32	31	INQUIRE LIST OF AVAILABLE SCALED EDGE WIDTHS
32	32	INQUIRE COLOUR CAPABILITY
32	33	INQUIRE CIE CHARACTERISTICS
32	34	INQUIRE MAXIMUM NUMBER OF SIMULTANEOUSLY SAVED ATTRIBUTE SETS
32	35	INQUIRE ARRAY OF SUPPORTED CHARACTER CODING ANNOUNCERS

Table 8 – List of function/response representation opcodes – (continued)

Description Table Inquiry Functions: Class 32 – (concluded)

Class code	Id code	Function name
32	36	INQUIRE SEGMENT CAPABILITY
32	37	INQUIRE INPUT CAPABILITY
32	38	INQUIRE LIST OF AVAILABLE INPUT DEVICES
32	39	INQUIRE COMMON INPUT DEVICE PROPERTIES
32	40	INQUIRE LIST OF SUPPORTED ECHO TYPES
32	41	INQUIRE LIST OF SUPPORTED PROMPT TYPES
32	42	INQUIRE LIST OF SUPPORTED ACKNOWLEDGEMENT TYPES
32	43	INQUIRE LIST OF ASSOCIABLE TRIGGERS
32	44	INQUIRE LOCATOR CAPABILITIES
32	45	INQUIRE STROKE CAPABILITIES
32	46	INQUIRE CHOICE CAPABILITIES
32	47	INQUIRE PICK CAPABILITIES
32	48	INQUIRE STRING CAPABILITIES
32	49	INQUIRE LIST OF AVAILABLE INPUT CHARACTER SETS
32	50	INQUIRE RASTER INPUT CAPABILITIES
32	51	INQUIRE LIST OF PERMITTED RASTER SPOT CENTRE SEPARATIONS
32	52	INQUIRE GENERAL CAPABILITIES
32	53	INQUIRE LIST OF SUPPORTED GENERAL MEASURE FORMATS
32	54	INQUIRE ECHO OUTPUT CAPABILITIES
32	55	INQUIRE LIST OF ECHO OUTPUT ECHO TYPES
32	56	INQUIRE LIST OF ECHO OUTPUT PROMPT TYPES
32	57	INQUIRE LIST OF ECHO OUTPUT ACKNOWLEDGEMENT TYPES
32	58	INQUIRE LIST OF SUPPORTED GENERAL FORMAT IDENTIFIERS
32	59	INQUIRE RASTER CAPABILITY
32	60	INQUIRE LIST OF SUPPORTED DRAWING-MODE/TRANSPARENCY PAIRS
32	61	INQUIRE LIST OF SUPPORTED DRAWING-MODE-3/TRANSPARENCY PAIRS

Table 8 - List of function/response representation opcodes - (continued)

State List Inquiry Functions: Class 33

Class code	Id code	Function name
33	1	INQUIRE CONTROL STATE
33	2	INQUIRE CURRENT PRECISION REQUIREMENTS
33	3	INQUIRE VDC TO DEVICE MAPPING
33	4	INQUIRE ERROR HANDLING
33	5	INQUIRE MISCELLANEOUS CONTROL STATE
33	6	INQUIRE LINE ATTRIBUTES
33	7	INQUIRE LIST OF LINE BUNDLE INDICES
33	8	INQUIRE LINE REPRESENTATION
33	9	INQUIRE MARKER ATTRIBUTES
33	10	INQUIRE LIST OF MARKER BUNDLE INDICES
33	11	INQUIRE MARKER REPRESENTATION
33	12	INQUIRE TEXT ATTRIBUTES
33	13	INQUIRE LIST OF TEXT BUNDLE INDICES
33	14	INQUIRE TEXT REPRESENTATION
33	15	INQUIRE FILL ATTRIBUTES
33	16	INQUIRE PATTERN DIMENSIONS
33	17	INQUIRE PATTERN
33	18	INQUIRE LIST OF PATTERN INDICES
33	19	INQUIRE LIST OF FILL BUNDLE INDICES
33	20	INQUIRE FILL REPRESENTATION
33	21	INQUIRE EDGE ATTRIBUTES
33	22	INQUIRE LIST OF EDGE BUNDLE INDICES
33	23	INQUIRE EDGE REPRESENTATION
33	24	INQUIRE OUTPUT STATE
33	25	INQUIRE OBJECT CLIPPING
33	26	INQUIRE LIST OF ATTRIBUTE SET NAMES IN USE
33	27	INQUIRE COLOUR STATE
33	28	INQUIRE LIST OF COLOUR TABLE ENTRIES
33	29	INQUIRE FONT LIST
33	30	INQUIRE CHARACTER SET LIST
33	31	LOOKUP ASPECT SOURCE FLAGS
33	32	INQUIRE SEGMENT STATE
33	33	INQUIRE LIST OF INHERITANCE FILTER SETTINGS
33	34	INQUIRE CLIPPING INHERITANCE
33	35	INQUIRE LIST OF SEGMENT INDENTIFIERS IN USE
33	36	INQUIRE INDIVIDUAL SEGMENT STATE

Table 8 - List of function/response representation opcodes - (continued)

State List Inquiry Functions: Class 33 - (concluded)

Class code	Id code	Function name
33	37	INQUIRE COMMON LOGICAL INPUT DEVICE STATE
33	38	INQUIRE LIST OF ASSOCIATED TRIGGERS
33	39	INQUIRE ECHO DATA RECORD
33	40	INQUIRE INPUT DEVICE DATA RECORD
33	41	INQUIRE LOCATOR STATE
33	42	INQUIRE STROKE STATE
33	43	INQUIRE VALUATOR STATE
33	44	INQUIRE CHOICE STATE
33	45	INQUIRE PICK STATE
33	46	INQUIRE STRING STATE
33	47	INQUIRE RASTER INPUT STATE
33	48	INQUIRE GENERAL STATE
33	49	INQUIRE EVENT INPUT STATE
33	50	INQUIRE LIST OF CURRENTLY EXISTING ECHO ENTITIES
33	51	INQUIRE ECHO ENTITY STATE
33	52	INQUIRE ECHO OUTPUT DATA RECORD
33	53	INQUIRE RASTER STATE
33	54	INQUIRE LIST OF NON-DISPLAYABLE BITMAP IDENTIFIERS
33	55	INQUIRE LIST OF DISPLAYABLE BITMAP IDENTIFIERS
33	56	INQUIRE BITMAP STATE

Table 8 - List of function/response representation opcodes - (continued)

Non-Inquiry Soliciting Functions: Class 34 - (concluded)

Class code	Id code	Function name
34	1	DEQUEUE ERROR REPORTS
34	2	GET TEXT EXTENT
34	3	GET NEW SEGMENT IDENTIFIER
34	4	SIMULATE PICK
34	5	GET ADDITIONAL STROKE DATA
34	6	GET ADDITIONAL PICK DATA
34	7	GET ADDITIONAL STRING DATA
34	8	GET ADDITIONAL RASTER DATA
34	9	REQUEST LOCATOR
34	10	REQUEST STROKE
34	11	REQUEST VALUATOR
34	12	REQUEST CHOICE
34	13	REQUEST PICK
34	14	REQUEST STRING
34	15	REQUEST RASTER
34	16	REQUEST GENERAL
34	17	SAMPLE LOCATOR
34	18	SAMPLE STROKE
34	19	SAMPLE VALUATOR
34	20	SAMPLE CHOICE
34	21	SAMPLE PICK
34	22	SAMPLE STRING
34	23	SAMPLE RASTER
34	24	SAMPLE GENERAL
34	25	ECHO REQUEST LOCATOR
34	26	ECHO REQUEST STROKE
34	27	ECHO REQUEST VALUATOR
34	28	ECHO REQUEST CHOICE
34	29	ECHO REQUEST PICK
34	30	ECHO REQUEST STRING
34	31	ECHO REQUEST RASTER
34	32	ECHO REQUEST GENERAL
34	33	INITIALIZE EVENT QUEUE
34	34	AWAIT EVENT

Table 8 - List of function/response representation opcodes - (continued)

Non-Inquiry Soliciting Functions: Class 34 - (concluded)

Class code	Id code	Function name
34	35	DEQUEUE LOCATOR EVENT
34	36	DEQUEUE STROKE EVENT
34	37	DEQUEUE VALUATOR EVENT
34	38	DEQUEUE CHOICE EVENT
34	39	DEQUEUE PICK EVENT
34	40	DEQUEUE STRING EVENT
34	41	DEQUEUE RASTER EVENT
34	42	DEQUEUE GENERAL EVENT
34	43	EVENT QUEUE TRANSFER
34	44	GET NEW BITMAP IDENTIFIER
34	45	GET PIXEL ARRAY
34	46	GET PIXEL ARRAY DIMENSIONS

GET ESCAPE Function: Class 47

Class code	Id code	Function name
47	1	GET ESCAPE

Table 8 - List of function/response representation opcodes - (continued)

Responses for Description Table Inquiry Functions: Class 48

Class code	Id code	Function name
48	1	Response for INQUIRE DEVICE IDENTIFICATION
48	2	Response for INQUIRE DEVICE DESCRIPTION
48	3	Response for LOOKUP FUNCTION SUPPORT
48	4	Response for LOOKUP PROFILE SUPPORT
48	5	Response for INQUIRE LIST OF PROFILE SUPPORT INDICATORS
48	6	Response for INQUIRE SUPPORTED VDC TYPES
48	7	Response for INQUIRE DEVICE CONTROL CAPABILITY
48	8	Response for LOOKUP ESCAPE SUPPORT
48	9	Response for LOOKUP GET ESCAPE SUPPORT
48	10	Response for INQUIRE PRIMITIVE SUPPORT LEVELS
48	11	Response for LOOKUP GDP SUPPORT
48	12	Response for INQUIRE GDP ATTRIBUTES
48	13	Response for INQUIRE LINE CAPABILITY
48	14	Response for INQUIRE LIST OF AVAILABLE LINE TYPES
48	15	Response for INQUIRE LIST OF AVAILABLE SCALED LINE WIDTHS
48	16	Response for INQUIRE MARKER CAPABILITY
48	17	Response for INQUIRE LIST OF AVAILABLE MARKER TYPES
48	18	Response for INQUIRE LIST OF AVAILABLE SCALED MARKER SIZES
48	19	Response for INQUIRE TEXT CAPABILITY
48	20	Response for INQUIRE LIST OF AVAILABLE CHARACTER SETS
48	21	Response for INQUIRE LIST OF AVAILABLE TEXT FONTS
48	22	Response for INQUIRE FONT CAPABILITIES
48	23	Response for INQUIRE LIST OF AVAILABLE CHARACTER EXPANSION FACTORS
48	24	Response for INQUIRE LIST OF AVAILABLE CHARACTER SPACINGS
48	25	Response for INQUIRE LIST OF AVAILABLE CHARACTER HEIGHTS
48	26	Response for INQUIRE LIST OF AVAILABLE CHARACTER ORIENTATIONS
48	27	Response for INQUIRE FILL CAPABILITY
48	28	Response for INQUIRE LIST OF AVAILABLE HATCH STYLES
48	29	Response for INQUIRE EDGE CAPABILITY
48	30	Response for INQUIRE LIST OF AVAILABLE EDGE TYPES

Table 8 - List of function/response representation opcodes - (continued)

Responses for Description Table Inquiry Functions: Class 48 - (concluded)

Class code	Id code	Function name
48	31	Response for INQUIRE LIST OF AVAILABLE SCALED EDGE WIDTHS
48	32	Response for INQUIRE COLOUR CAPABILITY
48	33	Response for INQUIRE CIE CHARACTERISTICS
48	34	Response for INQUIRE MAXIMUM NUMBER OF SIMULTANEOUSLY SAVED ATTRIBUTE SETS
48	35	Response for INQUIRE ARRAY OF SUPPORTED CHARACTER CODING ANNOUNCERS
48	36	Response for INQUIRE SEGMENT CAPABILITY
48	37	Response for INQUIRE INPUT CAPABILITY
48	38	Response for INQUIRE LIST OF AVAILABLE INPUT DEVICES
48	39	Response for INQUIRE COMMON INPUT DEVICE PROPERTIES
48	40	Response for INQUIRE LIST OF SUPPORTED ECHO TYPES
48	41	Response for INQUIRE LIST OF SUPPORTED PROMPT TYPES
48	42	Response for INQUIRE LIST OF SUPPORTED ACKNOWLEDGEMENT TYPES
48	43	Response for INQUIRE LIST OF ASSOCIABLE TRIGGERS
48	44	Response for INQUIRE LOCATOR CAPABILITIES
48	45	Response for INQUIRE STROKE CAPABILITIES
48	46	Response for INQUIRE CHOICE CAPABILITIES
48	47	Response for INQUIRE PICK CAPABILITIES
48	48	Response for INQUIRE STRING CAPABILITIES
48	49	Response for INQUIRE LIST OF AVAILABLE INPUT CHARACTER SETS
48	50	Response for INQUIRE RASTER INPUT CAPABILITIES
48	51	Response for INQUIRE LIST OF PERMITTED RASTER SPOT CENTRE SEPARATIONS
48	52	Response for INQUIRE GENERAL CAPABILITIES
48	53	Response for INQUIRE LIST OF SUPPORTED GENERAL MEASURE FORMATS
48	54	Response for INQUIRE ECHO OUTPUT CAPABILITIES
48	55	Response for INQUIRE LIST OF ECHO OUTPUT ECHO TYPES
48	56	Response for INQUIRE LIST OF ECHO OUTPUT PROMPT TYPES
48	57	Response for INQUIRE LIST OF ECHO OUTPUT ACKNOWLEDGEMENT TYPES
48	58	Response for INQUIRE LIST OF SUPPORTED GENERAL FORMAT IDENTIFIERS
48	59	Response for INQUIRE RASTER CAPABILITY
48	60	Response for INQUIRE LIST OF SUPPORTED DRAWING-MODE/TRANSPARENCY PAIRS
48	61	Response for INQUIRE LIST OF SUPPORTED DRAWING-MODE-3/TRANSPARENCY PAIRS

Table 8 - List of function/response representation opcodes - (continued)

Responses for State List Inquiry Functions: Class 49

Class code	Id code	Function name
49	1	Response for INQUIRE CONTROL STATE
49	2	Response for INQUIRE CURRENT PRECISION REQUIREMENTS
49	3	Response for INQUIRE VDC TO DEVICE MAPPING
49	4	Response for INQUIRE ERROR HANDLING
49	5	Response for INQUIRE MISCELLANEOUS CONTROL STATE
49	6	Response for INQUIRE LINE ATTRIBUTES
49	7	Response for INQUIRE LIST OF LINE BUNDLE INDICES
49	8	Response for INQUIRE LINE REPRESENTATION
49	9	Response for INQUIRE MARKER ATTRIBUTES
49	10	Response for INQUIRE LIST OF MARKER BUNDLE INDICES
49	11	Response for INQUIRE MARKER REPRESENTATION
49	12	Response for INQUIRE TEXT ATTRIBUTES
49	13	Response for INQUIRE LIST OF TEXT BUNDLE INDICES
49	14	Response for INQUIRE TEXT REPRESENTATION
49	15	Response for INQUIRE FILL ATTRIBUTES
49	16	Response for INQUIRE PATTERN DIMENSIONS
49	17	Response for INQUIRE PATTERN
49	18	Response for INQUIRE LIST OF PATTERN INDICES
49	19	Response for INQUIRE LIST OF FILL BUNDLE INDICES
49	20	Response for INQUIRE FILL REPRESENTATION
49	21	Response for INQUIRE EDGE ATTRIBUTES
49	22	Response for INQUIRE LIST OF EDGE BUNDLE INDICES
49	23	Response for INQUIRE EDGE REPRESENTATION
49	24	Response for INQUIRE OUTPUT STATE
49	25	Response for INQUIRE OBJECT CLIPPING
49	26	Response for INQUIRE LIST OF ATTRIBUTE SET NAMES IN USE
49	27	Response for INQUIRE COLOUR STATE
49	28	Response for INQUIRE LIST OF COLOUR TABLE ENTRIES
49	29	Response for INQUIRE FONT LIST
49	30	Response for INQUIRE CHARACTER SET LIST
49	31	Response for LOOKUP ASPECT SOURCE FLAGS
49	32	Response for INQUIRE SEGMENT STATE
49	33	Response for INQUIRE LIST OF INHERITANCE FILTER SETTINGS
49	34	Response for INQUIRE CLIPPING INHERITANCE
49	35	Response for INQUIRE LIST OF SEGMENT IDENTIFIERS IN USE
49	36	Response for INQUIRE INDIVIDUAL SEGMENT STATE