INTERNATIONAL STANDARD

ISO/IEC 9637-1

First edition
1994-03-01

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

Part 1:
Character encoding

Technologies de l'information — Infographie — Techniques interfaciales de dialogues avec dispositifs graphiques (CGI) — Liaison de courant D —

Partie 1: Codage des caractères

Reference number
ISO/IEC 9637-1:1994(E)

Contents

Foreword iv

Introduction v
0.1 Purpose of the character encoding v
0.2 Objectives v
0.3 Relationship to other International Standards v

1 Scope 1

2 Normative references 2

3 Notational conventions 3
3.1 7-Bit and 8-Bit code tables 3
3.2 Code extension techniques vocabulary 4
3.2.1 C0 sets 4
3.2.2 C1 sets 4
3.2.3 G sets 4

4 Overall structure 6
4.1 General form of the data stream 6
4.2 Entering and leaving the CGI environment 6
4.2.1 Implicitly entering the CGI environment 6
4.2.2 Designating and invoking the CGI coding environment from ISO 2022 6
4.3 Encoding functions 6

5 Method of encoding opcodes 8
5.1 Encoding technique of the basic opcode set 8
5.2 Extension mechanism 9
5.3 Opcode assignments 9
5.4 Opcodes for soliciting functions 9

6 Method of encoding parameters 20
6.1 Basic format 21
6.2 Bitstream format 21
6.3 Coding integers (I, IF8, IF16, IF32) 22
6.4 Coding real numbers (R) 23
6.5 Coding Virtual Device Coordinates and Points (VDC, P) 25
6.6 Coding point list parameters (nP) 26
6.6.1 Displacement mode 26
6.6.2 Incremental mode 26
6.6.3 Incremental mode encoding 31
6.7 Colour specifiers (CI, CD) 31
6.8 Colour lists (nCI, nCD) 32
6.8.1 Normal format (coding type=0) 33
6.8.2 Bitstream format (coding type=1) 33
6.8.3 Runlength format (coding type=2) 34
6.8.4 Runlength bitstream format (coding type=3) 34

6.8.5 Examples 34
6.9 String parameters (S, SF) 36
6.9.1 Overall string parameter format 36
6.9.2 Bit combinations permitted within string parameters of text functions 36
6.9.3 C0 control within string parameters 37
6.9.4 Using G-sets in string parameters 37
6.9.4.1 String parameters and character sets 37
6.9.4.2 String parameters of other CGI functions 38
6.10 Enumerated parameters (E) 38
6.11 Index parameters (IX) 39
6.12 Client specified name parameters (CSN) 39
6.13 Input colour parameters (ICO, nICO) 40
6.14 Data record parameters (D) 40

7 Character substitution 42

8 Representation of CGI functions 44
8.1 Soliciting functions 44
8.2 Enumerated parameters 45
8.3 Functions not specified in ISO/IEC 9636 45
8.3.1 DOMAIN RING 45
8.3.2 INTEGER PRECISION 45
8.3.3 REAL PRECISION 46
8.3.4 INDEX PRECISION 46
8.3.5 COLOUR PRECISION 47
8.3.6 COLOUR INDEX PRECISION 47
8.3.7 VDC INTEGER PRECISION 47
8.3.8 VDC REAL PRECISION 47
8.3.9 CLIENT SPECIFIED NAME PRECISION 48
8.4 Encoding of matrices 48

9 Defaults 50

10 Classification and designation 51

Annex A: List of CGI Opcodes 52

Annex B: List of escape sequences 63

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 9637-1 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Sub-Committee SC 24, *Computer graphics and image processing*, in collaboration with the European Computer Manufacturers Association (ECMA) and the European Conference of Postal and Telecommunications Administration (CEPT).

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*

— *Part 3: Clear text encoding*

Annexes A and B of this part of ISO/IEC 9637 are for information only.

Introduction

0.1 Purpose of the character encoding

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

If minimizing the processing overhead is more important than data compaction, an encoding such as the binary encoding contained in ISO/IEC 9637-2 may be more appropriate.

0.2 Objectives

This encoding was designed with the following objectives:

a) *regular syntax:* All CGI functions are encoded in a uniform way so that parsing the encoded data is simple;

b) *compactness:* the encoding provides a highly compact data stream, suitable for systems with restricted storage capacity or transfer bandwidth;

c) *extensibility:* the encoding allows for future extensions;

d) *transportability:* the encoding is suitable for use with transport mechanisms designed for character-oriented data based on a standard national character set derived from ISO/IEC 646.

0.3 Relationship to other International Standards

The character encoding has been developed in collaboration with the ISO/EIC JTC1/SC2. The encoding conforms to the rules for code extension specified in ISO 2022 in the category of complete coding system.

The representation of character data in this part of ISO/IEC 9637 follows the rules of ISO/IEC 646 and ISO 2022.

For certain functions, ISO/IEC 9636 defines value ranges as being reserved for registration. The values and their meanings will be defined using the established registration procedures (see ISO/IEC 9636-1).

This encoding is based on that for the Computer Graphics Metafile ISO/IEC 8632-2. Wherever possible, the opcodes and parameter representations of CGM elements have been followed.

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

Part 1:
Character encoding

1 Scope

This part of ISO/IEC 9637 specifies a character encoding of the Computer Graphics Interface. For each of the functions specified in ISO/IEC 9636 an encoding is specified.

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

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 revision, and parties to agreements based to 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 ISO and IEC 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 — Coded extension techniques.*
ISO 2375:1985,	*Data processing — Procedure for registration of escape sequences.*
ISO/IEC 6429:1992,	*Information technology — Control functions for coded character sets.*
ISO 7942:1985,	*Information processing systems — Computer graphics — Graphical Kernel System (GKS) functional description.*
ISO/IEC 8632-2:1992,	*Information technology — Computer graphics — Metafile for the storage and transfer of picture description information — Part 2 : Character 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 9637-2:1992,	*Information technology — Computer graphics — Interfacing techniques for dialogues with graphical devices (CGI) — Data stream binding — Part 2: Binary encoding.*
ISO/IEC TR 9973:1988,	*Information processing — Procedures for Registration of Graphical Items.*
ECMA 96,	*Graphics Data Syntax for a multiple Workstation Interface.*
CEPT,	*Rev. of T/CD 6.1 Videotex Presentation Layer Data Syntax.*

3 Notational conventions

3.1 7-Bit and 8-Bit code tables

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

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

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

In this part of ISO/IEC 9637 the bit combinations of a 7-bit code are identified by notation of the form *x*/*y*, where *x* is a number in the range 0 to 7 and *y* is a number in the range 0 to 15. The correspondence between the notations of the form *x*/*y* and the bit combinations consisting of the bits b7 to b1 is as follows:

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

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

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

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

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

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

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

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

In this part of ISO/IEC 9637 the bit combinations of an 8-bit code are identified by notation of the form *xx*/*yy*, where *xx* and *yy* are numbers in the range 00 to 15. The correspondence between the notations of the form *xx*/*yy* and the bit combinations consisting of the bits b8 to b1 is as follows:

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

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

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

3.2 Code extension techniques vocabulary

In describing the characters that may occur within string parameters, certain terms imported from other standards (e.g., ISO 2022) are useful. In the context of the CGI, these terms, and the concepts to which they refer, apply only within the string parameters of the functions listed in 6.9.4.1.

3.2.1 C0 sets

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

3.2.2 C1 sets

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

3.2.3 G sets

The G-sets (G0, G1, G2, G3) are coded character sets of 94 or 96 characters. CHARACTER SET INDEX designates which character set is to be the G0 set. ALTERNATE CHARACTER SET INDEX designates a character set to be used as both the G1 and G2 sets. The G-sets may be "invoked into" (caused to occupy) columns 2 through 7 of a 7-bit code table, or columns 02 through 07 and 10 through 15 of an 8-bit code table. This encoding of the CGI uses the G0 and G1/G2 sets within string parameters. The G3 set may be used within the string parameters of a conforming CGI data stream; this requires selection of the extended 7-bit or extended 8-bit mode in the CHARACTER CODING ANNOUNCER. ISO/IEC 9636 does not provide a function to explicitly designate the G3 sets; this may be done within a text string in accordance with ISO 2022, or by other means agreed upon by the interchanging parties.

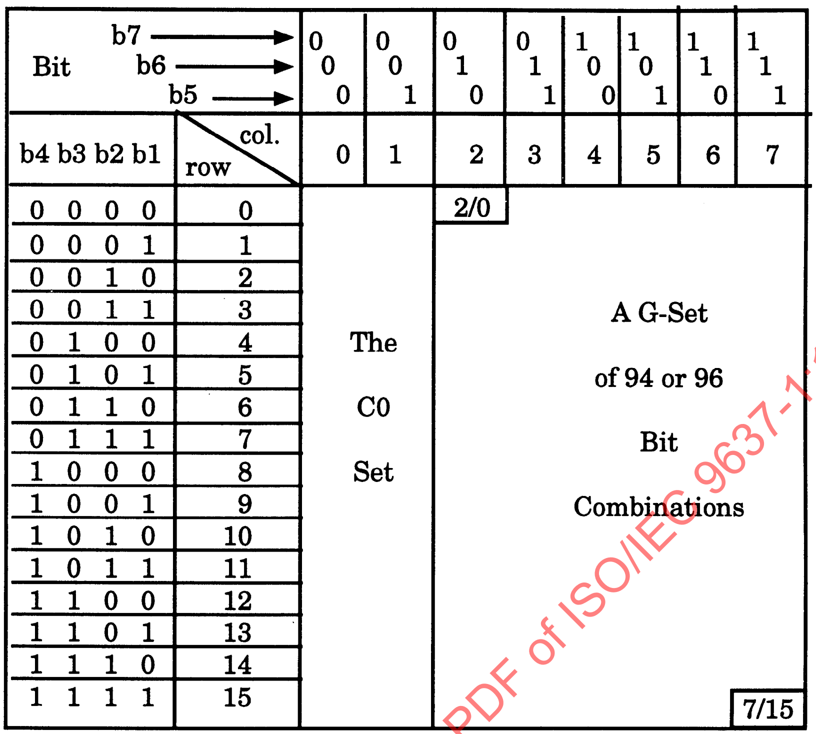


Figure 1 — The 7-bit code table.

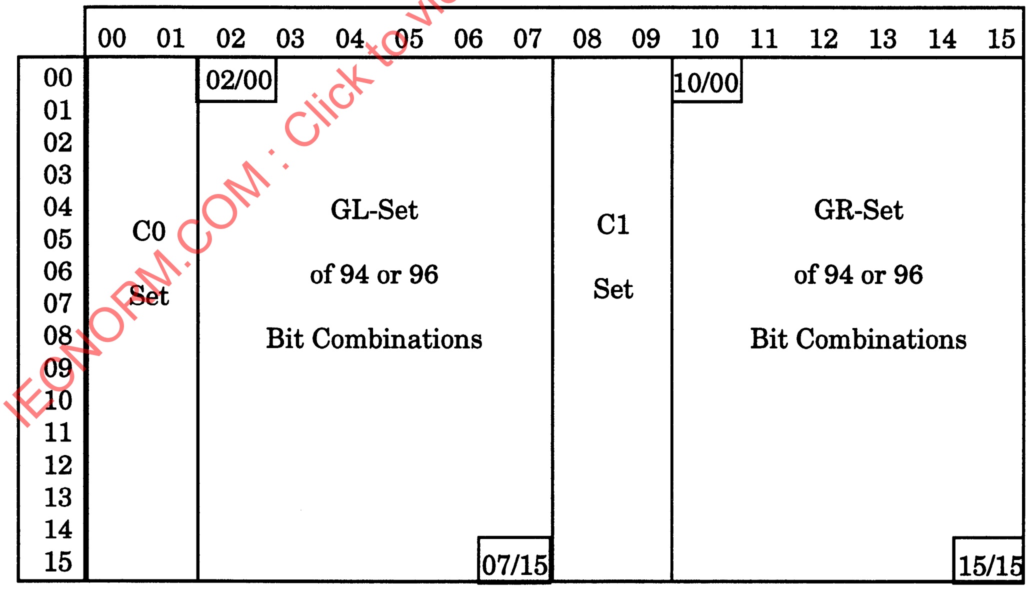


Figure 2 — The 8-bit code table.

4 Overall structure

This encoding specifies representations for each of the CGI functions of ISO/IEC 9636 as well as any associated response. 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 designating the particular function representation, and the parameter data (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 designating the particular response representation, and its return parameter data.

4.2 Entering and leaving the CGI environment

4.2.1 Implicitly entering the CGI environment

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

4.2.2 Designating and invoking the CGI coding environment from ISO 2022

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

ESC 2/5 *F*

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

The first bit combination occurring after this escape sequence will then represent the opcode of a CGI function.

After the end of the CGI data (i.e., after the TERMINATE function) the following escape sequence may be used to return to the ISO 2022 coding environment:

ESC 2/5 4/0

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

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.

The character encoding scheme has two basic components: the opcode and the parameter data. 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 the opcode 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 an opcode followed by the *Out* parameter data (see 5.4).

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

5 Method of encoding opcodes

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

- the basic opcode set;
- extension opcode sets.

Parameter bytes are coded in columns 4 to 7 (with the exception of strings, which are otherwise delimited). This scheme permits an interpreter to skip the parameters of an function representation which it does not recognize or support. The interpreter can in such circumstances skip bytes until another opcode is encountered.

5.1 Encoding technique of the basic opcode set

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

b8		b1
X	0 1 0	b b b b

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

b8		b1
X	0 1 1	b b b b

b8		b1
X	0 1	b b b b b

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

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

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

5.2 Extension mechanism

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

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

Opcode format	Extension codes	Basic opcode set codes
1	$\underbrace{\text{<EOS>...<EOS>}}_{n \text{ instances}}$	<2/*x*>
2	$\underbrace{\text{<EOS>...<EOS>}}_{n \text{ instances}}$	<3/*y* 2/*z*>
3	$\underbrace{\text{<EOS>...<EOS>}}_{n \text{ instances}}$	<3/y 3/z>

<EOS> = 3/15
n = 0 selects the basic opcode set,
n = 1 selects the first extension opcode set,
n = *N* selects the *N–th* extension opcode set

x = 0,1,..,15
y = 0,1,..,14
z = 0,1,..,15

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

5.3 Opcode assignments

CGI opcodes overlap with opcodes from CGM and GKS. Where the same function, with identical parameters and identical semantics exists in two of them, then the same opcode (and parameterisation) is used.

Whereas all CGM and GKS opcodes are one and two-byte opcodes from the basic opcode set, some CGI opcodes extend into the first extension opcode set. These are three-byte opcodes, commencing with the byte 3/15.

5.4 Opcodes for soliciting functions

CGI functions which have *Out* (i.e. return) parameters have distinct opcodes in this encoding for the outward data, sent from generator to interpreter, and the return data, or 'response', returned from the interpreter.

There is a simple relationship between the outward opcode and the response opcode of a soliciting function. For all soliciting functions with an outward opcode *x*/*k* *a*/*b*, the response opcode will be of the form *x* / *k*+2 *a* / *b*. For inquiry functions, which are in the first extension set, the outward opcode is 3/15 *x* / *k* *a* / *b* and the response opcode is 3/15 *x* / *k*+2 *a* / *b*.

NOTE 1 The allocation of opcodes to CGI functions is rather scattered due to the need for consistency with opcodes from CGM and GKS, ISO/IEC 8632-2 and ISO 7942.

Table 1 lists the complete set of 7-bit opcode assignments for CGI functions and responses. To determine the 8-bit opcode, a zero should be added in front of the column specification (e.g., 03/0 02/10 instead of 3/0 2/10).

Table 1 — Opcodes and Response Opcodes of CGI Functions

Function	Opcode	Response Opcode
INITIALIZE	3/0 3/2	
TERMINATE	3/0 3/3	
EXECUTE DEFERRED ACTIONS	3/0 3/6	
DEFERRAL MODE	3/0 3/7	
PREPARE DRAWING SURFACE	3/0 3/4	
END PAGE	3/0 3/5	
VDC TYPE	3/1 2/2	
VDC INTEGER PRECISION (*)	3/3 2/0	
VDC INTEGER PRECISION REQUIREMENT	3/3 3/6	
VDC REAL PRECISION (*)	3/3 2/1	
VDC REAL PRECISION REQUIREMENTS	3/3 3/7	
VDC EXTENT	3/2 2/5	
DEVICE VIEWPORT	3/2 2/7	
DEVICE VIEWPORT SPECIFICATION MODE	3/2 2/8	
DEVICE VIEWPORT MAPPING	3/2 2/9	
DRAWING SURFACE CLIP RECTANGLE	3/3 3/4	
DRAWING SURFACE CLIP INDICATOR	3/3 3/5	
DEQUEUE ERROR REPORTS	3/11 2/0	3/13 2/0
ERROR HANDLING CONTROL	3/1 3/14	
INTEGER PRECISION (*)	3/1 2/3	
INTEGER PRECISION REQUIREMENT	3/1 3/8	
REAL PRECISION (*)	3/1 2/4	
REAL PRECISION REQUIREMENTS	3/1 3/9	
INDEX PRECISION (*)	3/1 2/5	
INDEX PRECISION REQUIREMENT	3/1 3/10	
COLOUR PRECISION (*)	3/1 2/6	
COLOUR PRECISION REQUIREMENT	3/1 3/11	
COLOUR INDEX PRECISION (*)	3/1 2/7	
COLOUR INDEX PRECISION REQUIREMENT	3/1 3/12	
CLIENT SPECIFIED NAME PRECISION (*)	3/1 3/0	
CLIENT SPECIFIED NAME PRECISION REQUIREMENT	3/1 3/13	
MESSAGE	3/7 2/1	
ESCAPE	3/7 2/0	
GET ESCAPE	3/12 2/14	3/14 2/14
STATE LIST INQUIRY SOURCE	3/3 3/8	
INQUIRE DEVICE IDENTIFICATION	3/15 3/0 2/0	3/15 3/2 2/0
INQUIRE DEVICE DESCRIPTION	3/15 3/0 2/1	3/15 3/2 2/1
LOOKUP FUNCTION SUPPORT	3/15 3/0 2/2	3/15 3/2 2/2
LOOKUP PROFILE SUPPORT	3/15 3/0 2/3	3/15 3/2 2/3
INQUIRE LIST OF PROFILE SUPPORT INDICATORS	3/15 3/0 2/4	3/15 3/2 2/4
INQUIRE SUPPORTED VDC TYPES	3/15 3/0 2/5	3/15 3/2 2/5
INQUIRE DEVICE CONTROL CAPABILITY	3/15 3/0 2/6	3/15 3/2 2/6

Table 1 *(continued)*

Function	Opcode	Response Opcode
LOOKUP ESCAPE SUPPORT	3/15 3/0 2/7	3/15 3/2 2/7
LOOKUP GET ESCAPE SUPPORT	3/15 3/0 2/8	3/15 3/2 2/8
INQUIRE CONTROL STATE	3/15 3/4 2/0	3/15 3/6 2/0
INQUIRE CURRENT PRECISIONS REQUIREMENTS	3/15 3/4 2/1	3/15 3/6 2/1
INQUIRE VDC TO DEVICE MAPPING	3/15 3/4 2/2	3/15 3/6 2/2
INQUIRE ERROR HANDLING	3/15 3/4 2/3	3/15 3/6 2/3
INQUIRE MISCELLANEOUS CONTROL STATE	3/15 3/4 2/4	3/15 3/6 2/4
POLYLINE	2/0	
DISJOINT POLYLINE	2/1	
CIRCULAR ARC 3 POINT	3/4 2/1	
CIRCULAR ARC CENTRE	3/4 2/3	
CIRCULAR ARC CENTRE REVERSED	3/4 2/8	
ELLIPTICAL ARC	3/4 2/6	
CONNECTING EDGE	3/4 2/9	
POLYMARKER	2/2	
TEXT	2/3	
RESTRICTED TEXT	2/4	
APPEND TEXT	2/5	
POLYGON	2/6	
POLYGON SET	2/7	
RECTANGLE	2/10	
CIRCLE	3/4 2/0	
CIRCULAR ARC 3 POINT CLOSE	3/4 2/2	
CIRCULAR ARC CENTRE CLOSE	3/4 2/4	
ELLIPSE	3/4 2/5	
ELLIPTICAL ARC CLOSE	3/4 2/7	
CELL ARRAY	2/8	
GENERALIZED DRAWING PRIMITIVE (GDP)	2/9	
LINE BUNDLE INDEX	3/5 2/0	
LINE TYPE	3/5 2/1	
LINE WIDTH	3/5 2/2	
LINE COLOUR	3/5 2/3	
LINE CLIPPING MODE	3/3 2/6	
MARKER BUNDLE INDEX	3/5 2/4	
MARKER TYPE	3/5 2/5	
MARKER SIZE	3/5 2/6	
MARKER COLOUR	3/5 2/7	
MARKER CLIPPING MODE	3/3 2/7	
TEXT BUNDLE INDEX	3/5 3/0	
TEXT FONT INDEX	3/5 3/1	
TEXT PRECISION	3/5 3/2	
CHARACTER EXPANSION FACTOR	3/5 3/3	
CHARACTER SPACING	3/5 3/4	

Table 1 *(continued)*

Function	Opcode	Response Opcode
TEXT COLOUR	3/5 3/5	
CHARACTER HEIGHT	3/5 3/6	
CHARACTER ORIENTATION	3/5 3/7	
TEXT PATH	3/5 3/8	
TEXT ALIGNMENT	3/5 3/9	
CHARACTER SET INDEX	3/5 3/10	
ALTERNATE CHARACTER SET INDEX	3/5 3/11	
CHARACTER CODING ANNOUNCER	3/1 2/15	
FILL BUNDLE INDEX	3/6 2/0	
INTERIOR STYLE	3/6 2/1	
FILL COLOUR	3/6 2/2	
HATCH INDEX	3/6 2/3	
PATTERN INDEX	3/6 2/4	
FILL REFERENCE POINT	3/6 2/10	
PATTERN SIZE	3/6 2/12	
EDGE BUNDLE INDEX	3/6 2/5	
EDGE TYPE	3/6 2/6	
EDGE WIDTH	3/6 2/7	
EDGE COLOUR	3/6 2/8	
EDGE CLIPPING MODE	3/3 2/8	
EDGE VISIBILITY	3/6 2/9	
CLIP INDICATOR	3/3 2/5	
CLIP RECTANGLE	3/3 2/4	
LINE WIDTH SPECIFICATION MODE	3/2 2/2	
MARKER SIZE SPECIFICATION MODE	3/2 2/3	
EDGE WIDTH SPECIFICATION MODE	3/2 2/4	
COLOUR SELECTION MODE	3/2 2/1	
COLOUR VALUE EXTENT	3/1 2/9	
BACKGROUND COLOUR	3/3 3/9	
AUXILIARY COLOUR	3/3 2/2	
TRANSPARENCY	3/3 2/3	
COLOUR TABLE	3/6 3/0	
LINE REPRESENTATION	3/2 2/10	
MARKER REPRESENTATION	3/2 2/11	
TEXT REPRESENTATION	3/2 2/12	
FILL REPRESENTATION	3/2 3/0	
EDGE REPRESENTATION	3/2 3/1	
DELETE BUNDLE REPRESENTATION	3/2 3/2	
ASPECT SOURCE FLAGS	3/6 3/1	
PATTERN TABLE	3/6 2/11	
DELETE PATTERN	3/3 3/13	
FONT LIST	3/1 2/13	
CHARACTER SET LIST	3/1 2/14	

Table 1 ***(continued)***

Function	Opcode	Response Opcode
SAVE PRIMITIVE ATTRIBUTES	3/3 3/10	
RESTORE PRIMITIVE ATTRIBUTES	3/3 3/11	
DELETE PRIMITIVE ATTRIBUTE SAVE SET	3/3 3/12	
BEGIN FIGURE	3/0 2/7	
END FIGURE	3/0 2/8	
NEW REGION	3/3 2/9	
GET TEXT EXTENT	3/11 2/1	3/13 2/1
INQUIRE PRIMITIVE SUPPORT LEVELS	3/15 3/0 2/9	3/15 3/2 2/9
LOOKUP GDP SUPPORT	3/15 3/0 2/10	3/15 3/2 2/10
INQUIRE GDP ATTRIBUTES	3/15 3/0 2/11	3/15 3/2 2/11
INQUIRE LINE CAPABILITY	3/15 3/0 2/12	3/15 3/2 2/12
INQUIRE LIST OF AVAILABLE LINE TYPES	3/15 3/0 2/13	3/15 3/2 2/13
INQUIRE LIST OF AVAILABLE SCALED LINE WIDTHS	3/15 3/0 2/14	3/15 3/2 2/14
INQUIRE MARKER CAPABILITY	3/15 3/0 2/15	3/15 3/2 2/15
INQUIRE LIST OF AVAILABLE MARKER TYPES	3/15 3/0 3/0	3/15 3/2 3/0
INQUIRE LIST OF AVAILABLE SCALED MARKER SIZES	3/15 3/0 3/1	3/15 3/2 3/1
INQUIRE TEXT CAPABILITY	3/15 3/0 3/2	3/15 3/2 3/2
INQUIRE LIST OF AVAILABLE CHARACTER SETS	3/15 3/0 3/3	3/15 3/2 3/3
INQUIRE LIST OF AVAILABLE TEXT FONTS	3/15 3/0 3/4	3/15 3/2 3/4
INQUIRE FONT CAPABILITIES	3/15 3/0 3/5	3/15 3/2 3/5
INQUIRE LIST OF AVAILABLE CHARACTER EXPANSION FACTORS	3/15 3/0 3/6	3/15 3/2 3/6
INQUIRE LIST OF AVAILABLE CHARACTER SPACINGS	3/15 3/0 3/7	3/15 3/2 3/7
INQUIRE LIST OF AVAILABLE CHARACTER HEIGHTS	3/15 3/0 3/8	3/15 3/2 3/8
INQUIRE LIST OF AVAILABLE CHARACTER ORIENTATIONS	3/15 3/0 3/9	3/15 3/2 3/9
INQUIRE FILL CAPABILITY	3/15 3/0 3/10	3/15 3/2 3/10
INQUIRE LIST OF AVAILABLE HATCH STYLES	3/15 3/0 3/11	3/15 3/2 3/11
INQUIRE EDGE CAPABILITY	3/15 3/0 3/12	3/15 3/2 3/12
INQUIRE LIST OF AVAILABLE EDGE TYPES	3/15 3/0 3/13	3/15 3/2 3/13
INQUIRE LIST OF AVAILABLE SCALED EDGE WIDTHS	3/15 3/0 3/14	3/15 3/2 3/14
INQUIRE COLOUR CAPABILITY	3/15 3/0 3/15	3/15 3/2 3/15
INQUIRE CIE CHARACTERISTICS	3/15 3/1 2/0	3/15 3/3 2/0
INQUIRE MAXIMUM NUMBER OF SIMULTANEOUSLY SAVED ATTRIBUTE SETS	3/15 3/1 2/1	3/15 3/3 2/1
INQUIRE ARRAY OF SUPPORTED CHARACTER CODING ANNOUNCERS	3/15 3/1 2/2	3/15 3/3 2/2
INQUIRE LINE ATTRIBUTES	3/15 3/4 2/5	3/15 3/6 2/5
INQUIRE LIST OF LINE BUNDLE INDICES	3/15 3/4 2/6	3/15 3/6 2/6

Table 1 *(continued)*

Function	Opcode	Response Opcode
INQUIRE LINE REPRESENTATION	3/15 3/4 2/7	3/15 3/6 2/7
INQUIRE MARKER ATTRIBUTES	3/15 3/4 2/8	3/15 3/6 2/8
INQUIRE LIST OF MARKER BUNDLE INDICES	3/15 3/4 2/9	3/15 3/6 2/9
INQUIRE MARKER REPRESENTATION	3/15 3/4 2/10	3/15 3/6 2/10
INQUIRE TEXT ATTRIBUTES	3/15 3/4 2/11	3/15 3/6 2/11
INQUIRE LIST OF TEXT BUNDLE INDICES	3/15 3/4 2/12	3/15 3/6 2/12
INQUIRE TEXT REPRESENTATION	3/15 3/4 2/13	3/15 3/6 2/13
INQUIRE FILL ATTRIBUTES	3/15 3/4 2/14	3/15 3/6 2/14
INQUIRE PATTERN DIMENSIONS	3/15 3/4 2/15	3/15 3/6 2/15
INQUIRE PATTERN	3/15 3/4 3/0	3/15 3/6 3/0
INQUIRE LIST OF PATTERN INDICES	3/15 3/4 3/1	3/15 3/6 3/1
INQUIRE LIST OF FILL BUNDLE INDICES	3/15 3/4 3/2	3/15 3/6 3/2
INQUIRE FILL REPRESENTATION	3/15 3/4 3/3	3/15 3/6 3/3
INQUIRE EDGE ATTRIBUTES	3/15 3/4 3/4	3/15 3/6 3/4
INQUIRE LIST OF EDGE BUNDLE INDICES	3/15 3/4 3/5	3/15 3/6 3/5
INQUIRE EDGE REPRESENTATION	3/15 3/4 3/6	3/15 3/6 3/6
INQUIRE OUTPUT STATE	3/15 3/4 3/7	3/15 3/6 3/7
INQUIRE OBJECT CLIPPING	3/15 3/4 3/8	3/15 3/6 3/8
INQUIRE LIST OF ATTRIBUTE SET NAMES IN USE	3/15 3/4 3/9	3/15 3/6 3/9
INQUIRE COLOUR STATE	3/15 3/4 3/10	3/15 3/6 3/10
INQUIRE LIST OF COLOUR TABLE ENTRIES	3/15 3/4 3/11	3/15 3/6 3/11
INQUIRE FONT LIST	3/15 3/4 3/12	3/15 3/6 3/12
INQUIRE CHARACTER SET LIST	3/15 3/4 3/13	3/15 3/6 3/13
LOOKUP ASPECT SOURCE FLAGS	3/15 3/4 3/14	3/15 3/6 3/14
GET NEW SEGMENT IDENTIFIER	3/11 2/2	3/13 2/2
CREATE SEGMENT	3/8 2/13	
REOPEN SEGMENT	3/8 2/14	
CLOSE SEGMENT	3/8 2/15	
COPY SEGMENT	3/8 2/0	
DELETE SEGMENT	3/8 2/7	
DELETE ALL SEGMENTS	3/8 3/2	
RENAME SEGMENT	3/8 2/8	
DRAW ALL SEGMENTS	3/8 2/9	
IMPLICIT SEGMENT REGENERATION MODE	3/8 3/0	
RESET REGENERATION PENDING	3/8 3/1	
PICK IDENTIFIER	3/6 3/2	
SEGMENT VISIBILITY	3/8 2/10	
SEGMENT TRANSFORMATION	3/8 2/3	
SEGMENT HIGHLIGHTING	3/8 2/4	
SEGMENT DISPLAY PRIORITY	3/8 2/5	
SEGMENT DETECTABILITY	3/8 2/11	
SEGMENT PICK PRIORITY	3/8 2/6	

Table 1 *(continued)*

Function	Opcode	Response Opcode
SIMULATE PICK	3/11 2/3	3/13 2/3
INHERITANCE FILTER	3/8 2/1	
CLIPPING INHERITANCE	3/8 2/2	
INQUIRE SEGMENT CAPABILITY	3/15 3/1 2/3	3/15 3/3 2/3
INQUIRE SEGMENT STATE	3/15 3/4 3/15	3/15 3/6 3/15
INQUIRE LIST OF INHERITANCE FILTER SETTINGS	3/15 3/5 2/0	3/15 3/7 2/0
INQUIRE CLIPPING INHERITANCE	3/15 3/5 2/1	3/15 3/7 2/1
INQUIRE LIST OF SEGMENT IDENTIFIERS IN USE	3/15 3/5 2/2	3/15 3/7 2/2
INQUIRE INDIVIDUAL SEGMENT STATE	3/15 3/5 2/3	3/15 3/7 2/3
INITIALIZE LOGICAL INPUT DEVICE	3/9 2/0	
RELEASE LOGICAL INPUT DEVICE	3/9 2/1	
ECHO CONTROLS	3/9 2/2	
PUT CURRENT LOCATOR MEASURE	3/9 2/3	
PUT CURRENT STROKE MEASURE	3/9 2/4	
PUT CURRENT VALUATOR MEASURE	3/9 2/5	
PUT CURRENT CHOICE MEASURE	3/9 2/6	
PUT CURRENT PICK MEASURE	3/9 2/7	
PUT CURRENT STRING MEASURE	3/9 2/8	
PUT CURRENT RASTER MEASURE	3/9 2/9	
PUT CURRENT GENERAL MEASURE	3/9 2/10	
ECHO DATA	3/9 2/11	
LOCATOR DEVICE DATA	3/9 2/12	
STROKE DEVICE DATA	3/9 2/13	
VALUATOR DEVICE DATA	3/9 2/14	
CHOICE DEVICE DATA	3/9 2/15	
PICK DEVICE DATA	3/9 3/0	
STRING DEVICE DATA	3/9 3/1	
RASTER DEVICE DATA	3/9 3/2	
GENERAL DEVICE DATA	3/9 3/3	
ASSOCIATE TRIGGERS	3/9 3/4	
GET ADDITIONAL STROKE DATA	3/11 2/4	3/13 2/4
GET ADDITIONAL PICK DATA	3/11 2/5	3/13 2/5
GET ADDITIONAL STRING DATA	3/11 2/6	3/13 2/6
GET ADDITIONAL RASTER DATA	3/11 2/7	3/13 2/7
REQUEST LOCATOR	3/11 2/8	3/13 2/8
REQUEST STROKE	3/11 2/9	3/13 2/9
REQUEST VALUATOR	3/11 2/10	3/13 2/10
REQUEST CHOICE	3/11 2/11	3/13 2/11
REQUEST PICK	3/11 2/12	3/13 2/12
REQUEST STRING	3/11 2/13	3/13 2/13

Table 1 *(continued)*

Function	Opcode	Response Opcode
REQUEST RASTER	3/11 2/14	3/13 2/14
REQUEST GENERAL	3/11 2/15	3/13 2/15
SAMPLING STATE	3/9 3/5	
SAMPLE LOCATOR	3/11 3/0	3/13 3/0
SAMPLE STROKE	3/11 3/1	3/13 3/1
SAMPLE VALUATOR	3/11 3/2	3/13 3/2
SAMPLE CHOICE	3/11 3/3	3/13 3/3
SAMPLE PICK	3/11 3/4	3/13 3/4
SAMPLE STRING	3/11 3/5	3/13 3/5
SAMPLE RASTER	3/11 3/6	3/13 3/6
SAMPLE GENERAL	3/11 3/7	3/13 3/7
INITIALIZE ECHO REQUEST	3/9 3/6	
ECHO REQUEST LOCATOR	3/11 3/8	3/13 3/8
ECHO REQUEST STROKE	3/11 3/9	3/13 3/9
ECHO REQUEST VALUATOR	3/11 3/10	3/13 3/10
ECHO REQUEST CHOICE	3/11 3/11	3/13 3/11
ECHO REQUEST PICK	3/11 3/12	3/13 3/12
ECHO REQUEST STRING	3/11 3/13	3/13 3/13
ECHO REQUEST RASTER	3/11 3/14	3/13 3/14
ECHO REQUEST GENERAL	3/11 3/15	3/13 3/15
INITIALIZE EVENT QUEUE	3/12 2/0	3/14 2/0
RELEASE EVENT QUEUE	3/9 3/7	
ENABLE EVENTS	3/9 3/8	
DISABLE EVENTS	3/9 3/9	
EVENT QUEUE BLOCK CONTROL	3/9 3/10	
FLUSH EVENTS	3/9 3/11	
FLUSH DEVICE EVENTS	3/9 3/12	
AWAIT EVENT	3/12 2/1	3/14 2/1
DEQUEUE LOCATOR EVENT	3/12 2/2	3/14 2/2
DEQUEUE STROKE EVENT	3/12 2/3	3/14 2/3
DEQUEUE VALUATOR EVENT	3/12 2/4	3/14 2/4
DEQUEUE CHOICE EVENT	3/12 2/5	3/14 2/5
DEQUEUE PICK EVENT	3/12 2/6	3/14 2/6
DEQUEUE STRING EVENT	3/12 2/7	3/14 2/7
DEQUEUE RASTER EVENT	3/12 2/8	3/14 2/8
DEQUEUE GENERAL EVENT	3/12 2/9	3/14 2/9
EVENT QUEUE TRANSFER	3/12 2/10	3/14 2/10
INITIALIZE ECHO OUTPUT	3/10 2/0	
RELEASE ECHO OUTPUT	3/10 2/1	
ECHO OUTPUT CONTROLS	3/10 2/2	
ECHO OUTPUT DATA	3/10 2/3	

Table 1 *(continued)*

Function	Opcode	Response Opcode
PERFORM ACKNOWLEDGEMENT	3/10 2/4	
UPDATE LOCATOR ECHO OUTPUT	3/10 2/5	
UPDATE STROKE ECHO OUTPUT	3/10 2/6	
UPDATE VALUATOR ECHO OUTPUT	3/10 2/7	
UPDATE CHOICE ECHO OUTPUT	3/10 2/8	
UPDATE PICK ECHO OUTPUT	3/10 2/9	
UPDATE STRING ECHO OUTPUT	3/10 2/10	
UPDATE RASTER ECHO OUTPUT	3/10 2/11	
UPDATE GENERAL ECHO OUTPUT	3/10 2/12	
INQUIRE INPUT CAPABILITY	3/15 3/1 2/4	3/15 3/3 2/4
INQUIRE LIST OF AVAILABLE INPUT DEVICES	3/15 3/1 2/5	3/15 3/5 2/4
INQUIRE COMMON INPUT DEVICE PROPERTIES	3/15 3/1 2/6	3/15 3/3 2/6
INQUIRE LIST OF SUPPORTED ECHO TYPES	3/15 3/1 2/7	3/15 3/3 2/7
INQUIRE LIST OF SUPPORTED PROMPT TYPES	3/15 3/1 2/8	3/15 3/3 2/8
INQUIRE LIST OF SUPPORTED ACKNOWLEDGEMENT TYPES	3/15 3/1 2/9	3/15 3/3 2/9
INQUIRE LIST OF ASSOCIABLE TRIGGERS	3/15 3/1 2/10	3/15 3/3 2/10
INQUIRE LOCATOR CAPABILITIES	3/15 3/1 2/11	3/15 3/3 2/11
INQUIRE STROKE CAPABILITIES	3/15 3/1 2/12	3/15 3/3 2/12
INQUIRE CHOICE CAPABILITIES	3/15 3/1 2/13	3/15 3/3 2/13
INQUIRE PICK CAPABILITIES	3/15 3/1 2/14	3/15 3/3 2/14
INQUIRE STRING CAPABILITIES	3/15 3/1 2/15	3/15 3/3 2/15
INQUIRE LIST OF AVAILABLE INPUT CHARACTER SETS	3/15 3/1 3/0	3/15 3/3 3/0
INQUIRE RASTER INPUT CAPABILITIES	3/15 3/1 3/1	3/15 3/3 3/1
INQUIRE LIST OF PERMITTED RASTER SPOT CENTRE SEPARATIONS	3/15 3/1 3/2	3/15 3/3 3/2
INQUIRE GENERAL CAPABILITIES	3/15 3/1 3/3	3/15 3/3 3/3
INQUIRE LIST OF SUPPORTED GENERAL MEASURE FORMATS	3/15 3/1 3/4	3/15 3/3 3/4
INQUIRE COMMON LOGICAL INPUT DEVICE STATE	3/15 3/5 2/4	3/15 3/7 2/4
INQUIRE LIST OF ASSOCIATED TRIGGERS	3/15 3/5 2/5	3/15 3/7 2/5
INQUIRE ECHO DATA RECORD	3/15 3/5 2/6	3/15 3/7 2/6
INQUIRE INPUT DEVICE DATA RECORD	3/15 3/5 2/7	3/15 3/7 2/7
INQUIRE LOCATOR STATE	3/15 3/5 2/8	3/15 3/7 2/8
INQUIRE STROKE STATE	3/15 3/5 2/9	3/15 3/7 2/9
INQUIRE VALUATOR STATE	3/15 3/5 2/10	3/15 3/7 2/10
INQUIRE CHOICE STATE	3/15 3/5 2/11	3/15 3/7 2/11
INQUIRE PICK STATE	3/15 3/5 2/12	3/15 3/7 2/12
INQUIRE STRING STATE	3/15 3/5 2/13	3/15 3/7 2/13
INQUIRE RASTER INPUT STATE	3/15 3/5 2/14	3/15 3/7 2/14
INQUIRE GENERAL STATE	3/15 3/5 2/15	3/15 3/7 2/15
INQUIRE EVENT INPUT STATE	3/15 3/5 3/0	3/15 3/7 3/0

Table 1 *(concluded)*

Function	Opcode	Response Opcode
INQUIRE ECHO OUTPUT CAPABILITIES	3/15 3/1 3/5	3/15 3/3 3/5
INQUIRE LIST OF ECHO OUTPUT ECHO TYPES	3/15 3/1 3/6	3/15 3/3 3/6
INQUIRE LIST OF ECHO OUTPUT PROMPT TYPES	3/15 3/1 3/7	3/15 3/3 3/7
INQUIRE LIST OF ECHO OUTPUT ACKNOWLEDGEMENT TYPES	3/15 3/1 3/8	3/15 3/3 3/8
INQUIRE LIST OF SUPPORTED GENERAL FORMAT IDENTIFIERS	3/15 3/1 3/9	3/15 3/3 3/9
INQUIRE LIST OF CURRENTLY EXISTING ECHO ENTITIES	3/15 3/5 3/1	3/15 3/7 3/1
INQUIRE ECHO ENTITY STATE	3/15 3/5 3/2	3/15 3/7 3/2
INQUIRE ECHO OUTPUT DATA RECORD	3/15 3/5 3/3	3/15 3/7 3/3
GET NEW BITMAP IDENTIFIER	3/12 2/11	3/14 2/11
CREATE BITMAP	3/10 3/0	
DELETE BITMAP	3/10 3/1	
DRAWING BITMAP	3/10 3/2	
DISPLAY BITMAP	3/10 3/3	
MAPPED BITMAP FOREGROUND COLOUR	3/10 3/4	
MAPPED BITMAP BACKGROUND COLOUR	3/10 3/5	
TRANSPARENT COLOUR	3/10 3/6	
DRAWING MODE	3/10 3/7	
FILL BITMAP	3/10 3/8	
PIXEL ARRAY	3/10 3/9	
GET PIXEL ARRAY	3/12 2/12	3/14 2/12
GET PIXEL ARRAY DIMENSIONS	3/12 2/13	3/14 2/13
SOURCE DESTINATION BITBLT	3/10 3/10	
TILE THREE OPERAND BITBLT	3/10 3/11	
INQUIRE RASTER CAPABILITY	3/15 3/1 3/10	3/15 3/3 3/10
INQUIRE LIST OF SUPPORTED DRAWING-MODE/TRANSPARENCY PAIRS	3/15 3/1 3/11	3/15 3/3 3/11
INQUIRE LIST OF SUPPORTED DRAWING-MODE-3/TRANSPARENCY PAIRS	3/15 3/1 3/12	3/15 3/3 3/12
INQUIRE RASTER STATE	3/15 3/5 3/4	3/15 3/7 3/4
INQUIRE LIST OF NON-DISPLAYABLE BITMAP IDENTIFIERS	3/15 3/5 3/5	3/15 3/7 3/5
INQUIRE LIST OF DISPLAYABLE BI TMAP IDENTIFIERS	3/15 3/5 3/6	3/15 3/7 3/6
INQUIRE BITMAP STATE	3/15 3/5 3/7	3/15 3/7 3/7
DOMAIN RING (*)	3/7 3/0	
LIST TERMINATOR (*)	3/7 3/1	

(*) Encoding-specific function not defined in ISO/IEC 9636 (CGI functional specification)

6 Method of encoding parameters

The parameter part of a CGI function representation or response representation may contain one or more parameters, each parameter consisting of one or more bytes.

Subclause 5.2.10 of ISO/IEC 9636-1 (CGI functional specification) provides a list of the abstract data types employed in the specification of CGI functions, along with a mapping to basic data types. The following subclauses describe the encoding of parameters of these basic data types and in addition for the composite types P, nP, nCD, nCI and nICO.

CD	colour direct value	IX	index
CI	colour index	R	real
CSN	client specified name	S, SF	string
D	data record	VDC	virtual device coordinate
E	enumerated	P	virtual device point
I	integer	nP	list of virtual device points
ICO	input colour specifier	nCD	list of colour direct values
IF8	fixed 8-bit integer	nCI	list of colour indices
IF16	fixed 16-bit integer	nICO	list of input colour specifiers
IF32	fixed 32-bit integer		

The abstract data types device coordinate (DC), intrinsic name (IN), input surface coordinate (ISC) and the errors lost count within an error report (ER) are represented by fixed 16-bit integers (IF16) in this encoding.

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

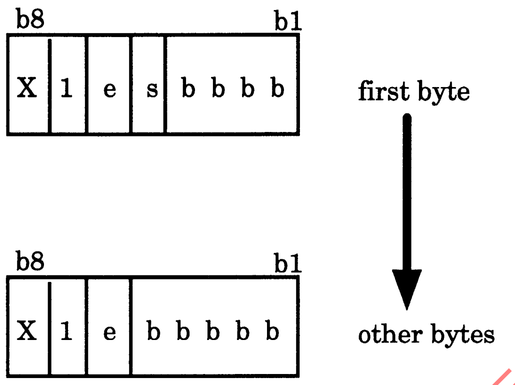
b8		b1
X	1	b b b b b b

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

Except for *string* parameters all parameters are encoded using one or both of two formats, basic format or bitstream format.

6.1 Basic format

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



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

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

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

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

The basic format is used to encode

a) enumerated types (E, see 6.10);

b) colour indices (CI, see 6.7 and 6.8);

c) indices other than colour indices (IX, see 6.11);

d) integers (I, see 6.3);

e) real numbers (R, see 6.4);

f) non-incremental coordinates (VDC, P, nP, see 6.5 and 6.6).

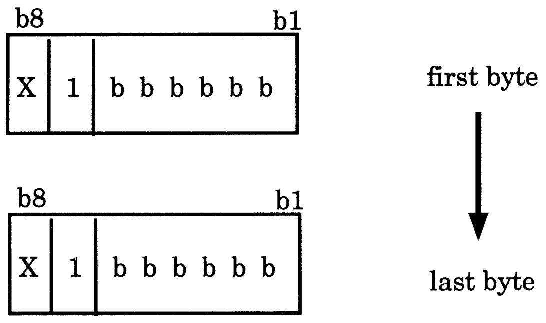
g) client specified names (CSN, see 6.12)

h) fixed integers (IF8, IF16 and IF32, see 6.3)

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

6.2 Bitstream format

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



The *X* is the parity bit (or omitted bit) in a 7-bit environment. In an 8-bit environment it is 0. Bit b7 is the parameter flag. Bits b6 through b1 are the data bits of the parameter.

The bitstream format is used to encode

a) incremental mode point lists (nP, see 6.6.2);

b) colour direct specifiers and lists (CD, nCD, see 6.8);

c) colour index lists (nCI, see 6.8).

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

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

- for an incremental mode point list, the end of the data (which identifies the end of the bitstream format parameter) is identified by the <End of Block> code;
- for a colour index list, the number of bits needed to encode each element of the colour index list shall be determined from the current COLOUR INDEX PRECISION, or by the LOCAL COLOUR PRECISION parameter (for those functions which contain such a parameter). The number of elements in the list of colour indices shall be determined from another parameter (or parameters) of the function, and thus the total size of the bitstream may be calculated. For functions which do not make explicit the length of the list by some parameter, then the end of the bitstream shall be defined by the next opcode byte.
- for a list of colour direct specifiers, the number of bits needed to encode each colour direct specifier shall be determined by the current value of COLOUR PRECISION, or by the LOCAL COLOUR PRECISION parameter (for those functions which contain such a parameter). The number of elements in the list of colour specifiers shall be determined from another parameter (or parameters) of the function, and thus the total size of the bitstream may be calculated. For functions which do not make explicit the length of the list by some parameter (e.g. COLOUR TABLE), then the end of the bitstream shall be defined by the next opcode byte.

6.3 Coding integers (I, IF8, IF16, IF32)

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

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

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

The size of integer parameters (I) is limited by the current INTEGER PRECISION value. Fixed integers (IF8, IF16, IF32) are coded with at most 8, 16, or 32 data bits respectively.

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

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

Larger integers require more bytes.

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

6.4 Coding real numbers (R)

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

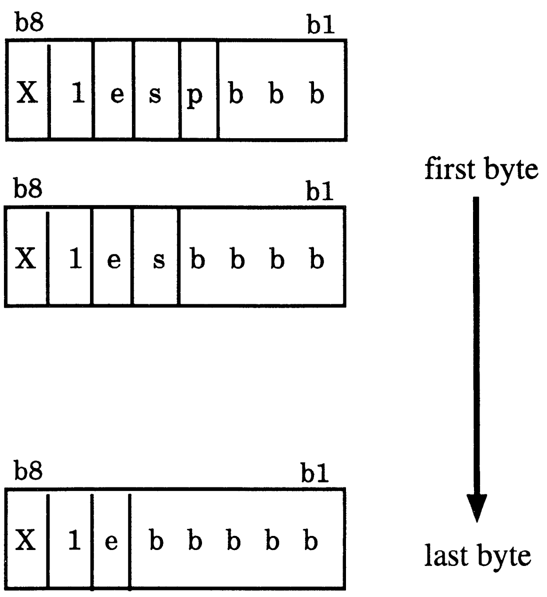
The exponent may be implicitly defined as a default exponent, which is then omitted in the real format, or the exponent may be coded explicitly as the second part of the real format:

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

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

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

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



If *explicit exponent allowed* = ALLOWED, i.e. the *exponent follows bit* is present.

If explicit exponent allowed = FORBIDDEN, i.e. there is no *exponent follows bit*.

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

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

Here *p* is the *exponent follows bit.* If the current *explicit exponent allowed* value of the REAL PRECISION function is ALLOWED, bit b4 of the first byte of a mantissa is used as the *exponent follows bit:* 1 if an explicit exponent follows, 0 if no exponent follows the mantissa (then the default exponent set by REAL PRECISION is assumed). If the current *explicit exponent allowed* value of the REAL PRECISION function is FORBIDDEN, bit b4 of the first byte is used as a databit and the default exponent set by REAL PRECISION is assumed.

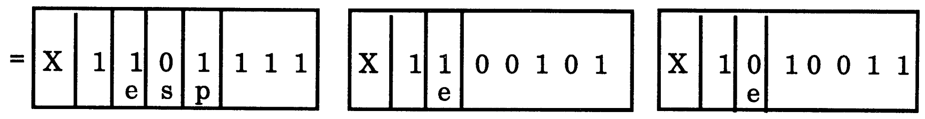
The exponent is coded as an integer in basic format, see 6.3.

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

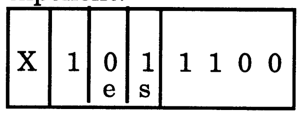
For example, suppose that REAL PRECISION has specified that each real parameter is to be coded with an *exponent follows bit* in its mantissa. In that case, the binary numeral +1.110010110011 is coded as follows:

(real: binary +1.11 00101 10011)
= (mantissa: +111 00101 10011, "exponent follows")
(exponent: −12)

mantissa:



exponent:

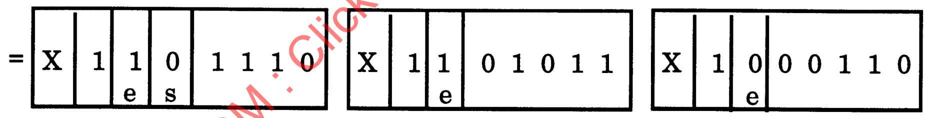


= 6/15 6/5 5/3 5/12

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

(real: binary +1.11 00101 10011)
= (mantissa: +1110 01011 0011)

mantissa:



= 6/14 6/11 4/6

6.5 Coding Virtual Device Coordinates and Points (VDC, P)

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

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

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

- If the point is not in a point list, or is the first point of a point list, the omitted exponent assumes the default value for exponents of VDC parameters, as specified by VDC REAL PRECISION.
- If the point is in a point list, but is not the first point of that point list, an omitted exponent may assume a different value as follows: if the exponent is omitted from the x-component of a real point, it defaults to the value of the exponent in the preceding x-component; similarly, an exponent omitted from a y-component assumes the value of the exponent in the preceding y-coordinate.

6.6 Coding point list parameters (nP)

A point list may be coded with one or both of the following coding structures. These structures are called displacement mode and incremental mode. Both formats may be used within a single point list.

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

The point list shall be terminated by the opcode LIST TERMINATOR (3/7 3/1).

6.6.1 Displacement mode

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

<P> = <VDC: delta x> <VDC: delta y>

6.6.2 Incremental mode

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

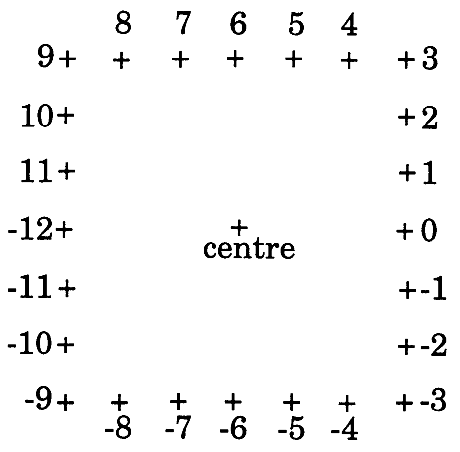
A ring is characterized by its *radius* (*R*) in basic grid units (*BGU*), its *angular resolution* (by a factor *p*) and its *direction* (*D*). The maximum number of points on a ring is *8R*. The actual number of points on a ring with a given *angular resolution factor p* follows from:

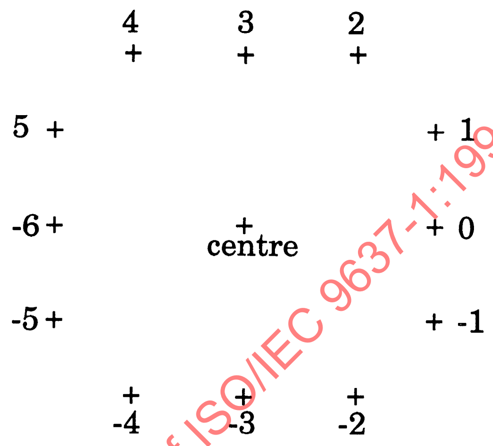
$$N = \frac{8R}{2^p} \quad , \qquad p = 0, 1, 2, 3$$

N is required to be even.

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

Following are examples of two rings, both having *radius R=3*. The ring on the left has *angular resolution factor p=0* and the one on the right has *angular resolution factor p=1*.

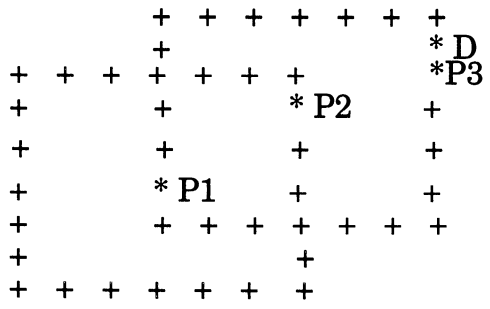




The *direction* of the ring is given by the position of the point with number ZERO. The initial position of this point is on the positive x-axis of VDC space, while the cartesian axes are drawn through the centre point of the ring. The *direction* of the rings following the initial one is dependent on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

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

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



Change of direction with *R*=3

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

smallest-real code	basic radius
$>g$	1
$=g$	1
$<g$	$2^g/BGU$

where g is a nominal *smallest real code*. A value of $g = -8$ is used in ISO/IEC 9637-1.

The basic *radius* and the *angular resolution factor* may be changed with the DOMAIN RING function. If the basic *radius* is set to a value less than ONE, the value ONE is assumed for the basic *radius*.

The encoding used in incremental mode makes use of the DCC property by using variable length code-words (Huffman Code). Before entering the incremental mode, the *radius* and the *angular resolution factor* can be specified by the DOMAIN RING function. The default value for the *radius* is 1 *BGU,* the default value for the *angular resolution factor* is 0. When entering the incremental mode, values set by the DOMAIN RING function are in effect. The encoding also allows changing of the effective *radius* and the effective *angular resolution factor*. The effective *radius* can have a value of *R, 2R, 4R or 8R*, where *R* is the basic *radius* set by the DOMAIN RING function. The effective *angular resolution factor* can be 0, 1, 2, or 3.

Changes to the effective *radius* and the effective *angular resolution factor* within the incremental mode have no effect on the basic values set by the DOMAIN RING function.

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

Table 2 — Huffman Code table for incremental mode

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

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

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

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

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

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

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

6.6.3 Incremental mode encoding

The structure of the incremental mode format is given below:

b8							b1	
X	1	0	1	0	0	0	0	first byte

b8							b1	
X	1	0	1	0	0	0	1	second byte

b8							b1	
X	1	b	b	b	b	b	b	other bytes

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

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

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

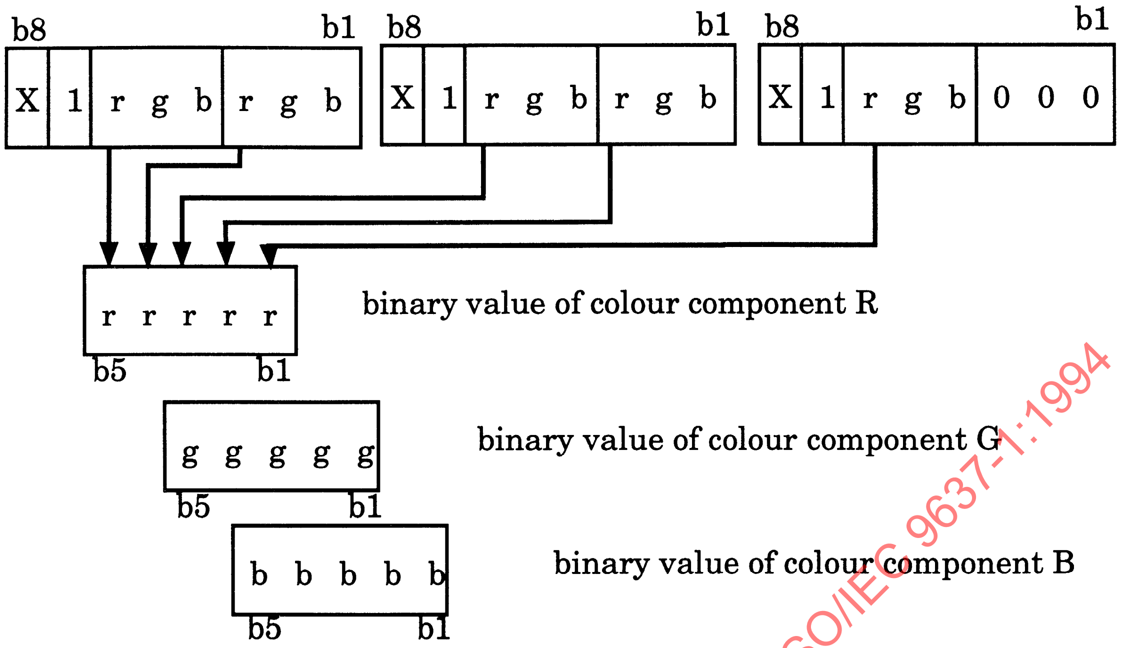
The end of incremental mode data is identified by the <End of Block> code. Remaining bits in the last incremental mode data byte have no meaning; they are required to be zero.

6.7 Colour specifiers (CI, CD)

A colour specifier is coded either as a colour index (if COLOUR SELECTION MODE is 'indexed') or as an RGB parameter (if COLOUR SELECTION MODE is 'direct').

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

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


b8
b1
X
1
r g b
r g b
r g b 0 0 0
r r r r r
b5
b1
binary value of colour component R
g g g g g
binary value of colour component G
b b b b b
binary value of colour component B

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

6.8 Colour lists (nCI, nCD)

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

A colour list shall be terminated by the opcode LIST TERMINATOR (3/7 3/1).

A colour list shall be used to represent a parameter which is an array or list of colour specifiers.

This form of encoding shall therefore be used for the following functions and parameters.

Function	**Parameter**
CELL ARRAY	cell colour specifiers
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
PIXEL ARRAY	colour specifiers
GET PIXEL ARRAY	colour specifiers

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

This coding type indicator, bits b2 and b1, is coded in basic format and may have values of 0 through 3; these can be viewed as setting two one-bit flags. Bits b5-b3 are reserved for future use, and shall be set to 0. The coding type indicator has the following format:

b8			b1
X	1	0	0 0 0 r f

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

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

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

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

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

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

6.8.1 Normal format (coding type=0)

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

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

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

6.8.2 Bitstream format (coding type=1)

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

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

6.8.3 Runlength format (coding type=2)

When COLOUR SELECTION MODE is 'indexed', the colour list is a sequence of pairs, each pair consisting of

- CI in basic format at *local colour precision*;
- number of repetitions in basic format.

This is the simplest form of runlength indexed list.

When COLOUR SELECTION MODE is 'direct', the colour list is a sequence of pairs, each pair consisting of

- RGB bitstream at *local colour precision*;
- number of repetitions in basic format.

6.8.4 Runlength bitstream format (coding type=3)

When COLOUR SELECTION MODE is 'indexed', the colour list is a sequence of pairs, each pair consisting of

- CI in bitstream format at *local colour precision*;
- number of repetitions in basic format.

The advantage over runlength format is that the "extend bit" and "sign bit" of the basic format are available as data bits for the index itself, which means that CI in the range 16..63 may be coded in one character instead of two.

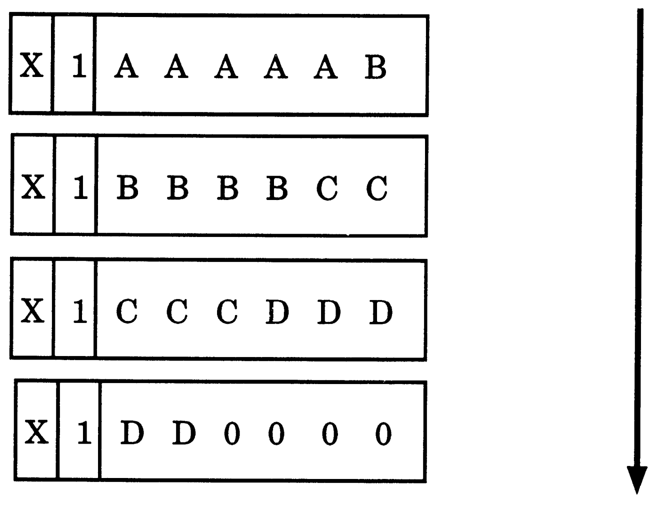
When COLOUR SELECTION MODE is 'direct', this format is identical to the runlength format.

6.8.5 Examples

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

Coding:	runlength
Datatype:	colour index list
Parameter Values:	2,3,3,3,3,4
4/2	first byte: r=1,f=0
4/2,4/1	Run Colour 1 (value 2), Run Count 1 (value 1)
4/3,4/4	Run Colour 2 (value 3), Run Count 2 (value 4)
4/4,4/1	Run Colour 3 (value 4), Run count 3 (value 1)

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



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

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

The following example shows the bitstream coding with the same colour index list as used in the runlength coding:

Coding:	bitstream
Datatype:	colour index list
Parameter Values:	2,3,3,3,3,4 (COLOUR INDEX PRECISION = 4)
4/1	first byte : r=0,f=1
4/8	Colour Index 1 and (half of) 2
7/3	Colour Index (half of) 2 and 3
4/12	Colour Index 4 and (half of) 5
7/4	Colour Index (half of) 5 and 6

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

Coding:	basic format
Datatype:	colour index list
Parameter Values:	2,3,3,3,3,4
4/0	first byte : r=0,f=0
4/2	Colour Index 1 (value 2)
4/3	Colour Index 2 (value 3)
4/3	Colour Index 3 (value 3)
4/3	Colour Index 4 (value 3)
4/3	Colour Index 5 (value 3)
4/4	Colour Index 6 (value 4)

6.9 String parameters (S, SF)

6.9.1 Overall string parameter format

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

SOS is represented by the C1 control 09/08. In a 7-bit environment SOS is coded ESC 5/8 (where ESC is the bit combination 1/11). ST is coded as ESC 5/12 in a 7-bit environment, and 09/12 in an 8-bit environment.

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

A string parameter of data type SF always uses the fixed character set ISO/IEC 646 and is not subject to control by CHARACTER CODING ANNOUNCER.

6.9.2 Bit combinations permitted within string parameters of text functions

If the character set is a complete code then all bit combinations are allowed. If ST is used, it terminates the string and so ST cannot be used within a string. For all other character set types the following bit combinations are allowed as bytes in string parameters in a conforming CGI data stream:

a) Bit combinations from columns 2 through 7 of a 7-bit or 8-bit code chart. If the CHARACTER CODING ANNOUNCER has selected an 8-bit coding technique, then bit combinations from columns 10 through 15 of an 8-bit code chart are also permitted within string parameters.

b) The following C0 control characters: NUL (0/0), BS (0/8), HT (0/9), LF (0/10), VT (0/11), FF (0/12), CR (0/13), SO (0/14), SI (0/15) and ESC (1/11). The NUL control is permitted, but has no effect on the meaning of the string. The bit combinations 0/8 to 0/13 are permitted, but have no standardized effect. The SO, SI, and ESC controls are permitted within a conforming CGI data stream, dependent on the CHARACTER CODING ANNOUNCER. If the bit combinations 0/14 or 0/15 are present, they shall have the meanings (SO or LS1, and SI or LS0) specified for them in ISO 2022, unless a private coding technique has been selected by CHARACTER CODING ANNOUNCER. Other C0 controls are reserved for future standardization.

c) The ESCAPE control (bit combination 1/11 with the exception of the bit combinations 1/11 5/8 and 1/11 5/12 for SOS and ST), and escape sequences formed according to the rules specified in ISO 2022. Those escape sequences for which meanings are specified in ISO 2022 shall have those specified meanings. Escape sequences are only permitted if the CHARACTER CODING ANNOUNCER has selected a ISO 2022-compatible technique. (If a private value of CHARACTER CODING ANNOUNCER is selected, ESCAPE may be used for escape sequences in a manner agreed upon by the interchanging parties. However this is not recommended as it will decrease the ability to interchange the CGI data.)

d) When the 7-bit or extended 8-bit mode has been selected using the CHARACTER CODING ANNOUNCER, the bit combinations 08/14 and 08/15 (or ESC 4/14 and ESC 4/15) are permitted and, if present, shall have the meanings SS2 (SINGLE SHIFT TWO) and

SS3 (SINGLE SHIFT THREE) as defined in ISO 2022.

The effect of these bit combinations is local to the string or compound string in which they appear.

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

6.9.3 C0 control within string parameters

Table 3 — C0 control set.

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

6.9.4 Using G-sets in string parameters

The G-sets (G0, G1, G2, and G3) are coded character sets of 94 or 96 bit combinations. They may be invoked into columns 2 through 7 of a 7-bit code chart, or columns 02 through 07 and 10 through 15 of an 8-bit code chart. This encoding of the CGI uses the G0 and G1/G2 sets within string parameters. The G3 set may be used within the string parameters of a conforming CGI data stream; this requires selection of the extended 7-bit or extended 8-bit mode in the CHARACTER CODING ANNOUNCER. ISO/IEC 9636 does not provide a function to explicitly designate the G3 sets; this may be done within a text string in accordance with ISO 2022, or by other means agreed upon by the interchanging parties.

6.9.4.1 String parameters and character sets

The following CGI functions have one or more string parameters:-

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

ECHO REQUEST STRING

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) as the G0 set. Likewise, ALTERNATE CHARACTER SET INDEX designates a particular character set as both the G1 and G2 sets.

At the start of the CGI data stream, or whenever it is subsequently issued, the INITIALIZE function invokes the G0 set into columns 2 through 7 of a 7-bit or 8-bit code chart. In an 8-bit environment INITIALIZE invokes the G1 set into columns 10 through 15 of the 8-bit code chart.

6.9.4.2 String parameters of other CGI functions

The CHARACTER SET INDEX and ALTERNATE CHARACTER SET INDEX functions apply only to the string parameters of the functions identified in 6.9.4.1. ISO/IEC 9637-1 does not specify the effect, if any, of using characters from code chart columns 10 through 15 in the string parameters of functions other than the functions identified in 6.9.4.1.

ISO/IEC 9637-1 does not specify the effect of ISO 2022 designating and invoking controls when they occur within the string parameters of functions other than the functions identified in 6.9.4.1.

6.10 Enumerated parameters (E)

Enumerated parameters represent choices within a fixed set of standardized options. All of the enumerated types may have private values. They are coded as integers (basic format) with a precision of 10 bits. Private (non-standard) values of enumerated parameters shall use negative integers.

Enumerations standardized in ISO/IEC 9636 shall be encoded as basic format integers beginning with 0 and ascending in a continuous sequence. The first enumerated value shown in lists in the formal grammars of ISO/IEC 9636 shall be allocated the encoded representation 0, the next 1, and so on.

This general rule 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 below.

Inheritance filter selection — enumerated values

Value	enumerated	Value	enumerated
0	LINE TYPE ASF	37	TEXT PRECISION
1	LINE WIDTH ASF	38	TEXT PATH
2	LINE COLOUR ASF	39	TEXT ALIGNMENT
3	MARKER TYPE ASF	40	FILL BUNDLE INDEX
4	MARKER SIZE ASF	41	INTERIOR STYLE
5	MARKER COLOUR ASF	42	FILL COLOUR
6	TEXT FONT INDEX ASF	43	HATCH INDEX
7	TEXT PRECISION ASF	44	PATTERN INDEX
8	CHARACTER EXPANSION FACTOR ASF	45	FILL BITMAP
9	CHARACTER SPACING ASF	46	EDGE BUNDLE INDEX
10	TEXT COLOUR ASF	47	EDGE TYPE
11	INTERIOR STYLE ASF	48	EDGE WIDTH
12	FILL COLOUR ASF	49	EDGE COLOUR
13	HATCH INDEX ASF	50	EDGE VISIBILITY
14	PATTERN INDEX ASF	51	EDGE CLIPPING MODE
15	FILL BITMAP ASF	52	FILL REFERENCE POINT
16	EDGE TYPE ASF	53	PATTERN SIZE
17	EDGE WIDTH ASF	54	AUXILIARY COLOUR
18	EDGE COLOUR ASF	55	TRANSPARENCY
19	EDGE VISIBILITY ASF	56	DRAWING MODE
20	LINE BUNDLE INDEX	57	PICK IDENTIFIER
21	LINE TYPE	58	LINE ASFS
22	LINE WIDTH	59	MARKER ASFS
23	LINE COLOUR	60	TEXT ASFS
24	LINE CLIPPING MODE	61	FILL ASFS
25	MARKER BUNDLE INDEX	62	EDGE ASFS
26	MARKER TYPE	63	ALL ASFS
27	MARKER SIZE	64	LINE ATTRIBUTES
28	MARKER COLOUR	65	MARKER ATTRIBUTES
29	MARKER CLIPPING MODE	66	LOCAL TEXT ATTRIBUTES
30	TEXT BUNDLE INDEX	67	GLOBAL TEXT ATTRIBUTES
31	TEXT FONT INDEX	68	FILL ATTRIBUTES
32	TEXT COLOUR	69	EDGE ATTRIBUTES
33	CHARACTER EXPANSION FACTOR	70	PATTERN ATTRIBUTES
34	CHARACTER SPACING	71	OUTPUT CONTROL
35	CHARACTER HEIGHT	72	PICK ATTRIBUTES
36	CHARACTER ORIENTATION	73	ALL ATTRIBUTES
		74	ALL

6.11 Index parameters (IX)

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

6.12 Client specified name parameters (CSN)

Client specified name (CSN) parameters are coded as integers (basic format), in a range limited by CLIENT SPECIFIED NAME PRECISION.

6.13 Input colour parameters (ICO, nICO)

The encoding of ICO values depends on the current state of the generating raster input device.

If the state list entry *colour* is NO, i.e. the raster input device delivers grey-scale information, these values are represented by integers at the precision given by the state list entry *bits per colour*.

If the state list entry *colour* is YES, i.e. the input device delivers colour information, these RGB-values are triplets of integers, each at the precision given by the *bits per colour* entry.

A list of input colour specifiers (nICO) is basically coded as a colour list (see 6.8) with *colour selection mode* DIRECT.

In case of grey-scale information, i.e. *colour* = NO, each value in the list consists of one integer. In case of colour information, i.e. *colour* = YES, each value in the list consists of a RGB-triplet.

This form of encoding shall be used for the parameter *list of input colour values* of the functions

PUT CURRENT RASTER MEASURE
REQUEST RASTER
SAMPLE RASTER
ECHO REQUEST RASTER
DEQUEUE RASTER EVENT

and for raster input measures within an event input record.

6.14 Data record parameters (D)

Like a string a data record parameter is introduced by START OF STRING (SOS) and terminated by STRING TERMINATOR (ST) (see 6.9).

The contents of a data record is coded as zero or more typed sequences of data items, where each sequence consists of a data type index, an element count, and the corresponding data elements of that type.

The data type index is coded as an integer (basic format) at the current INDEX PRECISION, the assignment of indices to data types is as follows:

1	data record (D)	12	real (R)
2	colour index (CI)	13	string (S)
3	colour direct value (CD)	14	fixed string (SF)
4	client specified name (CSN)	15	viewport coordinate (VC)
5	enumerated (E)	16	virtual device coordinate (VDC)
6	integer (I)	17	<reserved>
7	input solour specifier (ICO)	18	<reserved>
8	fixed 8-bit integer (IF8)	19	<reserved>
9	fixed 16-bit integer (IF16)	20	<reserved>
10	fixed 32-bit integer (IF32)	21	colour specifier list (CL)
11	index (IX)		

NOTES Data type CSN (client specified name) in ISO/IEC 9636 is the same as data type N (name) in ISO 8632.

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

The element count specifies the number of data items to follow in the sequence, and is coded as an integer (basic format) at the current INTEGER PRECISION.

The data items are coded in the format corresponding to the data type index and are subject to all of the relevant modes and precisions for that type.

A typed sequence within a data record can recursively consist of data records. These data items are also delimited by START OF STRING and STRING TERMINATOR.

7 Character substitution

To accommodate systems in which it is inconvenient or impossible to include C0 control characters, the SPACE character (2/0), or the DELETE character (7/15) in the CGI data stream, this CGI encoding includes a "character substitution" option. Characters in the range of 0/0 to 1/15, the characters 2/0, 7/14, and 7/15 may be replaced by 2-byte sequences provided each such 2-byte sequence is declared as an extra parameter of the INITIALIZE function.

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

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

Once a character (or rather, its 2-byte replacement) has been declared in the INITIALIZE function, "character substitution" is in effect immediately after the end of character substitution for that character and will remain in effect until the end of the CGI data stream (that is, until the TERMINATE function). The CGI interpreter would ignore any characters for which character substitution is in effect.

Example:

> Supposing that it is desirable to avoid using the characters SPACE and DELETE in the CGI data stream, and for the CGI interpreter to ignore those characters if they are inadvertently inserted (for example, by a host operating system or some process other than the CGI generator).
>
> In that case, the CGI generator declares "character substitution" for the above two characters and for the TILDE character, 7/14. It does this in the additional parameter of INITIALIZE as follows:

```
3/0 3/2                        {INITIALIZE opcode}
7/14 6/0 7/14 3/14 7/14 3/15   { string: ~‘~>~?}
1/11 5/12                      { STRING TERMINATOR (ST)}
```

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

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

Table 4 — Character substitution

The character:		May be replaced with: 7-bit		8-bit		ISO 646 char.
0/0	(NUL)	7/14	4/0	07/14	04/00	(~@)
0/1	(SOH)	7/14	4/1	07/14	04/01	(~A)
0/2	(STX)	7/14	4/2	07/14	04/02	(~B)
0/3	(ETX)	7/14	4/3	07/14	04/03	(~C)
0/4	(EOT)	7/14	4/4	07/14	04/04	(~D)
0/5	(ENQ)	7/14	4/5	07/14	04/05	(~E)
0/6	(ACK)	7/14	4/6	07/14	04/06	(~F)
0/7	(BEL)	7/14	4/7	07/14	04/07	(~G)
0/8	(BS)	7/14	4/8	07/14	04/08	(~H)
0/9	(HT)	7/14	4/9	07/14	04/09	(~I)
0/10	(LF)	7/14	4/10	07/14	04/10	(~J)
0/11	(VT)	7/14	4/11	07/14	04/11	(~K)
0/12	(FF)	7/14	4/12	07/14	04/12	(~L)
0/13	(CR)	7/14	4/13	07/14	04/13	(~M)
0/14	(SO)	7/14	4/14	07/14	04/14	(~N)
0/15	(SI)	7/14	4/15	07/14	04/15	(~O)
1/0	(DLE)	7/14	5/0	07/14	05/00	(~P)
1/1	(DC1)	7/14	5/1	07/14	05/01	(~Q)
1/2	(DC2)	7/14	5/2	07/14	05/02	(~R)
1/3	(DC3)	7/14	5/3	07/14	05/03	(~S)
1/4	(DC4)	7/14	5/4	07/14	05/04	(~T)
1/5	(NAK)	7/14	5/5	07/14	05/05	(~U)
1/6	(SYN)	7/14	5/6	07/14	05/06	(~V)
1/7	(ETB)	7/14	5/7	07/14	05/07	(~W)
1/8	(CAN)	7/14	5/8	07/14	05/08	(~X)
1/9	(EM)	7/14	5/9	07/14	05/09	(~Y)
1/10	(SUB)	7/14	5/10	07/14	05/10	(~Z)
1/11	(ESC)	7/14	5/11	07/14	05/11	(~[)
1/12	(FS or IS4)	7/14	5/12	07/14	05/12	(~\)
1/13	(GS or IS3)	7/14	5/13	07/14	05/13	(~])
1/14	(RS or IS2)	7/14	5/14	07/14	05/14	(~^)
1/15	(US or IS1)	7/14	5/15	07/14	05/15	(~_)
2/0	(SPACE)	7/14	6/0	07/14	06/00	(~‘)
7/14	(TILDE,~)	7/14	3/14	07/14	03/14	(~>)
7/15	(DELETE)	7/14	3/15	07/14	03/15	(~?)

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

8 Representation of CGI functions

This part of ISO/IEC 9637 does not replicate the formal grammar or function definitions which appear in the CGI Functional Specification ISO/IEC 9636.

Instead, this clause defines the rules by which the function definitions and formal grammar in ISO/IEC 9636 may be mapped onto their corresponding representation in character-encoded data.

The CGI Character 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 the CGI data types which admit variable precision. If these functions are not present in the CGI character data stream, the default precisions are assumed (see clause 9).

Some of the input functions specified in ISO/IEC 9636-5 are described as generic functions. 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 are encoded as separate function representations. Likewise, any responses for these functions are encoded as separate response representations.

8.1 Soliciting functions

A CGI function whose definition in ISO/IEC 9636 has a number of *Out* parameters is termed a *soliciting function*. For such functions, the encoded form of the outward data produced by the generator (the *In* parameters), and of the data returned from the interpreter (the *Out* parameters), shall be introduced by distinct opcodes, as shown in table 1.

The outward encoded data shall contain encoded representations of the CGI function's *In* parameters, and the response encoded data shall contain encoded representations of the CGI function's *Out* parameters.

All soliciting CGI functions have an initial *Out* parameter which is a *Response Validity*. When the interpreter returns such a response with the *Response Validity* set to INVALID, then no further parameters shall be encoded.

CGI functions which demand a response from the interpreter are thus encoded according to the following scheme:

Outward data stream:

<outward opcode> <*In* param1>... <*In* paramN>

Returned data stream:

<response opcode> <*Out* param1>... <*Out* paramM>

For example, consider the handling of the CGI function

SAMPLE CHOICE (IX=1, *response validity, measure validity, current value*)

The outward data is encoded as:

3/11 3/3	(SAMPLE CHOICE outward)
4/1	(IX=1 — *In* param 1)

The returned data is encoded as

3/13 3/3	(SAMPLE CHOICE response)
4/1	(*response validity* = VALID — *Out* param 1)
4/1	(*measure validity* = VALID — *Out* param 2)
4/1	(CHOICE_VALUE = 1 — *Out* param 3)

Alternatively, if the *response validity* were INVALID, then the returned data would have been encoded as

3/13 3/3	(SAMPLE CHOICE response)
4/0	(*response validity* = INVALID)

8.2 Enumerated parameters

As stated in paragraph 6.10 above, this part of ISO/IEC 9637 does not itself explicitly define the binding of enumerated parameters. Instead, the order of enumerated values as given in the formal grammar of ISO/IEC 9636 determines an integer value. Thus, the first enumerated value given in ISO/IEC 9636 shall be encoded as a basic format integer 0, the second enumerated value as 1, and so on.

8.3 Functions not specified in ISO/IEC 9636

8.3.1 DOMAIN RING

Parameters:

In	angular resolution factor	0..3	I
In	radius		I

Effect:

This function specifies the precision of the coordinate encoding, when using incremental mode.

The actual number of points on a ring with a given *radius* is determined by the *angular resolution factor*. If the *radius* parameter value is zero, the basic *radius* is used, which is 1 for VDC TYPE integer. For VDC TYPE real the basic *radius* is determined from the current *smallest real code* (as set with VDC REAL PRECISION).

8.3.2 INTEGER PRECISION

Parameters:

In	largest integer code + 1	I

Effect:

The *largest integer code* tells how many bits occur in the largest possible magnitude for an integer. For example, if integers in the encoded data can range from -32 767 to +32 767, the *largest integer code* is 15. One additional bit is required for the sign, and so is added to obtain the proper precision. Thus in this example the parameter would be 16.

8.3.3 REAL PRECISION

Parameters:

In	largest real code + 1		I
In	smallest real code		I
In	default exponent		I
In	exponent allowed flag	(ALLOWED, FORBIDDEN)	E

Effect:

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

The *largest real code* tells how many bits can occur to the left of the (binary) radix point in the largest possible real number. For example, if the largest possible real number is binary +11111111111111111111.111, the *largest real code* would be 20.

The *smallest real code* shows the granularity of the real number space by indicating how small a non-zero real number can be. It does this by specifying the smallest exponent that is permitted in the coded representation of a real number. For example, if all real numbers are integer multiples of 1/64 (2 raised to the power -6), the *smallest real code* would be -6. This indicates that there are six bits of precision to the right of the binary radix point and that the smallest allowable exponent is -6.

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

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

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

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

8.3.4 INDEX PRECISION

Parameters:

In	largest index code + 1	I

Effect:

The *largest index code* tells how many bits occur in the largest possible magnitude for an index. For example, if indexes in the encoded CGI data can range from -511 to +511, the *largest index code* is 9. One additional bit is required for the sign, and so is added to obtain the proper precision. Thus in this example the parameter would be 10.

8.3.5 COLOUR PRECISION

Parameters:

	largest colour component code	I

Effect:

The *largest colour component code* tells how many bits occur in the largest possible magnitude for a direct colour component, i.e., for each of the red, green, and blue components. For example, if the *largest colour component code* is 10, then direct colour components in the range 0...1 023 can be handled.

8.3.6 COLOUR INDEX PRECISION

Parameters:

In	largest colour index code + 1	I

Effect:

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

8.3.7 VDC INTEGER PRECISION

Parameters:

In	largest integer code + 1	I

Effect:

The *largest integer code* tells how many bits occur in the largest possible magnitude for an integer point. For example, if integer points in the encoded data can range from -32 767 to +32 767, the *largest integer code* is 15. One additional bit is required for the sign, thus an additional bit is added to obtain the appropriate field width. Thus in this example the parameter would be 16.

8.3.8 VDC REAL PRECISION

Parameters:

In	largest real code + 1		I
In	smallest real code		I
In	default exponent		I
In	exponent allowed flag	(ALLOWED, FORBIDDEN)	E

Effect:

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

The *largest real code* tells how many bits can occur to the left of the (binary) radix point in the largest possible VDC scalar. For example, if the largest possible real number is binary +11111111111111111111.111, the *largest real code* would be 20.

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

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

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

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

The *exponent allowed flag* specifies whether or not an *exponent follows bit* is included in the mantissa of the coded representation of a real VDC coordinate or VDC scalar. If this flag is 0, exponents are allowed in real VDC coordinates and VDC scalars, and the *exponent follows bit* is included in their mantissas (see 6.5). If this flag is 1, exponents are not allowed in real VDC coordinates and VDC scalars; in that case the mantissas of real VDC coordinates and VDC scalar do not include the *exponent follows bit.* In that case, the omitted exponents of all real parameters are deemed to have the value determined by the *default exponent* parameter.

8.3.9 CLIENT SPECIFIED NAME PRECISION

Parameters:

In	largest integer code + 1	I

Effect:

The *largest integer code* tells how many bits occur in the largest possible magnitude for a client specified name. For example, if names in the encoded data can range from -32 767 to +32 767, the *largest integer code* is 15. One additional bit is required for the sign, thus an additional bit is added to obtain the appropriate field width. Thus in this example the parameter would be 16.

8.4 Encoding of matrices

Certain CGI functions take parameters which are matrices. For example, COPY SEGMENT's *copy transformation* parameter consists of a 2X2 matrix of R, and a 2X1 matrix of VDC. Similarly, CELL ARRAY and PIXEL ARRAY, which have a parameter which is an array of colour specifiers.

This clause defines the method of encoding such matrices.

Where a parameter consists of a matrix of some object, then the elements of the matrix shall be encoded in row-major order. Thus, a 2X3 matrix of integers shall be encoded as I11, I12, I13, I21, I22, I23.

9 Defaults

Parameter precisions for the CGI character encoding:

REAL PRECISION:	*largest real code:* 10 *smallest real code:* -10 *default exponent:* -10 *exponent allowed flag:* FORBIDDEN
INTEGER PRECISION:	10
INDEX PRECISION:	10
COLOUR PRECISION:	6
COLOUR INDEX PRECISION:	10
COLOUR VALUE EXTENT:	*minimum colour value:* 0,0,0 *maximum colour value:* 63,63,63
VDC REAL PRECISION:	*largest real code:* 10 *smallest real code:* -10 *default exponent:* -10 *exponent allowed flag:* FORBIDDEN
VDC INTEGER PRECISION:	20
CLIENT SPECIFIED NAME PRECISION:	10

DOMAIN RING	*angular resolution factor:* 0;
	radius for VDC type integer: 1;
	radius for VDC type real if *smallest real code* [(*src*) < -8]: [2^{-8-src}];
	radius for VDC type real if *smallest real code* [(*src*) >= -8]: 1.

10 Classification and designation

Conformance: A CGI data stream conforms to this encoding if it satisfies the encoding specific conformance criteria given in clause 7 of ISO/IEC 9636-1 and satisfies the additional conformance requirements given below:

- each CGI function described in ISO/IEC 9636 is coded as a function representation or response representation (where applicable) in the manner described in this part of ISO/IEC 9637;
- private (non-standard) CGI function representations are all coded using ESCAPE and GDP functions. Opcodes reserved for future standardization shall not be used to code private (non-standard) function representations;
- private (non-standard) values of index and enumerated parameters are all coded using negative integers.;
- values specified as being *reserverd for registration or future standardization* are not used unless their meaning has been registered or standardized.

A conforming CGI character encoded data stream may include, within the string parameters of those functions listed in 6.9.4.1. as well as string parameters within the data records of GENERALIZED DRAWING PRIMITIVE (GDP) function representations (their admissibility here depends upon the particular GDP definition), the ISO 2022 controls for designating and invoking G-sets, but shall not include the escape sequence used to enter the encoding from ISO 2022, except as the first character of the data stream.

A CGI generator or interpreter to this encoding shall fully support the following coding methods for those implemented functions and responses that contain parameters with the indicated data types:

a) for point lists (nP), both incremental and displacement mode encoding shall be supported, including the capability to change point list mode within a point list;

b) for colour lists (nCI, nCD, nICO), all specified colour list formats shall be supported, including the capability to change colour list format within a colour list;

c) for real numbers (R), both explicit and default exponents shall be supported.