INTERNATIONAL STANDARD

ISO/IEC 9636-1

First edition
1991-12-15

Information technology — Computer graphics — Interfacing techniques for dialogues with graphical devices (CGI) — Functional specification —

Part 1:
Overview, profiles, and conformance

Technologies de l'information — Infographie — Interfaces pour l'infographie — Spécifications fonctionnelles —

Partie 1: Résumé, profils et conformité

Reference number
ISO/IEC 9636-1:1991(E)

Contents

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 9636-1 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology.*

ISO/IEC 9636 consists of the following parts, under the general title *Information technology — Computer graphics — Interfacing techniques for dialogues with graphical devices (CGI) — Functional specification*:

— *Part 1: Overview, profiles, and conformance*
— *Part 2: Control*
— *Part 3: Output*
— *Part 4: Segments*
— *Part 5: Input and echoing*
— *Part 6: Raster*

Annexes A and B form an integral part of this part of ISO/IEC 9636. Annexes C, D, E and F are for information only.

Introduction

Purpose

The Computer Graphics Interface (CGI) specifies a standard interface between device-independent and device-dependent parts of a graphics system. ISO/IEC 9636 specifies sets of functions for control and data exchange over this interface. This interface may be implemented as a software-to-software interface (as a data stream encoding used in a network environment or as a procedural binding to one or more software packages), or as a software-to-hardware interface (as a data stream encoding to one or more devices presenting a standardized Computer Graphics Interface as their device protocol). Related standards specify data stream encodings (for use in the software-to-software case, over a network, and in the software-to-hardware case), and procedural bindings (for use in the software to software case).

ISO/IEC 9636 includes a reference model and a description of the CGI's relationship to other standards.

ISO/IEC 9636 only defines graphics functions, graphics control functions, and functions to control data representations and deferral in data stream encodings. ISO/IEC 9636 does not define the protocols to be used to convey these functions between the parts of a (potentially distributed) system.

Benefits

Intrinsic	The CGI will simplify the development and implementation of graphics systems. ISO/IEC 9636 will encourage a uniform access to the graphics devices within an installation. As new devices are made available, graphics device drivers adhering to this interface can be installed for use by existing programs.
Interchange	ISO/IEC 9636 promotes the exchange of software between installations. By isolating the device-dependent aspects of any graphics system, modularity is encouraged, which promotes increased portability. The standard set of functions, access mechanisms, and terminology will allow developers and users to move between installations with minimal retraining.
Educational	The standard set of functions uses a standard terminology. This allows both the academic and industrial communities to develop instructional programs concentrating on programming techniques and methodologies based on these standard functions.
Economic	In view of the trend towards lower hardware and higher software costs, the following benefits accrue from ISO/IEC 9636:

- It encourages transporting of software between installations, thereby reducing costs associated with "reinvention";
- It protects the large software investment made by both users and vendors because the software will not be rendered obsolete by the introduction of new devices;
- It allows developers of new software to focus on higher-level graphics functions and applications instead of device-level functions;
- It reduces maintenance of software systems because the standard encourages modularity;
- It increases vendor independence for the user because any system designed to use a particular device can more easily be changed to use some other device;
- It allows vendors to develop and market devices that will easily interface to the customer's system;
- It enables users, manufacturers and vendors to take advantage of new, lower-cost graphics hardware designs. The total system's hardware cost may be reduced because system redesign may not be necessary.

Design requirements

To realize the benefits described above, a number of design principles have been adopted:

a) The Computer Graphics Interface should provide a suitable set of functions for the description of a wide range of pictorial information;

b) The Computer Graphics Interface should provide a suitable set of functions for the necessary CGI session control of a wide range of graphics devices;

c) The Computer Graphics Interface should address the more usual and essential features found on graphical devices directly and should provide access to less common facilities;

d) The design of the Computer Graphics Interface should not preclude extension of ISO/IEC 9636 at a later stage to cover facilities currently not standardized;

e) The Computer Graphics Interface should be usable from GKS (Graphical Kernel System - ISO 7942). In particular, the CGI should include functional capability to support the various levels of a GKS workstation in an efficient and concise manner, without compromising the ability of the interface to support non-GKS systems in an efficient and concise manner;

f) The Computer Graphics Interface should be compatible with the Computer Graphics Metafile - ISO 8632. In particular, those CGM elements not associated with the file-oriented aspects of the CGM shall have corresponding CGI functions which have identical abstract names and parameterization;

g) ISO/IEC 9636 should address the needs of different applications that have conflicting requirements for

- allocation of processing burden between host and device;
- speed of generation and interpretation of functions;
- ease of transfer through different transport mechanisms.

Design criteria

The above requirements were used to formulate the following design criteria:

a) Completeness
In any area of ISO/IEC 9636, the functionality specified by ISO/IEC 9636 should be complete in itself.

b) Conciseness
Redundant functions or parameters should be avoided.

c) Consistency
Contradictory functions should be avoided.

d) Extensibility
The ability to add new functions and generality to ISO/IEC 9636 should not be precluded.

e) Fidelity
The results and behaviour of functions should be well defined.

f) Implementability
A function should be able to be efficiently supported on most host systems andor graphics hardware.

g) Orthogonality
Independent functions for separate and noninteracting activities should be provided.

h) Predictability
The recommended or proper use of a standard function should guarantee the results of using that particular function.

i) Standard practice
Only those functions that reflect existing practice, that are necessary to support existing practice, or that are necessary to support standards being developed concurrently should be standardized.

j) Usefulness
Functions should be powerful enough to perform useful tasks.

k) Well-structured
The number of assumptions that functions make about each other should be minimized. A function should have a well-defined interface and a simply stated unconditional purpose. Multi-purpose functions and side effects should be avoided.

Parts of the CGI functional specification

ISO/IEC 9636, the functional specification of the Computer Graphics Interface, consists of a number of parts presenting portions of the CGI functionality, including an overview in this part of ISO/IEC 9636.

Table 1 – Parts of the CGI Functional Specification

Part No.	Title
ISO/IEC 9636-1	Overview, profiles, and conformance
ISO/IEC 9636-2	Control
ISO/IEC 9636-3	Output
ISO/IEC 9636-4	Segments
ISO/IEC 9636-5	Input and echoing
ISO/IEC 9636-6	Raster

This part of ISO/IEC 9636 gives a general overview and introduction to the basic concepts and principles of ISO/IEC 9636. It includes a reference model, the relationship to other standards, and profiles. In addition, it contains overviews of each of the subsequent parts. This part of ISO/IEC 9636 thus establishes the framework for all the parts of ISO/IEC 9636; it does not contain functional descriptions.

The functional capability provided by the CGI is separate from the specification of any particular encoding format or language binding.

This page intentionally left blank

Information technology – Computer graphics – Interfacing techniques for dialogues with graphical devices (CGI) – Functional specification –

Part 1:
Overview, profiles, and conformance

1 Scope

ISO/IEC 9636 establishes the conceptual model, functional capability, and minimum conformance requirements of the Computer Graphics Interface (CGI). It specifies design requirements for encodings of the CGI. ISO/IEC 9636 defines a set of CGI functions that is expected to satisfy the following needs of a majority of the computer graphics community:

a) provide an interface standard for computer graphics software package implementors;

b) provide an interface standard for computer graphics device manufacturers and suppliers;

c) provide an inquiry and response mechanism for graphics device capabilities, characteristics, and states;

d) provide a standard graphics escape mechanism to access non-standard graphics device capabilities;

e) allow for future functional extension of the CGI.

In addition to the CGI functionality, device classes, and Foundation and Constituency Profiles are defined. The device classes included in the CGI are output (OUTPUT), input (INPUT), and output/input (OUTIN). Profiles allow subsets of the CGI functions and features to be defined to suit particular well identified groups of users. There is also provision for Constituency Profiles to be registered after ISO/IEC 9636 is published. The Computer Graphics Interface (CGI) is a standard functional and syntactical specification of the control and data exchange between device-independent graphics software and an implementation of a CGI Virtual Device.

The syntax of the CGI, presented in ISO/IEC 9636, is an encoding-independent and binding-independent specification. Any similarity of the examples or function specifications to a particular encoding technique or language is coincidental unless explicitly stated otherwise.

The functions specified provide for the representation of a wide range of two-dimensional pictures and for control over their display on a wide range of graphics devices. The functions are split into groups that perform device and CGI session control, specify the data representations used, control the display of the picture, perform basic drawing actions, control the attributes of the basic drawing actions, acquire data from input devices, and provide access to non-standard device capabilities.

This part of ISO/IEC 9636 gives an overview of ISO/IEC 9636, explains the relationship between its parts and their relation to other standards, describes a reference model for graphics systems, and defines certain Foundation and Constituency Profiles. ISO/IEC 9636-2, ISO/IEC 9636-3, ISO/IEC 9636-4, ISO/IEC 9636-5, and ISO/IEC 9636-6 specify the CGI functions for different functional areas using an abstract notation.

ISO/IEC 9637 and ISO/IEC 9638 define standard data stream encodings, procedural library bindings, and single entry point procedural bindings of the CGI.

1.1 Relationship of CGI to a computing environment

ISO/IEC 9636 describes graphical services provided by a Virtual Graphics Device. The model for description of these services is expressed in terms of graphical capabilities of a single instance of a hypothetical graphics device. In all but the simplest of computing environments, CGI functions alone will not be sufficient to provide complete control over a device. Additional functions, not included in ISO/IEC 9636, will likely be needed. Examples of such functions include

- means to configure (sets of) physical devices to be accessed as CGI Virtual Devices;
- means to control a device capable of offering CGI-defined services as well as other, non-CGI-defined services, such as those implied by ISO 2022 and ISO 6429;
- means to differentiate among separate instances of CGI Virtual Devices in the same computing environment;
- means of defining or determining communication paths from CGI clients to CGI Virtual Devices.

In some cases, other standards exist that describe the functions required. For example, various communications standards address the needs of the last point above. In other cases, no standards may exist, but the tasks indicated are outside the scope of ISO/IEC 9636.

1.2 Position of CGI in a managed environment

There exists a large and growing family of computer controlled display systems that have the ability to act as if they are multiple individual display devices. Resources, most notably the visible drawing surface resources, are coordinated by the display system so that multiple non-cooperating client programs can each access the services of a separate individual device while all are actually running in a single managed environment.

The graphical capabilities of the CGI Virtual Device may suffice, in some instances, as the basis for implementing a complex, multiple-client display system. However, the complete needs of such a system are quite complex, include many non-graphical services, and (as current practice shows) are quite technology dependent. The CGI does not, therefore, purport to be a generally sufficient interface on which a managed display environment may be built. Rather, within a managed environment, the CGI will be one of the managed interfaces in a way not visible to the CGI client without recourse to services not part of ISO/IEC 9636. The use of the CGI as a managed interface within a managed display environment is not limited to raster devices.

2 Normative references

The following standards contain provisions which, through reference in this text, constitute provisions of this part of ISO/IEC 9636. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this part of ISO/IEC 9636 are encouraged to investigate the possibility of applying the most recent editions of the standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO 646 : 1983 *Information processing – ISO 7-bit coded character set for information interchange.*

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

ISO 2382-13 : 1984 *Data processing – Vocabulary – Part 13: Computer graphics.*

ISO 6429 : 1988 *Information processing – Control functions for 7-bit and 8-bit coded character sets.*

ISO 7942 : 1985 *Information processing systems – Computer graphics – Graphical Kernel System (GKS) functional description.*

ISO 7498 : 1984 *Information processing systems – Open systems interconnection – Basic reference model.*

ISO 8632-1 : 1987 *Information processing systems – Computer graphics – Metafile for the storage and transfer of picture description information – Part 1: Functional specification.*

ISO 8632-2 : 1987 *Information processing systems – Computer graphics – Metafile for the storage and transfer of picture description information – Part 2: Character encoding.*

ISO 8632-3 : 1987 *Information processing systems – Computer graphics – Metafile for the storage and transfer of picture description information – Part 3: Binary encoding.*

ISO 8805 : 1988 *Information processing systems – Computer graphics – Graphical Kernel System for Three Dimensions (GKS-3D) functional description.*

ISO 9282-1 : 1988 *Information processing – Coded representation of pictures – Part 1: Encoding principles for picture representation in a 7-bit or 8-bit environment.*

ISO/IEC 9592-1 1989 *Information processing systems – Computer graphics – Programmer's Hierarchical Interactive Graphics System (PHIGS) – Part 1: Functional description.*

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

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

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

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

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

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

ISO/IEC 9637-2 : -[1] *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.*

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

1) To be published.

3 Definitions and abbreviations

3.1 Organization of the definitions

The following list of definitions describes the usage of words or phrases which have been given special technical meaning in the context of ISO/IEC 9636. These meanings are not in conflict with dictionary meanings or usage in other ISO and IEC standards. However, there are usually additional semantics associated with the words or phrases which are essential to understanding ISO/IEC 9636.

The list of definitions is arranged in alphabetical order of main entry. Occasionally a term will be used with or without optional qualification in the text; in such cases, the optional qualification appears in parentheses in the term. Under each main entry, certain derived terms may also be covered; all such derived terms are underlined. With a few exceptions, where the appropriate main heading may not be immediately obvious, derived terms do not have separate main entries. Where a phrase appears in italic text, its definition may be as another main entry or it may be a derived term – in which case it will be found either under an entry for one of its constituents (not necessarily its first), under another compound entry with the same first constituent, or under an entry for another word with the same root.

3.2 Definitions

For the purposes of ISO/IEC 9636, the following definitions apply.

3.2.1 acknowledgement
An action performed automatically by a *logical input device* signals to the *operator* that an *input operation* has been completed. Initially enabled by default, this action is controllable by the *client* using acknowledgement control. In a *remote echoing* situation, the acknowledgement has to be explicitly performed by the client. Various acknowledgement may be available and the client can select from these different styles of signalling.

3.2.2 active (for a LID)
A *logical input device* is active if events or sampling have been enabled or if request or echo request input is in progress; otherwise, it is not active.

3.2.3 allowed latitude
There are a number of cases in which ISO/IEC 9636 identifies a limited set of allowed behaviours for functions or features and allows implementors latitude to provide precisely one of this set. In such cases, a *description table* entry gives a *client* the opportunity to inquire the implemented behaviour. ISO/IEC 9636 always identifies one of these allowed behaviours as a preferred behaviour. This method of regulating implementation variability contrasts with that referred to as *implementation dependence*. Where ISO/IEC 9636 allows latitude in the value of a state list default, then a particular value is identified as the preferred default. An example is the default VDC Type, which has allowed latitude in its value, and a preferred default of INTEGER.

3.2.4 appending (to a segment)
See under *segment definition*.

3.2.5 arming (a trigger)
An action which enables a *trigger* to be *fired* by the *operator* to signal the occurrence of something significant. A trigger cannot be fired unless it has been armed. Triggers are armed for a *logical input device* while events are enabled for it and while a request or echo request input operation is in progress.

3.2.6 aspect ratio
The ratio, using a uniform metric, of the width to the height (i.e. x to y) of a rectangular area, such as a *VDC extent* or a *device viewport*.

3.2.7 aspect source flag (ASF)
Aspect source flags which are themselves *attribute values* indicate the source of other attribute values which determine *aspects* of *primitives* or *objects*. If the value of a particular aspect source flag is INDIVIDUAL, then the attribute value for that particular aspect is *associated* with the primitive or object. If the value of a particular aspect source flag is BUNDLED, then

the corresponding *bundle index* attribute value associated with the primitive or object is used to access a *bundle table* to find the attribute value determining the aspect in question.

3.2.8 aspects (of primitive and object)
Characteristics determining the visual (rendered) appearance of *primitives* or *objects* are known as aspects. A particular aspect of a particular primitive or object is determined by an *attribute value* which may be directly *associated* with it or indirectly associated with it by means of a *bundle index* as determined by the appropriate *aspect source flag*. There are other characteristics of primitives and objects which are determined by attribute values, but if these are not concerned with visual appearance or are only specifiable directly (i.e. they cannot be bundled), they are not referred to as aspects.

3.2.9 association (of attribute values)
This refers to various steps in the *graphic object pipeline* where the connection between *graphic objects* and *attribute values* relevant to their *rendering* is established. For *aspects* relevant to a given *primitive type*, the corresponding attribute values are associated during the initial step of object creation and later, just before rendering, for those aspects for which the (already associated) *aspect source flag* values are bundled. For objects stored in segments, the associated object transformation attribute value can be modified by the concatenation of a segment transformation, which is a segment attribute, or a copy transformation, which is a parameter to the copy segment function. Associated attribute values can be replaced by means of the copy segment function and the inheritance filter.

3.2.10 association (of triggers with LIDs)
see under *trigger association*.

3.2.11 attribute function
A function which sets an *attribute value* in a *state list*. Unqualified, the term is used for functions which set *primitive* or *object* attribute values. If other attributes are intended, the term will be appropriately qualified, e.g. segment attribute function or raster attribute function.

3.2.12 attribute value
Certain characteristics of entities are determined by attribute values. When used without qualification, the term implies *primitive* or *object* attribute value. All other uses of attribute value will be appropriately qualified such as *segment attribute value*, *raster attribute value*, etc. Many primitive or object attribute values are directly concerned with rendered appearance and determine *aspects*. Others may not be directly or at all concerned with rendered appearance (such as *ASFs* and pick identifiers). When a single primitive forms a single object, there is no distinction between primitive attribute value and object attribute value. However, when several primitives contribute to one object (*compound text* or *closed figure*), the primitive attribute values are associated with each contributing primitive (see *local attribute value*) while object attribute values are associated with the object as a whole (see *global attribute value*). As an example, auxiliary colour is a primitive (local) attribute value for *edges* but an object (global) attribute value for *interiors* of *fill objects*.

3.2.13 auxiliary colour
The auxiliary colour attribute value *associated* with an *object* is used in conjunction with the transparency attribute value. See under *transparency*.

3.2.14 background colour
The colour used in preparing the *drawing surface*.

3.2.15 background colour (of mapped bitmap)
A *control value* that determines the full depth value to which the background *pixels* of a *mapped bitmap* are expanded when they participate in *raster operation functions*.

3.2.16 binding
A binding is a (concrete) realization of an abstract functional specification in a programming language. This may be in terms of a procedural library binding to a standard programming language with approximately one call per function or a procedural single entry point binding of the abstract functions with all input and output parameters. Procedural single entry point bindings are based on data stream encodings; see *encoding*. In the absence of any clear context implying the contrary, "procedural binding" or "binding" may be taken to refer to a procedural library binding.

3.2.17 bitblt
BIT aligned BLock Transfer. The transfer or combination of the *pixel values* in rectangular *bitmap regions* of *bitmaps*. The CGI provides two and three operand bitblt functions where there are one or two source bitmap regions respectively, and a destination bitmap region. Arbitrary logical combinations and some arithmetic combinations are supported by *drawing mode*. See *raster operation functions*.

3.2.18 bitmap
A resource of the *Virtual Device* which may be viewed by the *client* as a rectangular array of *pixels*. This resource is available only on devices offering the functional capability defined in ISO/IEC 9636-6. Some bitmaps are inherent resources of the device; others can be defined by the client. CGI provides for full depth bitmaps, where the values assignable to the individual pixels span the total colour or grey scale capability of the device, and for mapped bitmaps, where the values assignable to a pixel are only “foreground” or “background”; it does not provide arbitrary variable depth bitmap capability. Any bitmap may be selected as the drawing bitmap which is the destination (*drawing surface*) for the *rendering* of (graphic) *objects* as well as for *raster operation functions*. Bitmaps which are displayable bitmaps may also be selected as the display bitmap.

3.2.19 bitmap region
see *region*.

3.2.20 blocked (of a queue)
A queue is said to be blocked if the *client* has taken specific action to prevent the entry of elements into it. Contrast with *overflow* where there may or may not be implicit system action to “block” a queue in overflow state.

3.2.21 boundary
The boundary of a *fill object* consists of explicit boundary portions, implicit boundary portions, and clip boundary portions. The boundary of a fill *primitive* consists of the mathematical locus of its defining perimeter. During *closed figure* construction, the mathematical locus of each line primitive used (including CONNECTING EDGE) and the boundary of each fill primitive used defines an explicit boundary portion. Implicit boundary portions are those added automatically during closed figure construction to ensure closure. (The interpretation of the significance of edge out flags in the POLYGON SET function means that it can also introduce both explicit boundary portions and implicit boundary portions.) When a fill object undergoes locus clipping, pieces of the original closed boundary lying outside the effective clip region are discarded and clip boundary portions are added to maintain closure. Drawing surface clipping does not contribute any clip boundary portions. Clip and implicit boundary portions have no associated *attribute values*. Edge portions correspond to the explicit boundary portions and are rendered in accordance with their associated edge attribute values. Contrast with *edge*, and see also *interior*, *realized edge*, and *realized interior*.

3.2.22 break action
A specific action by the *operator* to indicate that an *input operation* should be aborted. Contrast with *firing (a trigger)* and with *timeout*.

3.2.23 bundle
Virtual Devices have a conceptual resource known as bundle tables. Bundle tables contain *attribute values* determining *aspects* of *primitives* or *objects*. There are different bundle types – one each for line, marker, text, (fill object) edge, and fill (object interior) aspects. A bundle table consists of all bundles of a given type; each of which is uniquely identified by its index. For a particular primitive or object, those aspects for which an attribute value from a bundle table is to be used must have the appropriate *aspect source flag* set to BUNDLED. In this case, the bundle which contains the relevant values is specified by another attribute value, the bundle index into the table of the appropriate type.

3.2.24 cell (of CELL ARRAY or pattern)
A subspace of *VDC space* may be divided into a regular array of smaller spaces referred to as cells. Cells occur in CELL ARRAY and in *patterns* used for interior filling of *fill objects*. In general, the subspace and the individual cells may be rectangular or skewed (i.e. a parallelogram) and of arbitrary *aspect ratio*. Some devices may not provide full support for arbitrary sized or skewed cells.

3.2.25 cell (of character cell)
The subspace of *VDC space* determined by text *attribute values* (and the positioning information for the containing text *object*) which is conceptually “occupied” by a character. This is a parallelogram; but since attributes can change within a text object and proportional spacing and kerning may be relevant, character cells may not be a uniform array as in the above definition. A character cell is relevant for positioning of subsequent character cells and also for where *auxiliary colour* is *rendered* in the remainder of the cell after the rendering of the actual glyph in *foreground colour* (in this case, text colour).

3.2.26 CGI Generator
Any agent which forms an *encoding* or *binding* of CGI functions and passes this to a *CGI interpreter*. The same agent may also interpret any responses.

3.2.27 CGI Interpreter
An implementation of a CGI *Virtual Device* which receives an *encoding* or *binding* of the CGI functions, performs the appropriate actions, and provides any required responses.

3.2.28 character set
A set of displayable glyphs represented by the individual values occurring in a text *primitive*. The particular values in the strings are dependent on the character set or the alternate character set which is referenced by the character set index *attribute value* of the text primitive and are subject to character coding announcer control. Character set is concerned with (logical) alphabet and special glyphs (national character sets, Katakana, etc.) and is orthogonal to *font* which determines the typeface and style of the actual *rendering* of the glyphs (e.g. italic, bold).

3.2.29 class
see *input class* and *device class*.

3.2.30 client
Any agent invoking functions of the abstract functional capability defined in ISO/IEC 9636; i.e. any entity invoking a *procedural binding* or generating an output data stream *encoding*. Contrast with *operator* and *CGI interpreter*.

3.2.31 clip boundary portion
see *boundary*.

3.2.32 clipping
A process by which portions of (graphic) *objects* which fall outside an *associated* object clip rectangle or portions of a *rendering* of a picture or raster operation which fall outside a *drawing surface* clip rectangle are discarded. Whether or not such discarding takes place is determined by the object or drawing surface clip indicator. The object clip rectangle and indicator are object *attribute values* whereas the drawing surface clip rectangle and indicator are *control values*. See *object clipping mode*.

3.2.33 closed figure
A *fill object* constructed on the device side of the CGI interface from a series of interface functions bounded by a BEGIN FIGURE and an END FIGURE. A closed figure consists of one or more *regions*. The *boundary* of the closed figure and of each region is constructed from explicit line, fill, and GDP *primitives* and *implicit boundary portions* added as necessary to ensure the boundary remains closed. Edge *attribute values* are *associated* with individual primitives used to construct the closed figure; these are *local attributes values*. (Interior) fill attribute values are associated with the fill object as a whole; these are *global attributes values*.

3.2.34 closure point
In *closed figure* and POLYGON SET construction, when the definition of the *boundary* of a *region* has been started but is not yet complete, the first point specified for the boundary of the region is referred to as the current closure point. The occurrence of an edge flag of CLOSE VISIBLE or CLOSE INVISIBLE in POLYGON SET or the invocation of a fill primitive, NEW REGION, or END FIGURE function while in state FIGURE OPEN (for closed figures) causes the automatic creation of a line segment constituting a portion of the boundary from the last line primitive end point to the current closure point. This line segment is either an *implicit boundary portion* or an edge portion, depending on whether CONNECTING EDGE precedes the closure.

3.2.35 colour selection mode
A mode determining whether colour specification is DIRECT by means of a *(direct) colour value* or INDEXED by means of an index value into a *colour table*. The colour values in both DIRECT colour specification and appearing as entries in the colour table are specified using an RGB additive colour model. Colour selection mode is independent of the setting of any colour *ASFs*.

3.2.36 colour table
A table of *(direct) colour values* used to map from an index value to a direct colour value when the *colour selection mode* is INDEXED.

3.2.37 (direct) colour value
A 3-tuple specifying the relative amounts of red, green, and blue light which must be combined to achieve the desired colour stimulus. See *colour selection mode*.

3.2.38 compound object
A generic term for both *compound text* and *closed figures*.

3.2.39 compound text
An *object* constructed on the device side of the CGI interface from a series of interface functions bounded by a TEXT or RESTRICTED TEXT with a not-final/final flag value of NOT FINAL and an APPEND TEXT with a flag value of FINAL. (Such construction might also be started, continued, or completed by *GDPs*.) The text attributes are subdivided into those

which are *local attribute values associated* with the individual component *primitives*, and those which are *global attribute values* associated with the compound text as a whole.

3.2.40 conjugate radius pair (CRP)
A pair of radii of an ellipse such that a tangent to the ellipse at each endpoint is parallel to the other radius.

3.2.41 control functions
A function which requests some control action by the *Virtual Device*. Unqualified, this term refers to the general device and coordinate space control defined in ISO/IEC 9636-2 or the *graphic object pipeline* controls defined in ISO/IEC 9636-3. For other controls, such as raster and input, the term will always be appropriately qualified.

3.2.42 control value
A *modal* value which affects the operation of processes in the *graphic object pipeline* which are *downstream* of the *segment storage*. Contrast with *attribute value*. (Object) attribute values and control values are disjoint sets. Note, however, that *segment attribute values* which do not affect the operation of processes downstream are actually control values in the sense of this definition. Attribute values associated with an object cannot be modified and therefore cannot be a source of *dynamic modification* of a picture. Conversely, control values may be modified at any time and therefore are a source of dynamic modification.

3.2.43 deferral mode
A *state list* entry for the device which specifies the acceptable delay between the invocation of a CGI function and the final realization of its effect. Possible values are ASTI (At Some TIme), BNI (Before Next Interaction) and ASAP (As Soon As Possible).

3.2.44 description table
One of a set of tables giving "permanent" as opposed to "dynamic" characteristics of the *Virtual Device*. Contrast with *state list*. Description tables may in fact change by means outside the CGI, such as device reconfiguration by the operating system or, in some environments, by explicit *operator* or resource manager action (e.g. windowing systems). All information in the description table, including that indicating whether other information is variable, is inquirable.

3.2.45 device class
CGI *Virtual Devices* are classified, based on input and output characteristics, as INPUT, OUTPUT, or OUTIN. Devices of class OUTPUT have one active *drawing surface* which may or may not be the same as the active *display surface* depending on the device. Devices offering the functional capability defined in ISO/IEC 9636-6 may have several drawing surfaces. Devices of class INPUT have one or more *logical input devices* (LIDs), each with its own *measure*, but have no drawing or display surface. Devices of class OUTIN combine the drawing and display surfaces of the OUTPUT class and the LID facilities of the INPUT class.

3.2.46 device coordinates (DC)
A position specified by means of a device dependent coordinate system which is related to, but not necessarily identical with, the physical coordinate system by which a physical position on a display or input surface, such as a screen or tablet, is specified. Device coordinates may be in any metric – not necessarily uniform and with arbitrary origin. There may well be a final mapping outside the CGI from the "logical" device coordinates used within the CGI to the actual physical coordinates of a device (e.g. in a windowing system or an implementation supporting several CGI instances, i.e. different logical output devices, on the same physical resources).

3.2.47 device driver (implementation of the CGI)
A procedurally bound (see *binding*) implementation which emits no CGI data stream encoding and which operates the physical device(s) by whatever means are necessary. This includes such examples as sending and receiving a non-CGI data stream, manipulating a frame buffer in host memory, and accessing control registers in an I/O bus peripheral.

3.2.48 display surface
The physical surface on which graphic information is presented to the *operator*. See *drawing surface*.

3.2.49 downstream
In the direction from *client* towards (virtual or real) device or *operator*.

3.2.50 drawing mode
This determines how colour or grey scale *pixel* values are combined during *rendering* of *graphic objects* and while performing *raster operation functions*. Drawing mode is either an *attribute value* of a particular graphic object, in which case it determines how the rendered information from that particular object is mixed into the *drawing surface*, or it is a parameter of a raster operation function, in which case it determines how the information from the one or two source *bitmap regions* is mixed into the destination bitmap region of the drawing surface. Drawing modes are composite values, the first component catering for

logical combinations (BOOLEANOP), arithmetic combinations (ADDITIVEOP), and comparative combinations (COMPARITIVEOP) classes and the second component specifying a particular combination within the class.

3.2.51 drawing surface
The conceptual surface of the device on which the graphic picture is rendered or (for devices implementing the functional capability of ISO/IEC 9636-6) the currently selected *drawing bitmap* onto which both *graphic objects* are rendered and *raster operation functions* have their destination *bitmap region* specified. See *display surface.*

3.2.52 dynamic modification
Change of a *control value* or a *segment attribute value* which may result in the rendered picture on the *drawing surface* being in some respects no longer a valid *rendering* of the content of *segment storage*. Note that *attribute values associated* with the *objects* in segments cannot be changed; so this is not a possible source of dynamic modification. Devices have different capabilities with respect to handling such dynamic modification. This capability is recorded for each possible cause of dynamic modification in the *description tables* under the entries dynamic modification accepted for <type of change>. Each such entry can have the values IMM (IMMediate), CBS (Can Be Simulated), or IRG (Implicit ReGeneration).

3.2.53 echoing
Echoing is the term used to describe feedback to the *operator* of the *measure* of a *logical input device.* Echoing may be provided by means of various styles referred to as echo types. The echo for a particular LID appears in a rectangular subspace referred to as the echo area on the *display surface* (as opposed to the *drawing surface*) of an OUTPUT or OUTIN device. Echoing is either automatically performed by the OUTIN device when requested by the *client* or, in the case of remote echoing on a device other than the one providing the *measure*, should be explicitly performed by the client using echo output facilities.

3.2.54 echo request input
A method of input by which the *client* can obtain the current *measure* of a *logical input device* in order to perform *remote echoing* on a different *Virtual Device*. This method returns data when the measure changes or on the occurrence of a trigger *firing*.

3.2.55 edge
The edge of a *fill object* consists of portions, each portion being initially the mathematical locus of a line *primitive* or the *boundary* of a fill primitive. The edge does not include any *implicit boundary portions* added during *closed figure* construction to maintain closure. Each portion of the edge has *associated* with it edge *attribute values* which can be different for different portions. After locus clipping, entire portions or pieces of portions lying outside the associated *clip rectangle* are discarded. *Clip boundary portions* are added to maintain closure; but these, like implicit boundary portions, are not edge portions. When the (locus clipped) edge is rendered, the remaining pieces of a portion (which may now be disjoint) are rendered using the edge attribute values associated with the portion. These may then be shape clipped to produce the final rendering of the edge. See also *boundary*, *interior*, *realized edge*, and *realized interior*.

3.2.56 (data stream) encoding
A (concrete) representation of an abstract functional specification in the form of a sequence of data items (e.g. characters, 8-bit bytes, etc.). The identification of each function and its parameter values are encoded to produce a *function representation.* Similarly, responses from an interpreter of the data stream are also encoded. Contrast with *binding*.

3.2.57 error class
Errors which may occur when using CGI are classified into one of nine different error classes. For some of these, an error report will be entered into the *error queue*. The functional description specifies error reactions for the errors it specifically defines; but, non-standard errors are also provided for in the classification scheme. There are specific mechanisms for switching error detection and error reporting on or off on a per class basis.

3.2.58 error queue
A first-in-first-out (FIFO) queue, each entry of which consists of an error identifier and an error report. This queue can *overflow*. It cannot be *blocked* per se since a control equivalent to "blocking" is provided on a per class basis.

3.2.59 escape function
CGI provides an extension mechanism in the form of escape functions. There are three distinct categories: *generalized drawing primitive (GDP)*, ESCAPE and GET ESCAPE. GDPs provide for non-standard *primitives* and, like the standard *primitive functions*, are never *soliciting functions*. For those escapes which do not generate primitives, those which are soliciting functions are GET ESCAPEs and the remainder are simply ESCAPEs. Escape functions in all three categories may be registered in the International Registry of Graphical Items or are private to an implementation.

3.2.60 event input
A method of input in which an *input operation* (event) occurs asynchronously as a result of some *operator* action (see *firing* and *break action*) or *timeout* and for which an entry referred to as an event report is made in the *event queue*. *Logical input devices* have to be enabled for event input before they can contribute events to this queue.

3.2.61 event queue
There is a single event queue for all *logical input devices*. It is a first-in-first-out (FIFO) queue which has to be explicitly read by the *client*. The client does not receive any notification that an entry has been made. The queue can be *blocked* and it can also *overflow*. Once overflow has occurred, it is implicitly blocked until the client takes remedial action. Entries are made in the queue in time sequence of events which occur on those LIDs enabled for *event input*. Each entry is an *event report* which includes the LID class and index, the trigger index of the trigger which caused the event, a timestamp, the measure value, and the measure validity status.

3.2.62 explicit boundary portion
see *boundary*.

3.2.63 external functions
An external function is one which is not concerned with either the content or the control of the graphic information in the *Virtual Device*. MESSAGE is one such function defined in ISO/IEC 9636; but, some *escape functions* may also fall into the external category.

3.2.64 fill object
A general term for all (graphic) *objects* which have a (closed) boundary, an *edge*, and an interior. Fill objects are created from fill *primitives* and the *association* of edge attribute and interior (fill) *attribute values* or they are constructed as *closed figures* from line and fill primitives and *implicit boundary portions* generated to ensure closure. The *boundary* of a fill object is always closed even after *clipping* and the boundary encloses the *interior* (or clipped interior). The boundary may consist of several closed *region* boundaries; regions may be disjoint, intersect with, or be wholly contained within other regions. The definition of the interior of a fill object is determined by the *parity fill algorithm*. In general, the edge is rendered on top of the rendered interior; the boundary is not rendered. See also *realized edge* and *realized interior*.

3.2.65 firing (a trigger)
An action by the *operator* which indicates to a *logical input device* that something significant has occurred. After such an action, the *trigger* is said to have fired. A trigger cannot be fired unless it has been *armed*.

3.2.66 font
A text primitive *attribute value* which is orthogonal to *character set* and determines the style of *rendering* of the individual glyphs.

3.2.67 foreground colour (of objects)
The colour, determined by an *associated attribute value*, in which primitive components of *objects* or object *interiors* are rendered onto the *drawing surface*. If a line type is not continuous or a fill style is not solid, then *auxiliary colour* is used for the "gaps". For text, foreground colour is used for the glyphs and auxiliary colour for the rest of the character cell.

3.2.68 foreground colour (of mapped bitmaps)
A *control value* which determines the full depth value to which the foreground *pixels* of a *mapped bitmap* are expanded when they participate in *raster operation functions*.

3.2.69 foundation profile
One of the basic *Profiles* defined in ISO/IEC 9636, (<input class> INPUT, 2-WAY OUTPUT, 1-WAY OUTPUT). All other Profiles should be defined to contain at least one of these Foundation Profiles.

3.2.70 (CGI) function
The unit of communication between the *client* and the *Virtual Device*. This functional specification defines functions abstractly. *Bindings* and *encodings* will specify, for each function, a function representation – a concrete representation of the function and its parameters or of the request and the response. Functions are classified in various ways – see *attribute function*, *control function*, *external function*, *graphic function*, *inquiry function*, *primitive function*, *raster operation function*, and *soliciting function*. All functions request actions of various kinds from the Virtual Device. Soliciting functions require a response from the device.

3.2.71 functionality
A contraction of "functional capability"; used to refer broadly to types of behaviour possible in a given context.

3.2.72 generator
see *CGI generator*.

3.2.73 generalized drawing primitive (GDP)
An *escape function* which generates a *primitive* or contributes to the construction of a *graphic object*.

3.2.74 global attribute value
An *attribute value* which is *associated* with a (compound graphic) *object* as a whole, rather than with the individual *primitives* from which the object is constructed, is termed a global attribute value. Contrast with *local attribute value*.

3.2.75 graphic function
This is a general term which covers all functions whose action is concerned with the content or control of the graphic information in the *Virtual Device*. Compare *external function* or *inquiry function*. Graphic functions include *attribute functions*, *control functions*, *primitive functions*, and *raster operation functions*.

3.2.76 graphic object
Primitives, when they have had all their applicable *attribute values* associated with them, are referred to as (graphic) objects. There are special construction facilities which allow a *(compound) object* to be formed from several primitives with *local attribute values* and *global attribute values* associated with individual contributing primitives or with the object as a whole respectively.

3.2.77 graphic object pipeline
An abstraction of the collection of conceptual processes through which graphic (i.e. primitives and objects) information passes en route from the *client* to the *display surface* where it can be observed by the *operator*. *Raster operation functions* bypass most of this pipeline. Contrast with <u>graphic pipeline</u> which is used as an abstraction for all CGI information flowing (both ways) between the client and the operator.

3.2.78 hatch style
A particular representation for the *interior* of a *fill object* using fill style HATCH. A hatch style consists of one or more sets of (semi) regularly spaced lines in different directions. The exact appearance of hatch lines is implementation dependent.

3.2.79 implementation-dependence
Where ISO/IEC 9636 recognizes that implementation variability is necessary, but cannot define a set of allowed behaviours, then the potential of variability is termed implementation-dependence. Contrast with *allowed latitude*.

3.2.80 implicit boundary portion
see *boundary*.

3.2.81 implicit closure
The process of ensuring that the *boundary* of a *fill object* remains closed even if the *client* does not explicitly create a closed boundary. Implicit closure is achieved by adding *implicit boundary portions* as necessary during the construction process.

3.2.82 implicit segment regeneration mode
A mode which determines whether automatic regeneration of the picture may occur or not (ALLOWED, SUPPRESSED, or UQUM) when some change of a *control value*, *segment attribute value*, or the contents of *segment storage* invalidates the current picture as an accurate *rendering* of the *objects* in segment storage. See *dynamic modification* and *quick update method*.

3.2.83 indexed colour
see *colour selection mode*.

3.2.84 input class
Logical input devices are classified according to the type of data value (*measure*) they provide for return to the *client*. The classes defined are: LOCATOR, STROKE, VALUATOR, CHOICE, PICK, STRING, RASTER, or GENERAL. These are defined in detail in ISO/IEC 9636-5.

3.2.85 input operation
A general term for any sequence of *operator* and/or input device measure setting actions which makes an input value (*measure*) available for return to the *client*. The exact sequence of actions and the method of return of the measure to the client are determined by the input method. See *request input*, *sample input*, and *event input*.

3.2.86 input surface coordinate (ISC) space
For input classes LOCATOR, STROKE, and PICK, there is a Cartesian coordinate system from which input points are specified by the *operator*. The space is referred to as the input surface coordinate (ISC) space. It is analogous to the *device coordinate space* for a *drawing surface*. A bounded rectangle within ISC space is termed the <u>input surface</u>. This is a subset of

ISC space for which the physical device is capable of producing point locations. The *client* may define the mapping from ISC space to VDC space; this is termed the ISC-to-VDC Mapping. Each LID (for which it is relevant) may have its own ISC-to-VDC Mapping.

3.2.87 inquiry function
A function which provides the *client* with access to information held in *description tables* or *state lists* of the *Virtual Device.* All inquiry functions are *soliciting functions.*

3.2.88 interior (of a fill object)
The interior of a *fill object* is that part of *VDC space* "within" the *boundary* – "within" is defined by the *parity fill algorithm.* The interior may consist of several distinct or abutting subspaces. After *clipping*, the clipped interior is the set (which may now be a different set) of subspaces within the clipped boundary. See also *realized interior.*

3.2.89 interpreter
See *CGI interpreter.*

3.2.90 local attribute value
An *attribute value* which is *associated* with each individual *primitive* used in the construction of a (compound graphic) *object* rather than with the complete object is termed a local attribute value (Contrast with *global attribute value*).

3.2.91 logical input device (LID)
A logical component of an OUTIN or an INPUT device. LIDs are classified into one of the various *input classes.* A *Virtual Device* may have several LIDs which consist of a *measure* (input value), state information, and possibly associated *triggers.*

3.2.92 mapped bitmap
see *bitmap.*

3.2.93 measure
The term used for the abstract component of a *logical input device* representing the input value for that device which will be returned to the *client* on receipt of the appropriate *soliciting function* either directly from the LID or, in the case of *event input*, from the *event queue* where it will have been placed by the LID.

3.2.94 message
A communication direct from the *client* to the *operator* which leaves no trace of its passage through any pipeline and is always displayed to the operator independent of whether the current display reflects the current *drawing surface.* Messages are character strings which are not subject to the *character set* or character coding announcer control which apply to text.

3.2.95 modal value (modally set value)
A value is referred to as modal if it is set by a specific function and recorded in a *state list.* That value is then applicable, whenever relevant, to subsequent functions until such time as it is reset by another use of the specific setting function or, in certain cases, it is reset to default by the *Virtual Device* as a result of VDC TYPE changes. Contrast with parameter value, which is always local to a particular function invocation.

3.2.96 non-dissociable trigger
A *trigger* whose association with a *logical input device* is an inherent characteristic of the device and cannot be overridden by the *client.* See *trigger association.*

3.2.97 non-retained data
Any rendered object information resulting from *objects* passed down the *graphic object pipeline* which were not stored in *segments* (i.e. while no segment was open) as well as all information resulting from *raster operation functions* which is on the *drawing surface* is referred to as non-retained data. In other words, any information on the drawing surface which cannot be reproduced by the sequence – PREPARE DRAWING SURFACE, set all segment visibility attributes to VISIBLE, REDRAW ALL SEGMENTS – is non-retained data.

3.2.98 object
see *graphic object.*

3.2.99 object clipping mode
A *control value* which determines how the process of *clipping* (graphic) *objects* is conducted. This is either on the mathematical locus of the object before the application of *rendering attribute values* (LOCUS), on a rendered shape after their application (SHAPE), or on first the one and subsequently the other (LOCUS THEN SHAPE). The three settings can all give different effects. There are separate clipping mode control values for each of lines, markers, and edges.

3.2.100 one-way (1-way) output
This describes the situation where the *client* never uses any *soliciting functions*. All communication across the CGI is from client to *Virtual Device* and there is never any response from device to client.

3.2.101 operator
The human user of the CGI *Virtual Device* realization. Contrast with *client*. The operator may interact with the client by observing the realized output and by providing input (related or unrelated to the output), responding to *messages*, etc.

3.2.102 overflow (of a queue)
Some implementations may limit the space of a queue to some pre-set number of entries (fixed or indeterminate depending on whether the size of the entry is fixed). Other implementations may use dynamic allocation from a resource pool used for other purposes as well as the queue entries. When queue space is exhausted, the next attempted entry causes the queue to overflow. Overflow of the *event queue* results in the latest entry always being overwritten with an "error reports lost" error which includes an updated count of entries lost. Overflow of the *input queue* results in the queue becoming implicitly *blocked* and it remains so until the *client* takes remedial action.

3.2.103 parity fill algorithm
This algorithm determines the *interior* of a *fill object* as that set of points not lying on the *boundary* for which all lines to infinity cross the boundary an odd number of times.

3.2.104 pattern (of fill pattern)
One of the styles for *interior* fill of a *fill object* is pattern fill. A pattern is a two-dimensional uniform array of *cells* each specifiable as having a different colour or grey scale value. (Compare with CELL ARRAY which is similar.) Patterns are selected by the *client* by means of the pattern index (object) *attribute value*.

3.2.105 pattern (of bitmap region)
A *bitmap region* used as one of the source parameters in a 3-operand *bitblt* or specified by the FILL BITMAP function for rendering with interior style BITMAP is referred to as a pattern. In either case, the pattern (bitmap region) is replicated as a tile. For the 3-operand bitblt case, the pattern is replicated to build a region matching the size of the other source region. In the bitmap fill case, the pattern is replicated as necessary to fill the *interior*.

3.2.106 picking
Conceptually, this is an operation which allows the *operator* to point at something on the *display surface* after which the *Virtual Device* responds to the *client* with an indication of what the operator pointed to. This feature works in conjunction with segment capability. The response lists data structures which indicate certain *segments* which would be drawn sufficiently close to where the operator pointed. The precise definition of the picking behaviour is in terms of the contents of *segment storage*, and it is the responsibility of the client to make sure that the visible display is relevant for the operator.

3.2.107 pixel
In the context of raster devices, the smallest element of the *drawing surface* of a *Virtual Device* to which an individual colour or grey scale value can be assigned. The assigned value is a device dependent pixel value and *rendering* of *graphic objects* and all *raster operation functions* act directly on pixel values so combination characteristics may vary from device to device. Pixels may lie on or between addressable device points as specified by an entry in a *description table*.

3.2.108 primitive
A primitive gives the geometric specification for some portion of a picture. Primitives are passed across the CGI interface by the *client*, using *primitive functions*. Primitives do not carry any *attribute values*. These are *associated* with them later in the *graphic object pipeline* to form *objects*.

3.2.109 primitive function
A primitive function is one whose primary action is to generate a *primitive*. Primitive functions are classified according to *primitive type*.

3.2.110 primitive type
One of "line", "marker", "text", "fill", or "image". A primitive type is an adjective which can apply to "function", "primitive", "object", "aspect", or "attribute". (E.g. "fill function".) Note the anomalous word "edge" which can be used as a type for "aspect" or "attribute", even though "edge" is not itself a primitive type. *Generalized drawing primitives* may generate *primitives* of various types and be subject to *aspects* associated with those types.

3.2.111 profile
A specification of a particular collection of CGI functions and resource requirements as well as any relevant constraints or *escapes* required to meet the needs of a particular set of users, sometimes referred to as a constituency. There are

Foundation Profiles and Constituency Profiles. Some Constituency Profiles are defined in ISO/IEC 9636; others may be registered.

3.2.112 prompting
An action, enabled or disabled by the *client*, then taken internally by a *logical input device* at the beginning of an *input operation* to draw the *operator's* attention to the fact that an input action is expected. In a *remote echoing* situation, prompting has to be specifically performed by the client. Prompting may take various styles known as prompt types selectable by the client.

3.2.113 quick update method
In the context of *implicit segment regeneration*, implementation-dependent methods of modifying the picture, in response to certain changes, are permitted for the sake of speeding up the display process. Such methods need not be completely faithful to the segment model. For example, *segments* may be "erased" by "drawing" them in background colour; segment display may be done without regard to display priorities. Such methods may be employed by an implementation only when explicitly allowed by the *client* when *implicit segment regeneration mode* is UQUM (Use Quick Update Method), and only for changes whose *dynamic modification accepted for* entry is CBS (Can Be Simulated).

3.2.114 raster operation functions
Functions which request operations directly on the *pixel values* in *bitmaps* on devices offering the functional capability defined in ISO/IEC 9636-6 are referred to as raster operation functions. They consist of the PIXEL ARRAY functions and the *bitblt* functions. The specific action of these functions is controlled by parameters rather than *state list* values for relevant factors such as *transparency* and *drawing mode*.

3.2.115 realized edge
This is what is rendered after the associated *attribute values* (width, type, colour, etc.) have been applied to the *explicit boundary portion* of a *fill object*. The exact impact of clipping is controlled by the edge *object clipping mode*. The realized edge is rendered "in front of" the *realized interior* and may therefore obscure or modify part of that interior. See *boundary*.

3.2.116 realized interior
This is what is rendered by applying associated fill *attribute values* and any applicable object clipping to the *interior* of a *fill object*. For all fill styles except HOLLOW, the interior is realized first. The *edge* is then realized and overlaid or mixed depending on *auxiliary colour*, *transparency*, and *drawing mode* (or on device overwrite capability). For interior style HOLLOW, the *boundary*, including the *clip boundary portion*, is rendered using the associated fill colour and implementation-dependent "boundary attributes" and the realized edge is then overlaid. This is the only case when the boundary is rendered; for all other interior styles, implicit boundary portions and clip boundary portions are not rendered.

3.2.117 regeneration
see *implicit segment regeneration mode*.

3.2.118 region (of a fill object)
Each *fill object* consists of one or more regions. Each region has its own closed *boundary* which may have been formed using *implicit closure* and modified by object *clipping*. Regions may have self-intersecting boundaries and may overlap or abut other regions. Regions are formed explicitly by use of the NEW REGION function or implicitly when a *fill primitive* is used during *closed figure* construction. Each closed polygon of a POLYGON SET is a region. The only fill objects which may have multiple regions are closed figures, POLYGON SETs and possibly GDPs.

3.2.119 region (of a bitmap)
A rectangular subset of a *bitmap* used as a source or a destination in *raster operation functions* or as a *pattern* for interior style BITMAP.

3.2.120 remote echoing
In situations in which a *Virtual Device* realization does not have the capability to do *echoing* (locally) within the device (e.g. when the input device and the output device are supported by different CGI instances), the *client* may simulate the echoing process by making use of some input method (possibly *echo request input*) on one *logical input device* and echo output on a different device. Such echoing performed explicitly by client action is referred to as remote echoing. Echo output facilities also cater for remote *acknowledgement* and remote *prompting*.

3.2.121 rendering
The collection of processes (*downstream* of *segment storage* in the *graphic object pipeline*) which convert the geometric specification of an *object* and its associated *attribute values* to the form in which it will be presented on the *drawing surface*. The term generally covers transformations, application of attribute values, and *clipping*, as well as processes specifically

shown as locus, shape, and physical rendering. All processing concerned with rendering may involve varying degrees of approximation which may be device-dependent and generally will not be inquirable.

3.2.122 request input
A method of input in which the *measure* value of a *logical input device* is returned synchronously to the *client* when the *operator* fires a trigger (see *firing*). All functions concerned with request input are *soliciting functions* and, having interpreted one, the *Virtual Device* will suspend all further interpretation until the operator has taken action or a *timeout* has occurred and a response is delivered to the client. Contrast with *event input*, *sample input*, and *echo request input*.

3.2.123 sample input
A method of input in which the current *measure* of a *logical input device* is returned immediately to the *client* when an appropriate input *soliciting function* is invoked. No *operator* action is required. Contrast with *request input* and *event input*.

3.2.124 segment
A grouping of (graphic) *objects* retained in the *Virtual Device* in a virtual resource referred to as segment storage. The *client* controls the existence and content of each segment. (See *segment definition*.) Each segment has its own unique, but changeable, segment identifier and carries an *associated* set of segment attribute values which determines its detectability, display priority, highlighting, pick priority, segment transformation, and visibility. Segment attribute values apply to the entire contents of the segment. These values may be changed by the client at any time and are thus a potential source of *dynamic modification*.

3.2.125 segment definition
The process of constructing the segment's contents (as opposed to setting its *segment attribute values*). Only the open segment can be undergoing definition. The process is initiated by CREATE SEGMENT and terminated by CLOSE SEGMENT though an existing segment can be subsequently reopened and its definition continued (sometimes referred to as appending to a segment). When a segment is open, *objects* flowing along the *graphic object pipeline* are entered into the open segment, thus defining its content.

3.2.126 skewed
Non-perpendicular. The term is used in several contexts – to refer to text when the orientation vectors are non-perpendicular, to refer to *cells* of CELL ARRAY or *patterns* when the defining points form a non-rectangular parallelogram, and so on. Transformations may also produce skewing.

3.2.127 soliciting function
Any function which is defined to have return parameters, i.e. which requires a response from the *Virtual Device*, is termed a soliciting function. The CGI is serial synchronous. This means that the effect of any function is (conceptually) completed before the execution of the next function is begun. Since the generation of the response is part of the effect of a soliciting function, the *client* may use soliciting functions to force synchronization.

3.2.128 specification mode
Modes which determine whether widths or sizes are specified in VDC values (and therefore transform geometrically) or in "scaled" values (relative to some fixed DC value) applied after transformation. Specification modes are associated with line width, edge width, and marker size.

3.2.129 state list
One of a set of lists giving the current operating state of the *Virtual Device*. (Contrast with *description table*.) State list information is continually updated based on *client* action and is all inquirable.

3.2.130 timeout
The termination of an *input operation* due to the expiry of a specified waiting time before the *operator* has taken any explicit action to terminate the operation. See *firing* (a trigger) and *break*.

3.2.131 transparency
This concept arises in the context of output *primitive functions* and in *raster operation functions*. For output primitives, the transparency attribute can have either of the values TRANSPARENT or OPAQUE. If the transparency attribute value *associated* with an *object* being *rendered* is TRANSPARENT, the object will not be drawn in front of previously rendered objects in places where there are "holes" in the newly rendered object. Examples of such "holes" include gaps in dashed lines or edges, the spaces between lines in a hatched fill area, and the background of a character cell. If the implementation provides support for the *drawing mode* attribute, then the drawing mode attribute value associated with an object controls the manner in which what is rendered is combined with what has already been rendered for any other object. When transparency is OPAQUE, the "holes" are rendered using the *auxiliary colour* attribute value associated with the object, and they are rendered over the existing rendered objects, again possibly subject to drawing mode. For raster operation functions, transparency is

specified as an input parameter and can have values TRANSPARENT and OPAQUE. If the transparency parameter in a raster operation function is TRANSPARENT, then *pixels* in the source *bitmap* which have (or expand to, for a *mapped bitmap*) the same value as the Transparent Colour entry in the Raster State List will not be combined with corresponding pixels in the destination bitmap, leaving them unchanged. When the transparency parameter is OPAQUE, source pixel values are always combined with the destination pixels according to the drawing mode parameter of the function.

3.2.132 trigger
A possible component of some *logical input device* which provides a mechanism by which the *operator* can indicate to the device that something significant has occurred. A trigger cannot be used (*fired*) to provide such an indication unless it has first been *armed*. Triggers are only relevant to *request input* and *event input*. Some triggers are inherent to the logical input device, see *non-dissociable triggers*, other triggers may be associated with one or more logical input devices by the *client*. See *non-dissociable triggers* and *trigger association*.

3.2.133 trigger association
The process of establishing a relation (dissociation is breaking this relation) between one or more *triggers* and one or more *logical input devices*. While the relation exists, a trigger is said to be associated with a LID. Some triggers are *non-dissociable triggers*.

3.2.134 upstream
In the direction from (virtual or real) device or *operator* towards the *client*.

3.2.135 (device) viewport
A rectangular region of *device coordinate space* into which the *VDC extent* (window) is mapped. The requested device viewport is that requested by the *client* while the effective viewport is the one the implementation provides which will be as close to the requested as is possible but not necessarily identical with it. The current specification of the effective viewport is a single *state list* entry in the Control State List. For devices which offer the functional capability defined in ISO/IEC 9636-6, each *bitmap* has a viewport specification in its individual state list, which will be used for the current specification when that bitmap is selected as the *drawing bitmap*.

3.2.136 virtual device
An idealized conceptual graphic device that presents a set of graphics capabilities (defined by means of an abstract functional specification) to the *client* in a way independent of its actual realization on one or more physical devices. Capitalization is used, (CGI) Virtual Device, to denote a virtual device as defined in ISO/IEC 9636. Virtual devices are classified according to their input and output characteristics. See *device class*.

3.2.137 virtual device coordinate (VDC) space
Graphic information flowing across the CGI from the *client* to the *Virtual Device* is positioned in a virtual device coordinate (VDC) space. This is a two-dimensional Cartesian space of infinite extent and precision. An implementation will only realize a finite subset of this conceptually infinite space known as the VDC range. Graphic *primitives* are positioned by VDC points, each point being an xy-coordinate pair within the VDC range. When the graphic information is subsequently realized (see *rendering*), a rectangular region of VDC space within the VDC range known as the VDC extent (but sometimes referred to as a window) is mapped onto a *viewport* (more accurately, the effective viewport) in *device coordinate space*. This mapping is referred to as the VDC-to-Device Mapping.

3.3 Abbreviations

For the purposes of all parts of ISO/IEC 9636, the following abbreviations apply.

ASAP	As Soon As Possible (deferral mode setting)
ASF	Aspect Source Flag
ASTI	At Some TIme (deferral mode setting)
CBS	Can Be Simulated (dynamic modification acceptance)
CRP	Conjugate Radius Pair
CGI	Computer Graphics Interface
CGM	Computer Graphics Metafile

Abbreviations

DC	Device Coordinates
DSCRECT	Drawing Surface Clipping RECTangle
FIFO	First In First Out (queueing discipline)
GDP	Generalized Drawing Primitive
GKS	Graphical Kernel System
ID	IDentifier
IMM	IMMediate (dynamic modification acceptance)
IRG	Implicit ReGeneration (dynamic modification acceptance)
ISO	International Organization for Standardization
LID	Logical Input Device
OSI	Open Systems Interconnection
RGB	Red, Green, Blue (colour model)
UQUM	Use Quick Update Methods
VDC	Virtual Device Coordinates
VC	Viewport Coordinates
VP	Viewport Point

4 Reference models

4.1 Introduction

This clause describes reference models which show a variety of relationships that the CGI defines between its client and other parts of a graphics system, referred to as the target. These relationships are described using a general set of possible client/target CGI configurations. A number of client and target generic types are also described. The clause concludes with a description of the relationship of CGI to other related standards.

4.2 Model for CGI client/target relationships

A CGI interface mediates a relationship between a client program and some target. The concept "target" covers a general class of entities, of which a typical graphics device is one example. A general presentation of the full variety of possible types of targets appears later in this clause.

4.2.1 CGI configurations diagram

Figure 1 portrays a model of the relationships that may exist between a CGI client and target. The diagram permits recursive interpretation of its components (i.e. a target may itself be a CGI client) which leads to a variety of potentially sophisticated configurations (see annex E). Since the CGI is a single-device interface, figure 1 depicts relationships between a client and a single CGI Virtual Device. A client is permitted to be in communication with multiple CGI Virtual Device implementations, but the behaviour of any one CGI is not affected by operations with other such CGIs.

4.2.1.1 Basic generator/interpreter configuration.

The basic configuration of the model is the Generator/Interpreter configuration depicted in configuration (a) of figure 1. A client program uses a procedurally bound CGI Generator subroutine library to create a CGI data stream which is transmitted to a CGI Interpreter. The interface between the client and the Generator is device-independent. It is labelled as "target independent" since the target concept is more general than that of "device". Further downstream, the CGI Interpreter communicates with the target by unspecified means.

Configurations (b), (c), (d), and (e) of figure 1 correspond to simpler configurations derived from this basic configuration.

Configurations (a), (c), and (d) present the same procedurally bound CGI interface to a client. However, the existence of a CGI data stream downstream is of potential significance. In a generator implementation, the client may control encoding techniques in the Generator in order to optimize use of I/O connections over which the CGI data stream flows. For example, some applications may require 32-bit integer virtual device coordinates, while others may only need 16-bit integer VDCs. A Generator should recognize type and precision requirement functions and maintain state information to control the types and precisions for the data that it encodes into the data stream. In a Driver implementation of CGI, it may be possible to ignore precision requirement functions.

4.2.1.2 Generating clients

In configuration (b) of figure 1, the role of the CGI data stream generator has been merged with the client program. In this case, the client generates the CGI data stream itself, without necessarily using a CGI procedural binding. The target independent interface is a standard CGI data stream encoding.

4.2.1.3 Callable drivers

In configuration (c), the Generator and Interpreter have been combined to form a "Driver". In this case, the CGI data stream no longer exists, and the subroutine library communicates directly with the target by whatever means are available.

This configuration, while it is conforming, is deprecated. This illustration is provided for completeness.

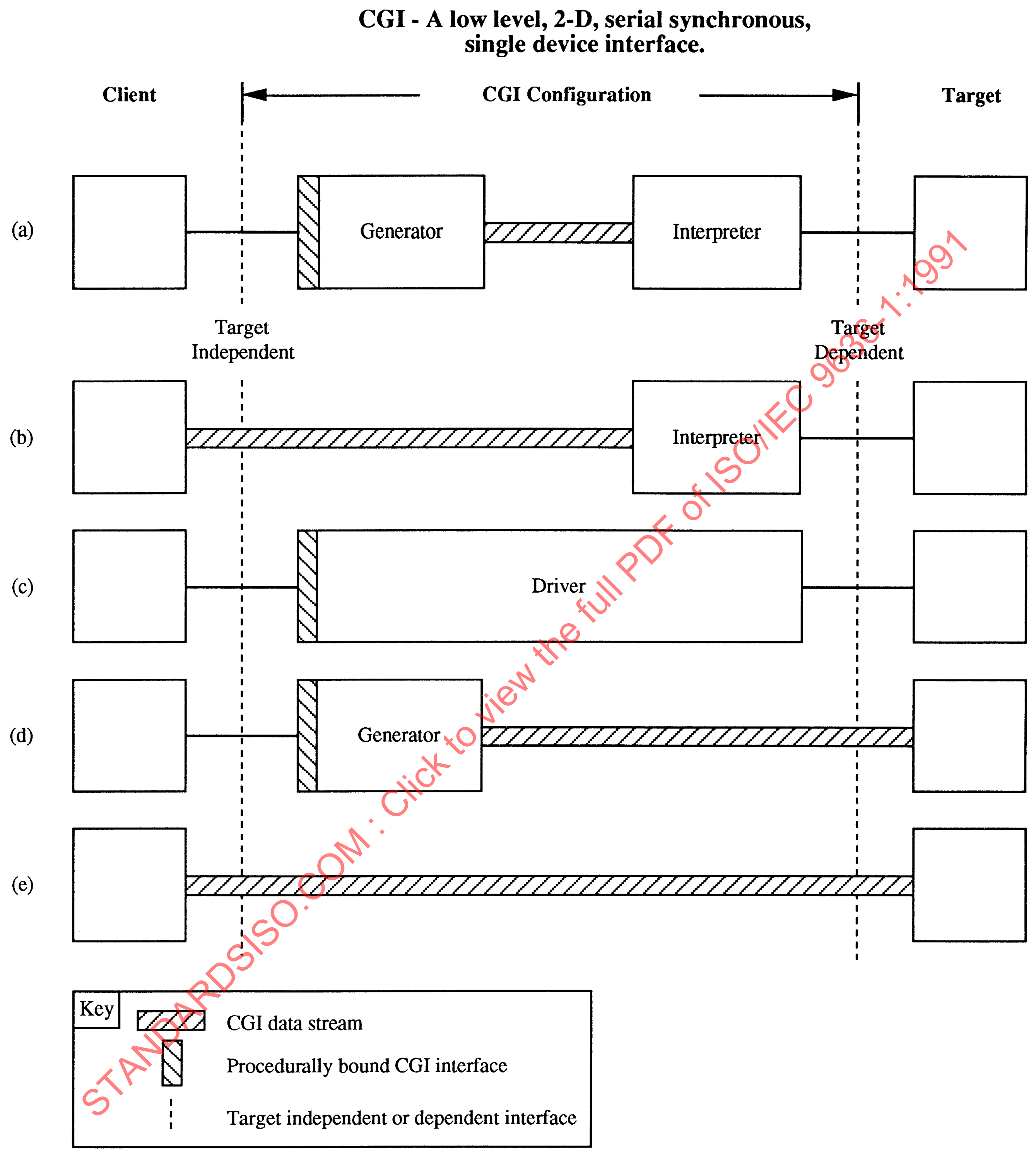

Figure 1 – CGI Client/Target Configurations

4.2.1.4 CGI targets

In configurations (d) and (e), the CGI interpreter has been combined with the target. In particular, the target may be an implementation of a CGI Virtual Device.

Configuration (e) of figure 1 depicts a generating client sending a CGI data stream directly to a target which is itself a CGI Interpreter. This configuration corresponds to the simplest model of the relationship between a client and a CGI Virtual Device.

4.2.2 Types of clients

The need to provide efficient support for GKS implementations and CGM interpreters has been a fundamental design goal for the development of the CGI. In particular, there is an extremely close match between the set of functions naturally required by these clients and those defined in the CGI.

4.2.2.1 GKS implementations

The CGI is designed to provide a good functional match between GKS and the CGI at the GKS workstation interface. When a CGI device is used to implement a GKS workstation, a substantial increase in performance may be possible on the computer hosting the GKS implementation by offloading most of the graphics workload into the CGI Virtual Device.

4.2.2.2 CGM interpreters

The functional match between ISO 8632 (CGM) and CGI is such that CGM elements in a picture body may be passed directly to a CGI Interpreter that handles the same encoding without the CGM Interpreter needing to have any knowledge of the effect of the functions.

4.2.2.3 GKS-3D and PHIGS

Additional types of clients for CGI include implementations of other graphics standards, such as ISO/IEC 9582 (PHIGS) and ISO 8805 (GKS-3D). Since the CGI is a standard for two-dimensional graphics, such use requires additional work upstream of the CGI interface. The CGI still provides a standard target independent interface for accessing hardcopy devices, metafiles, and numerous display devices of lower capability than those designed for three-dimensional graphics work.

4.2.2.4 Proprietary packages

Proprietary software packages may be developed (or adapted) to utilize the CGI. The CGI provides a standard target independent interface which greatly increases the flexibility and adaptability of the vendor's software package.

4.2.2.5 CGI implementations

One possible client is another CGI implementation. Such a CGI client may provide procedural or data stream interfaces upstream and downstream. Two sample reasons for the existence of such CGI clients are encoding conversion and function simulation.

An encoding conversion “filter” interprets a data stream from upstream and produces an equivalent data stream in a different encoding downstream. Since CGI data streams are often bidirectional, an inverse mapping may be necessary for any returning CGI messages.

Function simulation may be required when the downstream CGI lacks certain functional capability expected by the simulator's client. In such cases, it is possible for the simulator to handle the functions not supported downstream and produce the expected results by maintaining its own internal state and invoking multiple CGI functions in the downstream CGI implementation.

4.2.3 Types of targets

4.2.3.1 General graphics devices

The most familiar example of a target is a typical graphics hardware device. In this case, nothing is presumed about the nature of the interface between the Interpreter or Driver and the device. The Interpreter or Driver does whatever it must in order to achieve the desired effects on the target device.

4.2.3.2 Metafiles

The target may be a metafile written in accordance with the CGM standard. In this case, the CGI implementation produces a metafile such that rendering it will produce the same picture as is required by ISO/IEC 9636 for a CGI Virtual Device implementation driving an actual display. (CGMs generated through a CGI interface comprise only a subset of all possible CGMs.)

There is no requirement that a metafile produced by a CGI implementation must be a CGM.

4.2.3.3 Windowing systems

Graphical information generated through the CGI may be displayed in a window of a windowing system. In such cases, the CGI implementation may interface with the windowing system by means of the graphics primitives the windowing system provides. The windowing system may have other clients in addition to the CGI implementation.

4.2.3.4 CGI Virtual Device implementations

As indicated in configurations (d) and (e) of figure 1, the target may itself be a CGI Virtual Device. A CGI Virtual Device can also be considered to be a CGI client. This allows for a variety of useful CGI configurations.

4.3 Example configurations

4.3.1 CGI Virtual Devices as GKS workstations

Figure 2 shows a possible relationship between an implementation of GKS and a collection of CGI Virtual Devices. An application typically communicates with the GKS implementation through one of the standard language bindings. The GKS implementation manages a collection of GKS Workstations, some of which may be implemented in terms of instances of CGI Virtual Devices. Four Workstation to CGI Virtual Device connections are shown. Workstation 1 uses Workstation Dependent Segment Storage, but does not use any segment capabilities in the Virtual Device. Workstation 2, also implementing WDSS, uses the segment capability in its underlying instance of the CGI. The position of the CGI in Workstation 2 is illustrated as "higher" than that of Workstation 1 because the capabilities of the CGI in Workstation 2 more closely approximate the needs of the Workstation interface than those of Workstation 1. Workstation 3 uses two CGI Virtual Devices, one INPUT, the other OUTPUT, to implement an input/output Workstation. Workstation 4 shows how a remote device may be implemented. Both of the interfaces below the Workstation interface are CGIs. The GKS Workstation interface to the CGI Generator is a procedurally bound interface. The CGI generator translates procedural CGI calls into a CGI data stream for interpretation by a CGI Virtual Device.

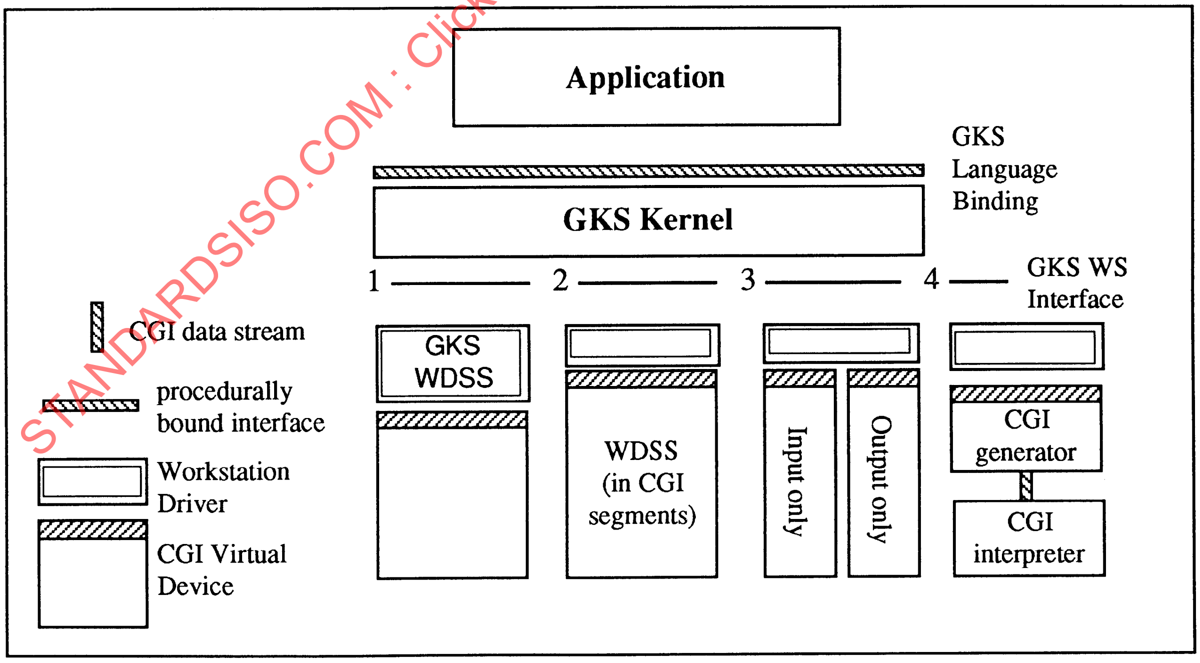

Figure 2 – CGI Virtual Devices as GKS workstations

4.3.2 CGM interpretation using CGI

Figure 3 shows how a CGM interpreter might use the CGI. An application controls the interpreter and selects a CGM for interpretation. The CGM interpreter has direct control of the CGI Virtual Device. It reads the Descriptor elements from the CGM metafile and sets up the Virtual Device based on default attributes required, precisions, and other modes. If the requirements stated in the metafile are sufficiently close to the capabilities of the CGI Virtual Device (as indicated by the CGM ELEMENTS LIST, required precisions, etc.) and the CGI supports the appropriate data stream encoding, the CGM interpreter can pass Picture Body elements directly to the CGI for realization. Note that direct interpretation of CGMs is not covered by the CGI.

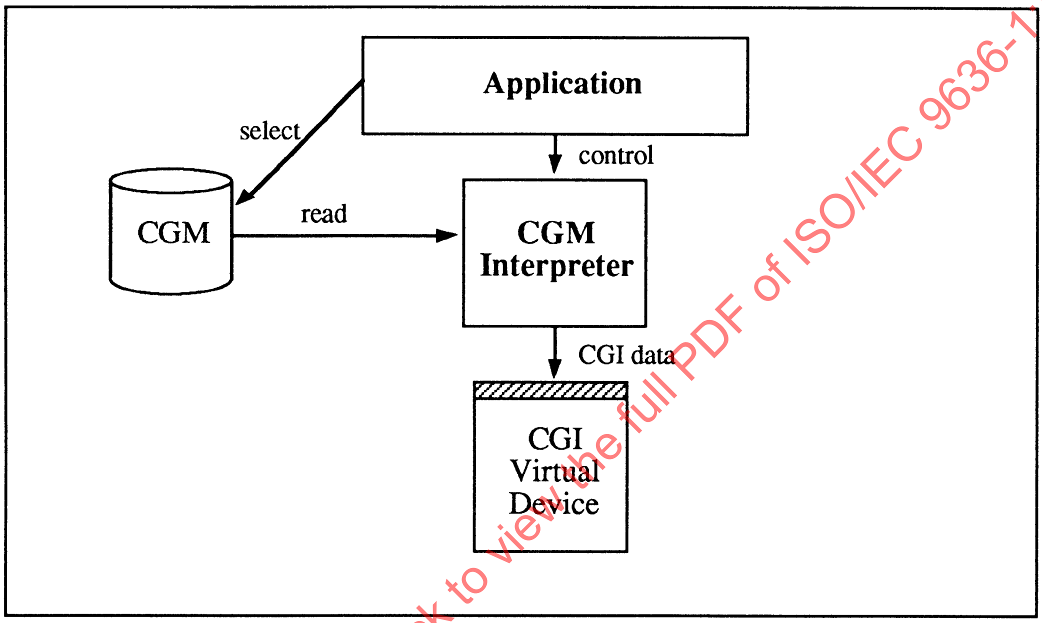

Figure 3 – CGM interpretation using CGI

4.3.3 CGI Virtual Devices as 3D workstations

Figure 4 shows how a GKS-3D or PHIGS implementation might use CGI Virtual Devices. The basic mechanism is as for GKS, as indicated in figure 2. The important distinction shown is that all 3D to 2D transformations must be resolved above the CGI, since the CGI is a 2D-only interface.

4.3.4 Input/output workstations as combinations of CGI Virtual Devices

Figure 5 shows three ways CGI Virtual Devices may be used to build input/output workstations for a hypothetical graphics system. Workstation 1 drives OUTIN CGI (a) directly, using both the input and output capabilities of the CGI Virtual Device. Workstation 2 drives two Virtual Devices, one which supports output (b) and one which supports input (c). To coordinate the input and output of the two CGIs, Workstation 2 must continually read input from CGI (c), potentially process the data read, and pass workstation code generated echoes to CGI (b). Workstation 3 is configured similarly to Workstation 2, however, CGI Virtual Devices (d) and (e) support remote echoing. Remote echoing provides a more efficient path for data read through one CGI to be echoed through another. The workstation code still needs to provide the communication path between the separate CGI Virtual Devices. The majority of the echo processing is provided by the CGI (d) in a way that has less impact on its other output capabilities.

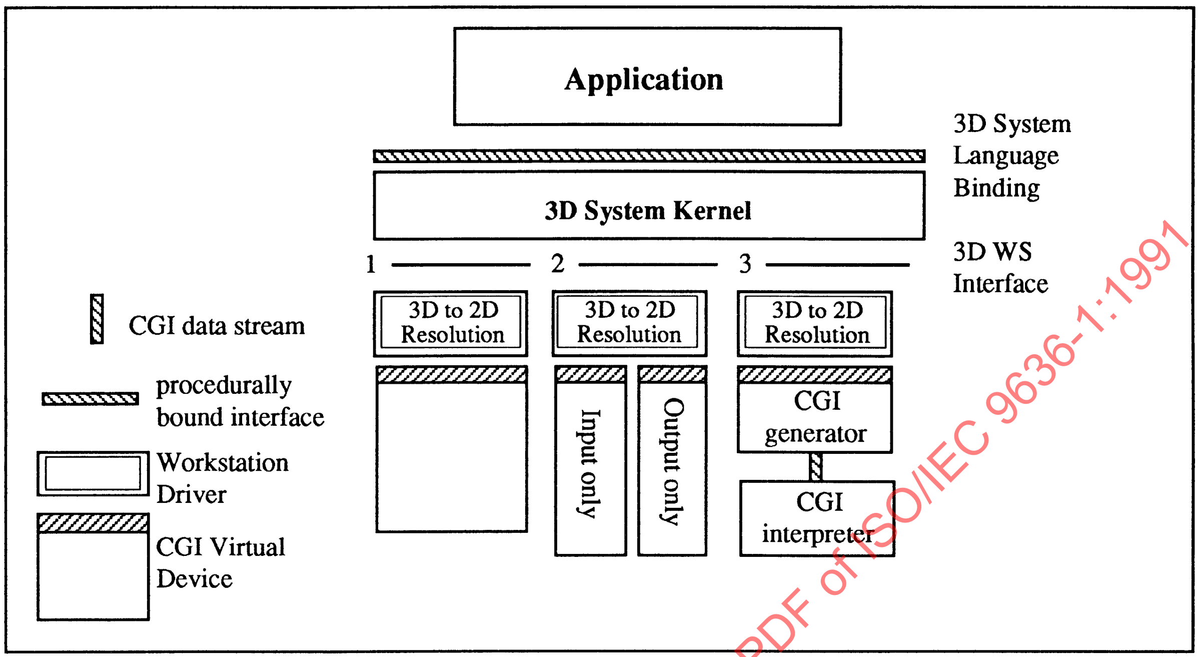

Figure 4 – CGI Virtual Devices as 3D workstations

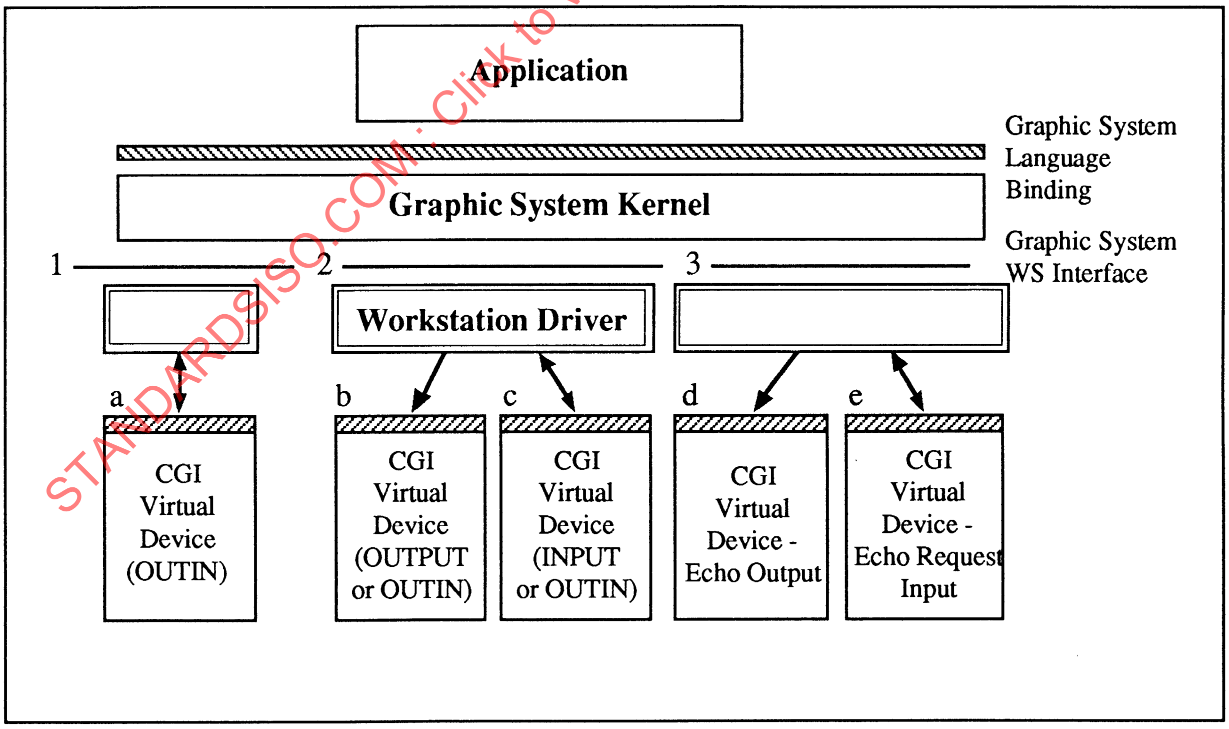
Application
Graphic System Language Binding
Graphic System Kernel
Graphic System WS Interface
1
2
3
Workstation Driver
a
b
c
d
e
CGI Virtual Device (OUTIN)
CGI Virtual Device (OUTPUT or OUTIN)
CGI Virtual Device (INPUT or OUTIN)
CGI Virtual Device - Echo Output
CGI Virtual Device - Echo Request Input

Figure 5 – Implementing input/output workstations as combinations of CGI Virtual Devices

4.4 Relationship to other standards

4.4.1 Introduction

The reference model configuration examples in 4.3 show the relationship between different components of a graphics reference model. They show that the CGI is located between the workstation software of a graphics system and a device. On the system side, the CGI has to interface in a concise and efficient manner with GKS. On the device side, the CGI supports the interpretation of a CGM metafile in a straightforward way.

4.4.2 CGI data encodings standards

ISO/IEC 9636 is closely related to a separate multipart standard in which several specific data stream encodings of CGI functions appear. Encodings are specific representations of the function syntaxes defined in ISO/IEC 9636. They are intended to describe exactly a data stream connecting a graphics system and a device or metafile.

The data stream encodings of CGI are based on the encoding principles of the corresponding data stream encodings of the CGM.

The character encoding conforms to the rules of ISO 9282-1. It is particularly suitable for transfer through networks that cannot support binary transfers, and its compressed coding mechanism minimizes data transfer.

The binary encoding provides an encoding that requires fewer calculations to generate and interpret on many systems.

The clear text encoding provides an encoding that can be created, viewed, and edited with standard text editors. It is also suitable for transfer through networks that support only transfer of text files, or over data communications links that only support printable characters.

4.4.3 CGI language bindings standards

The primary users of procedural bindings of the CGI include graphics system programmers and device driver programmers, but exclude high level end-user application programmers, for whom the ISO application program interface standards (e.g. GKS and PHIGS) are intended.

There are two different methods for mapping CGI functions to programming language procedures: a multi-procedure mapping, and a single entry point mapping. An objective for language binding interfaces on the graphics system level (as in GKS or GKS-3D) may be a one-to-one mapping to programming language procedures. A workstation device interface may be handled by a single entry point interface.

There are multipart standards for each of these types of procedural bindings. Both of the language bindings standards have the same part structure, with a separate part for each programming language.

4.4.4 Relation to GKS (ISO 7942: 1985)

ISO/IEC 9636 draws extensively for its model of a graphics system from GKS.

GKS communicates with devices at the level of the GKS workstation interface. The CGI supports communication at this level. Note that the CGI also provides support for communication to simpler devices.

Constituency Profiles are defined in clause 6 of this part of ISO/IEC 9636 for all levels of GKS. The CGI defaults and the preferred behaviour of functions, where latitude is allowed, are specified to suit GKS.

4.4.5 Relation to GKS-3D (ISO 8805: 1989) and PHIGS (ISO/IEC 9592: 1989)

The CGI is an International Standard for 2-D graphics systems. Direct support of 3-D functionality, as may be needed by GKS-3D and PHIGS, will require additional processing above the level of the CGI. The CGI may be extended, if necessary, to contain 3-D functionality. Should these extensions be made, the parts of ISO/IEC 9636 that describe the extensions will outline the specific relations between the CGI and these standards. If such extensions are made, the general principles governing the CGI's relationship to GKS shall apply.

4.4.6 Relation to CGM (ISO 8632: 1987)

The CGM is a static picture capture metafile. ISO/IEC 9636 draws extensively for its model of a graphics picture from the CGM. There is a very close correspondence between most of the CGM elements and a subset of the functions in the CGI. In particular, all elements which appear in a CGM picture body have direct counterparts amongst the CGI functions. Furthermore, the standard data stream encodings of CGI functions are based on the encoding principles of the corresponding data stream encodings of the CGM. When a CGI function and a CGM element correspond, their representations are identical in a given encoding.

4.4.6.1 Relation to CGM interpretation

A CGM interpreter is a natural client of the CGI. CGM elements within a picture body may be passed directly to a CGI interpreter without modification of their abstract parameterization. The interpretation of a CGM picture body element as a CGI function, will result in CGI Virtual Device behaviour which is consistent with the intended effect of the corresponding metafile element. Note that it may be necessary for the metafile interpreter to change the encoding of such elements before they are sent to the CGI interpreter (e.g. from character to binary encoding). Such changes are limited to a change in parameter representation and would not affect the abstract parameterization.

Constituency Profiles are defined in clause 6 of this part of ISO/IEC 9636 to describe CGI functions and resources suitable for use by CGM interpreters.

4.4.6.2 Relation to CGM generation

A CGI implementation may be used to assist in the generation of a CGM. A CGI generator interface is a procedurally bound CGI Virtual Device implementation which is realized by means of an interpreting CGI implementation. The generator encodes the CGI functions and sends them to the downstream interpreter. For CGM elements which have corresponding CGI functions, the encodings are identical. Such encoded functions may be placed directly in a metafile. That is, a procedurally bound CGI generator may be used to encode portions of the data stream which constitutes a metafile. It is the responsibility of the CGM generator implementation to encode CGM elements which do not correspond to CGI functions.

4.4.7 Relation to Procedures for Registration of Graphical Items (ISO/IEC TR 9973 : 1988)

For certain functions, the CGI defines value ranges as being reserved for registration or future standardization. The values and their meanings will be defined using the procedures established for coordinating registration across graphics standards. It is intended that, where applicable, the CGI's usage of registered values (such as GDP identifier) be identical with their usage in other graphics standards.

Constituency Profiles, other than those defined in ISO/IEC 9636, may also be registered.

5 Concepts

5.1 Introduction

The objective of the Computer Graphics Interface (CGI) is to provide for the description and communication of graphical information between a client program and a Virtual Device in a device-independent manner. To accomplish this, ISO/IEC 9636 defines the form (syntax) and functional behaviour (semantics) of a set of functions that may occur in the CGI.

- Control functions provide for session initialization and termination, specification of modes of operation of certain other functions and the coordinate space to be used, selection of protocols for exchange of data, and control of the device's operation.
- Graphic primitive functions describe the visual components of a picture on the Virtual Device.
- Attribute functions control the setting of state list entries which describe the appearance of graphic primitives.
- Escape functions describe device-dependent or implementation-dependent functions; however, the functions are not otherwise standardized within the context of ISO/IEC 9636 itself (they may, however, be registered).
- External functions communicate information not directly related to the generation of a graphical image.
- Segment functions can be used to store and manipulate groups of graphic objects.
- Input functions provide the ability to obtain input from the Virtual Device in different ways and control the form and timing of that acquisition.
- Raster functions can be used to generate and manipulate images on or within raster devices.
- Inquiry functions provide access to the content of description tables and state lists.
- Error functions can be used to select the level of error reporting and detection as well as to obtain error information.

5.2 Global CGI concepts

5.2.1 CGI Graphic Object Pipeline

This subclause presents the principles of the CGI Graphic Object Pipeline model. Additional details are presented in clause 3 of the parts of ISO/IEC 9636 in which they are relevant. Figures 6 and 7 depict the Graphic Object Pipeline; figure 6 represents a Virtual Device with none of the raster functionality defined in ISO/IEC 9636-6; figure 7 represents the addition of the functionality of ISO/IEC 9636-6 and illustrates the impact that this conceptually has on the pipeline.

The presented pipeline is a conceptual model intended to help explain the abstract behaviour of a CGI Virtual Device. It is not intended as a typical method for implementing a CGI Virtual Device.

There are four basic types of entities indicated in these figures:

- Pipes: paths along which graphic objects flow. The Segment Storage and Drawing Surface are considered to lie on a pipe. Alternative paths appear at several points in the pipe network.
- Operations: processes that may be applied to graphic objects as they flow along the pipes.
- State list information: information relevant to the execution of an operation.
- Graphic object annotations: text alongside of pipes to indicate how an object is conceptually represented at that point in the pipeline.

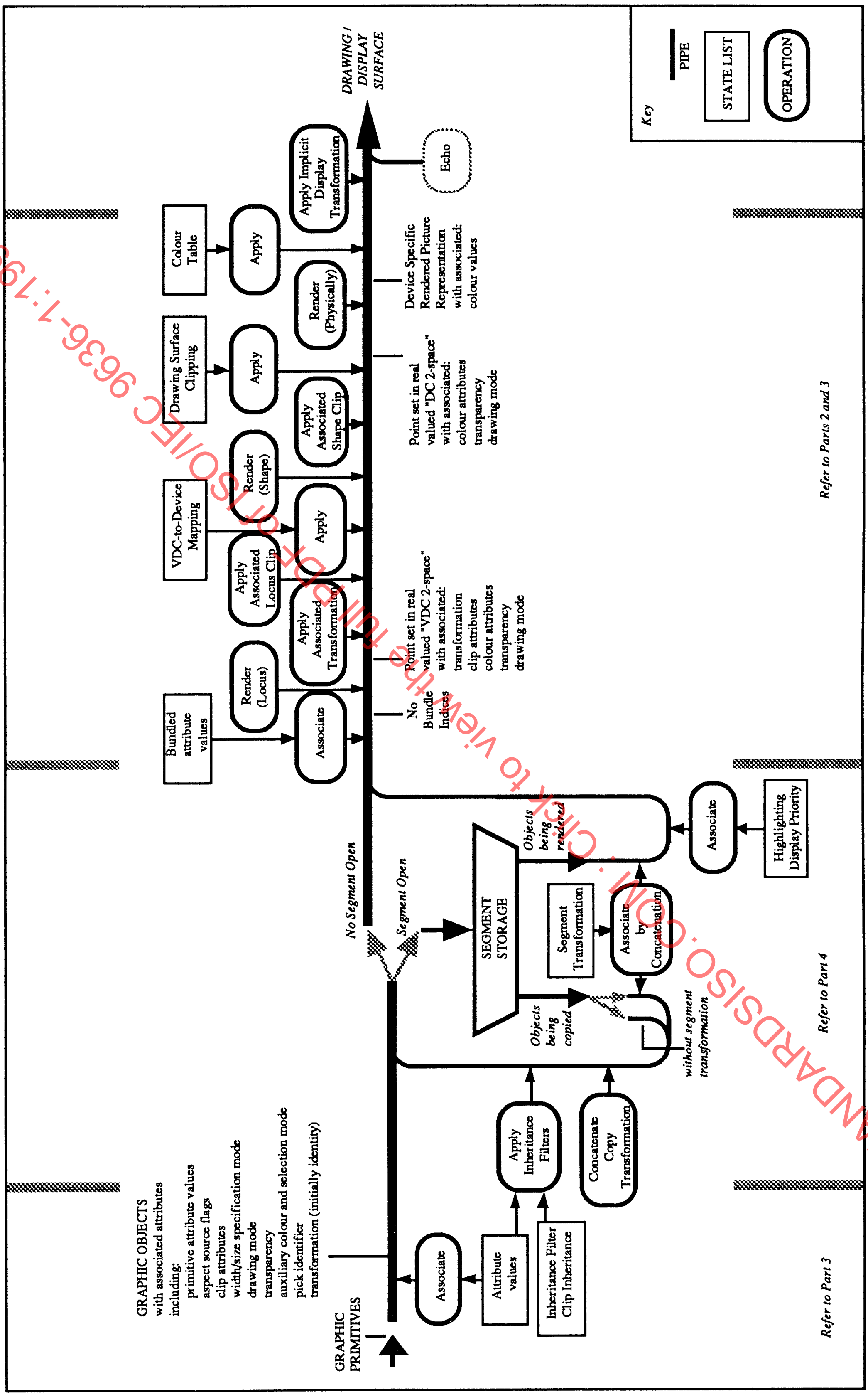
GRAPHIC PRIMITIVES
GRAPHIC OBJECTS with associated attributes including: primitive attribute values aspect source flags clip attributes width/size specification mode drawing mode transparency auxiliary colour and selection mode pick identifier transformation (initially identity)
Associate
Attribute values
Inheritance Filter Clip Inheritance
Apply Inheritance Filters
Concatenate Copy Transformation
No Segment Open
Segment Open
SEGMENT STORAGE
Objects being copied
Objects being rendered
Segment Transformation
Associate by Concatenation
without segment transformation
Associate
Highlighting Display Priority
Bundled attribute values
Associate
Render (Locus)
Apply Associated Transformation
Apply Associated Locus Clip
VDC-to-Device Mapping
Apply
Render (Shape)
Apply Associated Shape Clip
Drawing Surface Clipping
Apply
Render (Physically)
Colour Table
Apply
Apply Implicit Display Transformation
Echo
DRAWING / DISPLAY SURFACE
No Bundle Indices
Point set in real valued "VDC 2-space" with associated: transformation clip attributes colour attributes transparency drawing mode
Point set in real valued "DC 2-space" with associated: colour attributes transparency drawing mode
Device Specific Rendered Picture Representation with associated: colour values
Key
PIPE
STATE LIST
OPERATION
Refer to Part 3
Refer to Part 4
Refer to Parts 2 and 3

Figure 6 – CGI Graphic Object Pipeline

5.2.1.1 Graphic objects

Conceptually, the interpretation of a graphic primitive function generates a graphic primitive which enters into the CGI pipeline. A simple graphic object is created from a graphic primitive when its primitive attributes and general attributes have been associated. A compound graphic object is created from a group of graphic primitives (see ISO/IEC 9636-3, 3.5 for an elaboration of the model for creating compound graphic objects). Graphic objects flow, in their entirety, along the pipeline and have operations applied to them. The application of an operation may change the geometric and attribute information contained in the graphic object (e.g. geometric information is transformed by the VDC-to-Device Mapping). Graphic objects may be stored in segments. Subsequently, they may be re-entered into the pipeline for further processing, either through an implicit action, or through explicit instruction by the client.

5.2.1.2 State list information

State list information maintained by the Virtual Device is conceptually associated with the various operations along the pipeline. Control over the contents of these state lists is provided by CGI functions which directly modify the state list entries. These functions do not generate graphic objects and do not pass along the Graphic Object Pipeline. (Note that the partitioning of state list information depicted in figures 6 and 7 does not necessarily reflect the actual structure of the state list within the CGI.) In figure 7, bitmap state list information is held on a per bitmap basis.

5.2.1.3 Association and application of attribute values

The Graphic Object Pipeline model distinguishes between the association of attribute values with a graphic object and the application of these attribute values. For example, the association of transformations with a graphic object occurs at different points along the pipeline from when the actual application of the transformations occurs.

5.2.1.4 Alternative paths

Figure 6 illustrates several alternative paths in the pipeline along which graphic objects may flow. If a segment is open, graphic objects are directed into this segment in segment storage. If a segment is not open, graphic objects are directed along the rendering pipeline for further processing. Graphic objects may be copied from segment storage in order to render them and in order to copy them. Graphic objects stored in segments may be rendered during segment creation and on segment regeneration. Regeneration can be explicit or implicit (e.g. due to a change in the CGI state list information when Implicit Segment Regeneration is ALLOWED). There are two pipes flowing from segment storage. One is used for graphic objects being explicitly copied, the other for graphic objects being rendered. For graphic objects copied from segment storage, the segment transformation associated with the stored segment may optionally be used during the copying process. The segment attributes of display priority and highlighting are not associated with an object until it is taken from a segment for rendering. After a copy segment operation the associated clip rectangle attribute may have been transformed to become a convex polygon.

5.2.1.5 Transformations

An identity transformation is conceptually associated with every primitive after it enters the pipeline. This transformation is used to accumulate any subsequent segment and copy transformations to be applied to a graphic object during rendering. The segment transformation of a segment identified for display is concatenated with the current transformation associated with each graphic object as the objects are copied from the segment for further rendering. When a segment copy operation is invoked, the client has control over whether the segment transformation of the copied segment is to be accumulated with the transformation associated with each copied object. A copy transformation (provided as a parameter) is also concatenated with the transformation of each copied object. Thus, the segment transformation of a segment affects the objects it contains during a copy or render operation and does not affect objects as they are stored. Application of these transformations to the associated clip rectangle attribute following a segment copy operation may result in this attribute becoming a convex polygon.

5.2.1.6 Filtering of copied attributes

Attribute values already associated with graphic objects may be changed when a graphic object is copied. Selected attribute values associated with the graphic object may be replaced by current state list attribute values.

5.2.1.7 Bundled attribute values

Some attribute values may be collected into entities termed “bundles”, which are organized into “bundle tables”. These attribute values may be associated with a graphic object indirectly through an index reference to a bundle table entry. Whether a particular attribute value is to be taken directly from a state list or indirectly from bundled attribute values is determined by an aspect source flag (ASF) corresponding to each bundled attribute of a primitive.

For graphic objects rendered from segments, the actual values of bundled attributes are not associated with the graphic objects until they leave segment storage. After this association step, a graphic object should no longer be perceived as having associated ASF and bundle index attributes. The actual values for all attributes will have been explicitly associated.

5.2.1.8 Abstract rendering

The operation *Render Abstractly* in figure 6 marks a point where the conceptual representation of a graphic object changes significantly. Up to this point, a graphic object has been represented by data similar to its representation at the CGI interface. After this abstract rendering step, however, the graphic object should be regarded as a subset of an *abstract VDC space*. This abstract VDC space is an infinite, continuous, real-valued, two-dimensional space, as opposed to what is representable within the VDC range.

Conceptually, a locus is a mathematical object like a point, line segment or polygon; loci are zero-dimensional, one-dimensional, or two-dimensional subsets of the abstract VDC space. For marker and text objects, the loci are points. For line objects, they are individual line segments or portions of arcs. Fill objects may have a locus for the edges and a locus for the interior. The interior locus consists of the set of points that lie "within" the boundary of the object as determined by the parity fill algorithm.

5.2.1.9 Application of associated transformation

The associated transformation, arising from the possible concatenation of segment transformations and copy transformations, is always a mapping from VDC space to VDC space. Consequently, its application to the (infinite) point set representing a graphic object produces the same type of representation, (assuming that the applied transformation is not singular). This type of transformation is very general, that is, it may introduce translation, anisotropic scaling, rotation, and skew. The only requirement is that each coordinate of the image of some given point must be a linear function of the pair of coordinate values for the given point.

Note that such transformations may change the shape of some graphic objects. If there is skew, a graphic object initially specified as a rectangle may become a parallelogram. If there is anisotropic scaling, a graphic object initially specified as a circle may become an ellipse. The shape and orientation of markers is exempt from such transformations.

5.2.1.10 Application of associated locus clip

There are object clipping modes for lines, edges, and markers. When the clip indicator associated with a graphic object is ON, only what lies inside the associated clip region and the VDC extent is rendered. The clipping process in abstract VDC space may be described in simple mathematical terms as the intersection of the graphic object point set with the set of points covered by the associated clip region. The object clipping modes allow the client to be more precise concerning the level at which clipping decisions are made and is represented in figure 6 by the two operations *Apply Associated Locus Clip* and *Apply Associated Shape Clip*. An object clipping mode may be one of LOCUS, SHAPE, or LOCUS THEN SHAPE.

If an object clipping mode is LOCUS or LOCUS THEN SHAPE, locus clipping is applied immediately after the application of the associated transformation. Locus clipping is performed on each portion of a graphic object based on its mathematical location, independent of the area it will occupy after rendering.

5.2.1.11 Application of VDC-to-Device Mapping

The VDC-to-Device Mapping is a transformation from VDC space to an abstract DC space. This abstract DC space is also an infinite, continuous, real-valued, two-dimensional space. The VDC-to-Device Mapping is determined by the current values of VDC Extent, Device Viewport, and Device Viewport Mapping. This type of transformation is restricted to allow only translation, scaling, and reflection. No general rotation or skew is possible.

The result of the application of the VDC-to-Device mapping is a two-dimensional point set in the abstract DC space.

5.2.1.12 Shape rendering

Application of the shape rendering operation produces a representation of a graphic object such that all primitive attributes that affect the object's shape have been realized (e.g. Line Width, or Marker Type). After shape rendering, a graphic object is considered to cover an area in a two-dimensional space and, conceptually, is represented as an infinite two-dimensional set of points (where "infinite" refers to the cardinality of the set).

Although attributes such as line type and character orientation have already been realized in the point set representation of a graphic object, other properties are still associated. These may include colour (often as an index), transparency, drawing mode, auxiliary colour, clip rectangle, clip indicator, object clipping mode, and transformation.

5.2.1.13 Application of associated shape clip

If an object clipping mode is SHAPE or LOCUS THEN SHAPE, shape clipping is performed. The application of the clip is performed in an abstract DC space and provides the finest level of clipping.

After application of the associated shape clip, the following attributes remain associated with a graphic object: pick identifier, colour, auxiliary colour, transparency, and drawing mode.

5.2.1.14 Size specification modes

There are specification modes for line widths, edge widths, and marker sizes. These modes may be either VDC or SCALED. For SCALED sizes, the size remains fixed with respect to the drawing surface, independent of the VDC-to-Device Mapping or any associated transformation. The pipeline shown in figure 6 corresponds to this size specification mode. In contrast, VDC sizes can change proportionally with the size of the device viewport (assuming the VDC extent remains unchanged). These sizes are subject to transformation by the copy and/or segment transformation associated with the graphic object. In the case of VDC size specification, shape rendering is performed prior to the application of the associated transformation. (See also ISO/IEC 9636-3, figure 4.)

5.2.1.15 Application of drawing surface clipping

When enabled, drawing surface clipping conceptually occurs in the abstract DC space, similarly to SHAPE clipping with the associated clip rectangle in VDC space. Note that the Drawing Surface Clip Rectangle comes from a state list and is not associated with any graphic objects. (See also 5.2.1.21.)

5.2.1.16 Physical rendering

The operation *Render (Physically)* corresponds to the actual physical rendering step. This is where the abstractions take on physical representation by whatever means are provided in the implementing hardware. The result is referred to as a *device-specific representation.* Examples include pixels in a frame buffer and pen up/down and movement commands.

The physical rendering step is where most of the approximations involved in representing the picture occur. The former abstract VDC and DC spaces could still convey precise information about areas in continuous two-dimensional space. In the device-specific representation, coordinates must be rounded off to integers (or whatever the device is capable of implementing). Therefore, line width, edge width, etc., may no longer be realized exactly.

5.2.1.17 Application of colour

This operation is applicable for both direct and indexed colour values. In the case of indexed colour, the colour indices associated with the graphic object or bitmap contents are translated into direct colour values determined from the colour table. In either case, physical translation to colour realizations on the drawing surface occurs in this operation by means outside the scope of ISO/IEC 9636.

5.2.1.18 Application of implicit display transformation

The step of going from the device-specific representation of the actual visible picture on a display surface involves a further transformation. This transformation is outside the control of any software yet it may greatly affect the appearance of the picture. Note that it is possible for the implicit display transformation to include anisotropic scaling in x and y. For example, some displays have pixels that are not square when they get to the display surface. Even though the CGI client has no direct control over this implicit display transformation, its role in the model has been explicitly illustrated. However, the client can indirectly control this transformation if the Virtual Device implements the functional capability defined in ISO/IEC 9636-6. By utilizing the pixel size information in the Raster Description Table the client can determine the anisotropic scaling, and hence offset this transformation by adjusting its own transformations accordingly.

5.2.1.19 Isotropy

If isotropy is not forced, then the client has complete, explicit control over the VDC-to- Device Mapping. If isotropy is forced, the implementation must take into account knowledge about relative x- and y-scalings in the implicit display transformation.

In particular, the explicitly specified device viewport is modified (by shrinking it in either x or y) so that the net effect of the VDC-to-Device Mapping composed with the implicit display transformation is isotropic, i.e. equal x- and y-displacements in VDC space will map onto what are perceived as equal x- and y-displacements on the display surface.

5.2.1.20 Bitmaps and raster operations

For a Virtual Device which implements raster functionality as defined in ISO/IEC 9636-6, the results of the operation *Render (Physically)* are directed into the currently selected drawing bitmap (see figure 7). If this bitmap is a displayable bitmap (or if the contents of the drawing bitmap are subsequently moved to a displayable bitmap), then the bitmap contents may be visible on the display surface. All bitmaps, whether displayable or not have distinct state list information. Each bitmap state list includes information to obtain the VDC-to-Device Mapping and the Drawing Surface Clip Rectangle.

5.2.1.21 Application of drawing surface clipping to bitblts and PIXEL ARRAY

The application of drawing surface clipping applies to both bitblts and PIXEL ARRAY. If the Drawing Surface Clip Indicator from the current drawing bitmap state list is ON, the drawing surface clipping operation is applied. If it is OFF, no clipping is required and no additional overhead is incurred.

5.2.1.22 Echo

The operation *echo* using input functions as defined in ISO/IEC 9636-5 indicates that the effect of echo is conceptually separate from drawing (and has no effect on the bitmaps of a Virtual Device with raster functionality). The type of echo used is determined by a state list entry for Echo Type given in ISO/IEC 9636-5.

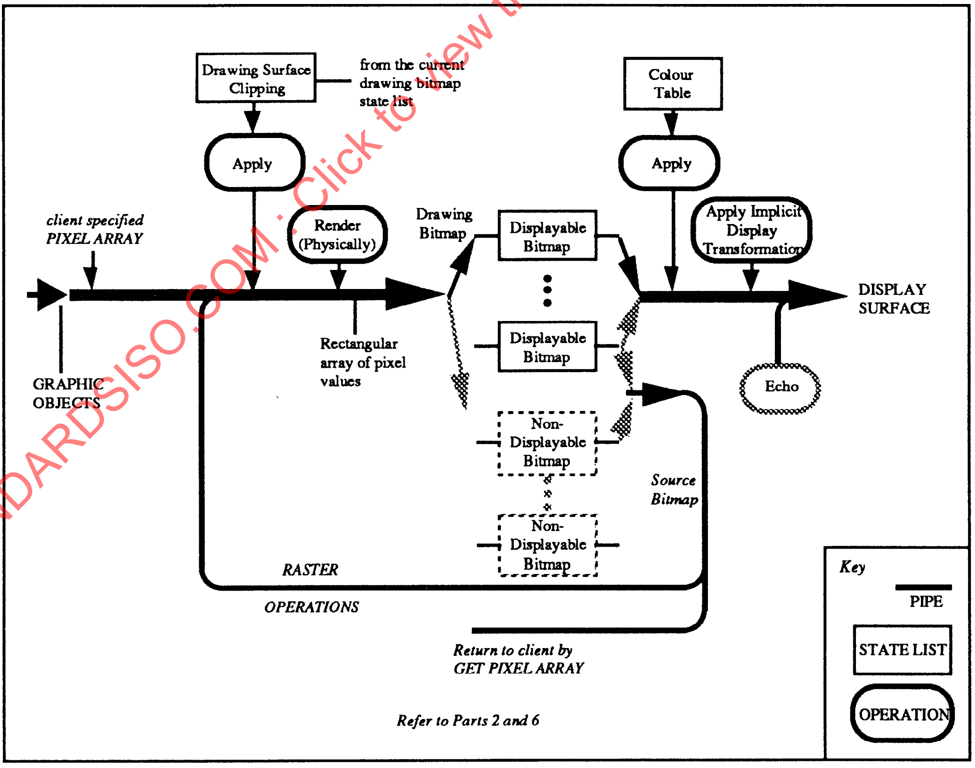

Figure 7 – Additional raster pipeline components

5.2.1.23 Implementation

It should be emphasized that figures 6 and 7 do not represent a method of implementing the CGI. The representations used for the abstract VDC and DC spaces cannot be realized in any current computer system. In practice, many computer graphics systems collapse everything from abstract rendering (after association of bundled attributes values) through to physical rendering into a single rendering process. This step may include both forms of clipping as well as the realization of geometric and non-geometric attributes.

5.2.2 CGI state model

The allowed sequence of operations in a system can be described in terms of operating states and state transitions. The operating states of a system are defined in terms of state variables. The overall operating state of a system can be defined as the set of values of the state variables at any instant provided that transitions between operating states occur as a result of atomic actions of the system.

For the CGI these actions may occur:

- as a result of complete execution of functions by a CGI Virtual Device;
- as a result of operator interaction.

The operating state of a CGI Virtual Device may prohibit the use of a certain CGI function. Operating states are defined in ISO/IEC 9636-2, ISO/IEC 9636-3, ISO/IEC 9636-4, ISO/IEC 9636-5, and ISO/IEC 9636-6 and are presented in state lists. The state variables, their values, and the actions which cause a transition from one state to another are described in these parts. In addition, there is conceptually a global operating state for a CGI Virtual Device which defines whether the Virtual Device is "initialized". With the exception of the INITIALIZE function, all other functions of the CGI are ignored by the Virtual Device when it has not been initialized. (Note that this state restriction and error reaction are not documented in any of the CGI function specifications.)

State restrictions which have to be detected are minimized in the CGI. If the client of the CGI disobeys the rules imposed by state restrictions, the CGI implementation guarantees an error and reacts in a defined fashion.

Each part of ISO/IEC 9636, where applicable, contains one or more tables which define where the use of a CGI function is restricted by certain operating states.

5.2.3 Description tables

The functional specification parts of ISO/IEC 9636 define description tables. Each description table lists features of the implementation that characterize the CGI Virtual Device.

The information contained in description tables is available to clients of the CGI in order that they may determine the implementation-dependent and device-dependent behaviour of the CGI implementation. The information in the description tables is not changeable through any action of the client through the CGI. This information is considered permanent. Description table entries may change, however, by means outside the CGI, such as by device reconfiguration by the operating system or, in some environments, by explicit operator or resource manager action (e.g. windowing systems). That such spontaneous changes are possible is indicated by an entry in the Output Device Description Table.

5.2.4 State lists

The function specification parts of ISO/IEC 9636 define state lists. Each state list details some aspect of the state of the CGI Virtual Device.

The information contained in state lists and thus available for client inquiry are "realized" values; "set" values differ from "realized" values only when:

- ISO/IEC 9636 allows an implementation latitude in the range of values of a datum that the implementation is able to represent, and;
- an implementation that only employs a restricted range of values could lose information through rounding or quantization errors.

NOTE – An example of such information is line width. An implementation may only permit line widths that are integer multiples of its minimum line width. If the line width is specified in VDC units, information may be lost by scaling a quantized value after a change of the VDC-to-Device Mapping. An implementation will need to maintain more internal state information, such as "set" values, than is specified in state lists in order to return the "realized" value.

A change in VDC Type shall as a side effect reset certain state list entries to their default, as described in ISO/IEC 9636-3, 4.2.2.

The information contained in state lists is available to clients of the CGI in order that they may determine the state of the CGI implementation.

State list entries are initialized at the time of INITIALIZE to the values shown in the default column of the state lists in clause 7 of ISO/IEC 9636-2, ISO/IEC 9636-3, ISO/IEC 9636-4, ISO/IEC 9636-5, and ISO/IEC 9636-6. Although the client may subsequently reset these state list entry values, it is possible for the client to inquire the original state list defaults at any time after initialization by means of the function STATE LIST INQUIRY SOURCE. This function, defined in ISO/IEC 9636-2, allows the client to control whether state list inquiry functions return current or default values.

Inquiry by the client of state list defaults may be useful if the client wishes to restore some state list entry to its initial state without having to preserve a copy of this information within its own memory (or re-INITIALIZE). The client may control whether state list inquiry functions shall return current or default values by means of the function STATE LIST INQUIRY SOURCE. This function is described in ISO/IEC 9636-2, 5.5.10.

5.2.5 Static and dynamic state lists

State lists are further distinguished as Static or Dynamic.

A static state list records the state of the implementation as directly defined by ISO/IEC 9636. Only one copy of each static state list may exist in an implementation of the CGI.

A dynamic state list records the state of an entity that was created by the implementation in response to a function invoked by the client.

Lists of indices, or other means of accessing dynamic state lists, are always available in a static state list. Examples of entities that may be created dynamically are bundle table entries (defined in ISO/IEC 9636-3), segments (defined in ISO/IEC 9636-4), echo output entities (defined in ISO/IEC 9636-5), and bitmaps (defined in ISO/IEC 9636-6).

5.2.6 The structure of description tables and state lists

ISO/IEC 9636 defines the information that is to be available from description tables and state lists; it contains no provision that defines the way in which an implementation may store the information. For example, the description table information concerning which functions defined by ISO/IEC 9636 are implemented may be kept (in an interpreter implementation of the CGI) only as a function decode dispatch table.

The definitions of description tables and state lists define the grouping and data types of information that they contain. These groupings and data types are relevant to the corresponding inquiry functions which return this information.

5.2.7 The principles of inquiry

The content of description tables and state lists is available to the client of the CGI by means of inquiry functions. An abstract function name starts with "INQUIRY" if and only if that function only returns the values from a description table or state list. An inquiry function name starts with "LOOKUP" when it answers a question about the existence and possibly the value of an entry in a description table. The specifications of the inquiry functions are provided in clause 6 of ISO/IEC 9636-2, ISO/IEC 9636-3, ISO/IEC 9636-4, ISO/IEC 9636-5, and ISO/IEC 9636-6.

The presence, in an implementation, of information in any description table or state list defined by ISO/IEC 9636, may be deduced by determining that the corresponding inquiry function, which would return this value, is supported. The function LOOKUP FUNCTION SUPPORT, defined in ISO/IEC 9636-2, is provided to allow for inquiry about the presence of support for any CGI function (inquiry or otherwise) defined in any implemented parts of ISO/IEC 9636.

5.2.7.1 Response validity

All soliciting functions have as their first return parameter a response validity value. When the response validity is INVALID, the values of any other return parameters should be regarded as undefined. No attempt should be made by the client to attach any meaning to other returned parameters, if any, unless the response validity is VALID. A value of VALID for response validity does *not* guarantee that *all* returned parameters are valid. Where there is a possibility that only a subset of the returned parameters are valid, additional validity parameters will also be returned.

5.2.7.2 List inquiries

The returning of lists to the client requires special treatment. There are two different cases which may arise: lists of indeterminate length and arrays with a known "small" upper bound length containing a variable number of valid elements.

Lists of indeterminate length are cases in which the client cannot be expected to have any prior knowledge of the expected list length. The length of these lists may be quite large. Each of these lists will always be the only description table or state list entry returned by the corresponding inquiry function; no other description table or state list entry information will be returned with it. Since the client may not be able to provide enough memory to receive an entire list in a single inquiry, the client may specify the upper bound for the size of a returned list. Multiple inquiry invocations may be required to retrieve an entire list.

If there is a known "small" upper bound on the possible size of a returned list, the list has been named an "array" and its inquiry has been included with other entries in the same description table or state list. In these special cases, the size of a returned array is fixed. There is an additional parameter to indicate the number of valid elements in an array, with the convention that valid elements are returned at the beginning of each array. The standardized size of a returned array is indicated along with the data type in the abstract specification of these inquiry functions.

Note that a CGI implementation may return an amount of list elements which is smaller than both the requested number and the actual remaining number of list elements in a description table or state list. Regardless of the amount requested, the returned list is always accompanied by the actual number of list elements in the particular description table or state list being inquired. This total is returned in the "total number of list elements in state list" or "total number of list elements in description table" output parameter. This total may be obtained without returning any list elements by requesting zero "number of list elements".

5.2.7.3 Character strings

Character strings are sequences of characters. The client can limit the actual size of a returned string by specifying an upper bound on the returned string length. A returned string shall always be accompanied by its size, which may be smaller than requested.

5.2.7.4 Data records

A data record is actually a list of octets. Therefore, like other returned lists, a data record can be returned in parts. The size of a data record is measured in uniform units of 8-bit octets. The client may limit the actual size of the returned data record by specifying the upper bound on the returned data record size (in octets). The returned data record will always be accompanied by its size, which may be smaller than requested. The total size of the data record in the state list or description table is also returned.

5.2.8 Error philosophy

Different classes of errors are defined below. The detection of all error classes, other than class 9, is required by ISO/IEC 9636. The client may switch on and off error reporting and detection separately for each error class.

5.2.8.1 Error classification scheme

Class 1 errors: These errors are caused by parameter range violations. They are defined by the range constraints given in the functional specification to the left of the relevant parameter's data type. Class 1 errors occur only if the range constraint is constant for a certain implementation, i.e. defined by an explicit numeric range or by a description table entry. Class 1 errors are not listed with the function specification. The reaction to a class 1 error is always to ignore the function. The error number for a range violation for a particular parameter of a particular function should reflect the parameter number as listed in the "Parameter" section of the abstract specification.

As an example of a class 1 error, the function SEGMENT DISPLAY PRIORITY might be invoked with a given priority which is either smaller than 0 or greater than the (constant) Number of Supported Display Priorities, the error 1:002 is reported (i.e. the second parameter of this function). Where an index has a sparse set of valid values which is static and provided in a description table (e.g. LID indices) then it is a class 1 error to request a non-existent index. (This contrasts with an attempt to access dynamically-allocated entities (e.g. bundles, echo entities), which would cause a class 3 error.)

Class 2 errors: These errors occur if an "Out" parameter of a soliciting function is subject to a current precision which is insufficient to accommodate the returned data. The error reaction is that the response validity flag is set to INVALID. The error numbers reflect the "Out" parameter number as listed in the Parameter section of the abstract specification of the function. For example, error 2:002 is reported if the second Out parameter cannot be returned.

Class 3 errors: These errors arise from a request for an unsupported feature, or an operation which cannot be executed, if the error condition is not already covered by a class 1 error. Class 3 errors are always listed in the "Errors" section under the abstract specification of the function and a specific error reaction is defined. This error reaction may be to ignore the function or to use the default value. An example would be if the client attempts to set a line type which is not supported, or to delete a segment which does not exist.

If the erroneous request is for a state list entry with continuous range, then the nearest available value is used and no class 3 error is defined. The realized value is set in the state list and will be returned by the related inquiry function. An example, in this case, is a request for a line width that is greater than the maximum scaled line width.

Class 4 errors: If the client attempts to use a function which the implementation does not support, it is a class 4 error. Only one error identifier is associated with error class 4. This class of error is not listed with the abstract function specification. The reaction is always to ignore the function.

Class 5 errors: If the client attempts to use a function when the CGI implementation is not in the correct state for using that function, it is a class 5 error. Class 5 error identifiers are specified by ISO/IEC 9636. This class of error, along with its error reaction, are listed with the abstract specification of the function. Unless otherwise specified, the error reaction will be to ignore the function. A class 5 error occurs if, for example, the client invokes the TEXT function in figure open state.

Class 6 errors: When the CGI implementation encounters an internal situation which prevents the correct execution of a function (e.g. out of memory) it is a class 6 error. These are situations in which the function has been invoked by the client and it is not reasonable to expect that the client could have anticipated the difficulty the implementation encountered. They may be listed with the abstract specification of the function. If not, they are listed at the end of each part of ISO/IEC 9636. ISO/IEC 9636 does not generally specify the error reaction.

Class 7 errors: When the CGI encounters a situation which cannot be corrected without operator intervention (e.g. a paper jam) it is a class 7 error. Class 7 errors are implementation-dependent and are not listed in ISO/IEC 9636; their error identifiers are always negative.

Class 8 errors: These errors provide a mechanism for the client to detect when it uses a non-standardized function, which is only available in some implementations. The error reaction will be to continue normal execution of the function. The error identifier is always implementation-dependent (and thus always negative).

Class 9 errors: This is an implementation-dependent error class. ISO/IEC 9636 does not define error identifiers or error reactions. Class 9 error identifiers are always negative.

ESCAPE and GDP may use standardized error identifiers when there arise errors to which the meaning of the standardized identifiers are applicable.

5.2.8.2 Action following an error

ISO/IEC 9636 specifies three actions that make up the response of an implementation to an error in a function or to an abnormal condition in the implementation or its environment: error detection; error reporting; and error reaction. These actions are conceptually performed in order.

Error detection is the action that discovers that an error has occurred. Error detection may be suppressed by the client for particular classes of error. If it is so suppressed, ISO/IEC 9636 specifies only that the error should not be reported, though the implementation may also suppress detection and reaction.

Error reporting is the action that makes a record of the error in the error queue. It is further detailed in 5.2.8.5. Error reporting may be suppressed by the client, for particular classes of error.

Error reaction is the action taken by the implementation to deal with the error detected.

5.2.8.3 The error queue

A CGI Virtual Device maintains reports of errors in a first-in-first-out data structure termed the "error queue". The entries of the "error queue", the error reports, consist of an error identifier (error class and error number) and a function identifier (see 5.2.10). The status of the queue is recorded in the Control State List and may be inquired. A function is provided to retrieve error reports from the queue. When the CGI Virtual Device is initialized, the error queue is empty.

The CGI does not automatically report errors to the client program. All transfer of error information is under client control and takes place synchronously.

5.2.8.4 Error checking sequence

The error checking should conceptually be performed in the following sequence for every function:

1) Class 4
2) Class 8
3) Class 5
4) Class 1
5) Class 3
6) Class 2

Error processing is done in the described order for all parameters. If an error is detected with an error reaction which is not to ignore the function, the error is put into the error queue and the error checking process is continued until an error with reaction to ignore the function is detected or all parameters are checked. Errors of classes 6, 7, and 9 may only occur when a function is executed so there is no rule for the sequence in which they should be checked.

5.2.8.5 Error report queuing

When an error has been detected and error reporting for the class of error is ON, the following sequence of operations is performed:

a) If the "error queue" is full, the most recent error report is removed from the queue. The entered report is an ERROR REPORTS LOST report. The error identifier indicates, through a class 6 error, an ERROR REPORTS LOST error. Instead of a function identifier, the report is a counter for errors lost (see 5.2.10).

   1) If the removed error report is an ERROR REPORTS LOST error, the count of errors lost is increased by one and the error report is re-entered in the queue.

   2) If the error report that has been removed is not an ERROR REPORTS LOST error, a new ERROR REPORTS LOST report is created, its count of error reports lost is set to two and the new report is entered in the queue.

b) If the "error queue" is not full, a new error report is created and entered in the queue.

5.2.9 Method of describing functions

The format used throughout clauses 5 and 6 of ISO/IEC 9636-2, ISO/IEC 9636-3, ISO/IEC 9636-4, ISO/IEC 9636-5, and ISO/IEC 9636-6 is designed to separate functional behaviour from implementation. Each function description consists of:

- Function name;
- Input and output parameters, along with their data types (note, *In* indicates that the parameter values are provided by the client, *Out* indicates that the parameter values are returned by the CGI implementation);
- Effect of the function;
- Implicit relationships and notes clarifying how the function fits into the system;
- Errors which can occur, along with the error identifier and the error reaction (if any is defined).

5.2.10 Data types employed

The abstract specifications of functions detail the functions in terms of input and output parameters. The data type of each parameter is selected from a standard set and is identified in the functional specification by a standard abbreviation. Both the data type and the abbreviation are taken from the following list.

Data Type	Representation	Semantics
ASN	CSN	Attribute set name for a set of attribute values being SAVED or RESTORED.
BN	CSN	Bitmap identifier for built-in or client-defined bitmap.
CD		Basic data type, direct colour value. This is a red-green-blue integer triple subject to COLOUR PRECISION within a data stream encoding as controlled by COLOUR PRECISION REQUIREMENT.
CI		Basic data type, colour index. This unsigned index, subject to COLOUR INDEX PRECISION within a data stream encoding as controlled by COLOUR INDEX PRECISION REQUIREMENT, is an index into the colour table.
CO	CI or CD	Colour specifier in the relevant COLOUR SELECTION MODE and subject to the corresponding precision, or, for some functions, subject to local colour precision.
CS	E,S	Character set. A composite type consisting of the character set type(E) and the designator sequence tail string(S). The symbolic names for character set type are (94 CHARACTER SET, 94 CHARACTER SET, The tail string is composed of characters extracted from registered designating escape sequences. (See ISO/IEC 9636-3, annex F). This string is not subject to code extension techniques.
CSN		Basic data type, client specified name, realized as an integer, subject to CLIENT SPECIFIED NAME PRECISION within a data stream encoding as controlled by CLIENT SPECIFIED NAME PRECISION REQUIREMENT.
D		Data record. Data records for the CGI employ basic data types defined here, plus VC and VDC, and have internal structure. A data record is an ordered set of sub-sequences of parameters of a given type. Each such sub-sequence consists of an identifier for the data type, a count of the number of parameters in the sub-sequence, and the list of parameters itself. Any data types may be used for any sub-sequence and different sub-sequences may use the same data type. In data stream encodings, type (for VDC) and precisions apply in the same manner as for normal parameters of a given data type. For VC, Viewport Specification Mode applies.
DC	IF	Device coordinate in native device units in the physical device coordinate system.
DP	DC,DC	A pair of DC values which represent in native device units the x and y coordinates of a point in the physical device coordinate system.
E		Basic data type, enumerated. Can take any value from a finite prescribed set for the particular instance as determined by the context. The set is specified by listing the identifiers which denote the members.
EI	IN,IN	Error identifier. A composite type consisting of the error class (IN) and the error number (IN) within the class as specified by ISO/IEC 9636 (if positive) or by private use (if negative).
ER	EI,IF or EI,FN	Error report. A composite type consisting of an error identifier (EI) followed by either a count of errors lost (IF) for ERROR REPORTS LOST error, or a function identifier (FN) for all other errors.

Data Type	Representation	Semantics
EV	D	Event. A data record of returned input data, the contents and structure of which depends on the input class associated with the event, as follows: Input Class — E LID index — IX Trigger — IX Timestamp — R Measure Validity — E Measure Value: if input class = LOCATOR: position (P) if input class = STROKE: list of points (in stroke) (nP) if input class = VALUATOR: value (R) if input class = CHOICE: choice number (I) if input class = PICK: list of pick values (nPV) if input class = STRING: string (S) if input class = RASTER: xcount, ycount (2I) list of input colour values (nICO) if input class = GENERAL: data record (D)
EVL	D	Event reports list. A sequence of events (EV) returned as a data record.
FN	IN	Function identifier. Values are specified by annex A of this part of ISO/IEC 9636 to identify standardized functions. Negative values are specified by implementors to identify extensions (private functions).
I		Basic data type, integer. Subject to INTEGER PRECISION within a data stream encoding as controlled by INTEGER PRECISION REQUIREMENT.
ICO		Basic data type, input colour specifier used with RASTER class input. Realization depends on the Colour and Bits Per Colour entries in the RASTER Class Specific Logical Input Device State List. When Colour is YES, realization is identical to CD with local colour precision = Bits Per Colour. When Colour is NO, realization is identical to CI with local colour index precision = Bits Per Colour.
IF8		Basic data type, fixed 8-bit integer.
IF16		Basic data type, fixed 16-bit integer.
IF32		Basic data type, fixed 32-bit integer.
IF	IF8 or IF16 or IF32	Fixed integer. Of fixed precision, 8-bit, 16-bit, or 32-bit. The representation of IF within a binding or encoding may differ for distinct uses of IF. That is, fixed precisions of IF as used for DC, ISC, IN, or ER are not necessarily identical.
IN	IF	Intrinsic name. Specified by ISO/IEC 9636 or the implementation. Each use of this type is always within a context which defines the set of entries with which it is concerned.
ISC	IF	Input surface coordinate in Input Coordinate Space. A fixed precision integer value.
ISP	ISC,ISC	A pair of ISC values giving the x and y coordinates of a point in the input coordinate space.
IX		Basic data type, index. Realized as an integer but subject to INDEX PRECISION within a data stream encoding as controlled by INDEX PRECISION REQUIREMENT. Often the range of permitted values for a particular instance of an index data type will be specified by ISO/IEC 9636. For some instances positive values are reserved for standard use while negative ones are available for private use.

Data Type	Representation	Semantics
P	VDC,VDC	A pair of VDC values giving the x and y coordinates of a point in VDC space.
PN	CSN	Pick identifier.
PRN	IN	Profile identifier.
PV	SN,PN	Pick value. A composite value consisting of a segment identifier (SN) followed by a pick identifier (PN).
R		Basic data type, real. Subject to REAL PRECISION within a data stream encoding as controlled by REAL PRECISION REQUIREMENTS.
S		Basic data type, string. A sequence of characters represented by values which have to be interpreted according to a set of character set and character coding announcer controls (input or output).
SF		Basic data type, string with fixed representation using the fixed character set (ISO 646) and not subject to CHARACTER CODING ANNOUNCER. (E.g. font names.)
SN	CSN	Segment identifier.
SS	R or VDC	Size specification. A real or VDC depending on the particular specification mode. Context determines whether this is for edge width, line width, or marker size and therefore which of the specification modes is relevant.
VC	R or DC	Viewport coordinate in viewport space. Real or integer depending on DEVICE VIEWPORT SPECIFICATION MODE
VDC		Virtual device coordinate in VDC space. Real or integer depending on the VDC TYPE in effect. Subject to VDC INTEGER PRECISION or VDC REAL PRECISION within a data stream encoding as controlled by VDC INTEGER PRECISION REQUIREMENT or VDC REAL PRECISION REQUIREMENTS.
VP	VC,VC	A pair of VC values giving x and y coordinates of a point in viewport space.

Type IX parameters used as selectors in some functions have a fixed number of values with defined and standardized meanings and have other values available for implementation-dependent definition and use. To avoid possible conflict with client-defined or implementation-dependent values, the standardized and private use values are assigned to distinct ranges of the IX parameter. Negative values of IX are available for private use, and non-negative values are reserved for registration or future standardization where not specified herein.

Type E (enumerated) parameters are also extensible for private values, but the method of specifying private values is binding or encoding dependent.

Further types may be constructed from combinations of simple types. Examples of such types are: nP or 2*3*R; these are respectively, indeterminate and fixed repetitions of simple types. Constructions of the form nX implicitly carry the number of elements in the list as well as the elements themselves. Types may also be constructed as tuples of other types (e.g. [I,E,R,E]).

How these data types are represented in an encoding or a binding is defined in the context of the encoding or the binding itself. The client may specify various numerical precision requirements for those CGI data types which have variable precision within a data stream encoding. (See ISO/IEC 9636-2, 3.5.1.)

5.2.11 Formal grammar

The syntax of the CGI functions in a formal grammar is provided in annex A of the part in which the function description appears. Each formal grammar annex forms a normative annex of the corresponding part of ISO/IEC 9636.

5.3 Overview of ISO/IEC 9636-2: Control

ISO/IEC 9636-2 defines those functions of the Computer Graphics Interface concerned with Virtual Device management, coordinate space control, and error control. The functionality incorporated in ISO/IEC 9636-2 is concerned with the management of the graphics image and the interrelationship of the graphical and non-graphical parts of the interface. This functionality is divided into the following areas:

- *Virtual Device management functions,* which allows the CGI client to initiate and terminate sessions of dialogue and to manage the graphics image on the Virtual Device.
- *Coordinate space control functions,* for the establishment of coordinate information, placement of the picture on the drawing surface, and for the management of drawing surface clipping.
- *Error control functions,* which involves the detection of errors both at and subsequent to the transmission of parameters through the CGI.
- *Miscellaneous control functions,* for the establishment of data stream numeric precisions, the accessing of implementation specific functionality, and the accessing of CGI external functions.
- *Control inquiry functions,* which provide access to the description tables and state lists concerned with function and Profile support, device description, and CGI control information.

5.4 Overview of ISO/IEC 9636-3: Output

ISO/IEC 9636-3 covers the device-independent graphic object output functionality of the CGI. It covers primitive functions, attributes, object formation and subsequent processing, and related control and inquiry functionality. This functionality is divided into the following areas:

- *Graphic primitive functions,* which describe the geometry of the components of a picture in the CGI.
- *Attribute functions,* which set modal values in state lists that are used to determine certain properties (including visual aspects) of these geometric picture components.
- *General attribute and output control functions,* which specify the modes of operation of certain other functions, control some aspects of the device's operation with respect to graphic objects and attribute functions, and which provide facilities for the construction of compound objects.
- *Retrieval function,* which returns information useful for the positioning of text objects.
- *Output inquiry functions,* which provide access to the description tables and state lists concerned with output and attributes.

5.5 Overview of ISO/IEC 9636-4: Segments

ISO/IEC 9636-4 specifies how graphic objects may be grouped in segments and be identified by a unique segment identifier. The graphic objects stored in segments are composed of graphic primitives with associated attribute values. ISO/IEC 9636-4 defines a set of functions for creating, modifying, and manipulating segments. This functionality is divided into the following areas:

- *Segment manipulation functions,* including segment creation, deletion, renaming, and copying. Graphic objects may also be appended to the end of an existing segment. Graphic objects within a segment, however, cannot be modified or deleted.
- *Segment attribute functions,* including modification of segment attributes (e.g. transformation, visibility). Some segment attributes affect the rendered appearance of segments and can be a basis for feedback during graphic picture manipulation. Other attributes affect how input concepts and segments are associated in support of pick input.

- *Segment inquiry functions,* by which access is provided to the information in the description tables and state lists concerned with segments.

5.6 Overview of ISO/IEC 9636-5: Input and echoing

ISO/IEC 9636-5 defines those functions of the Computer Graphics Interface concerned with input and echoing. This functionality is divided into the following areas:

- *Input control functions,* which provide control over initialization and deallocation of Logical Input Devices (LIDs), together with the means to tailor their characteristics.
- *Request and sample functions,* which permit LIDs to be used with Request and Sample input methods.
- *Echo request input functions,* which permit LIDs to be used with the Echo Request input method, a special type of Request method in which changes to the LID's measure value can be tracked by the client.
- *Event input functions,* which permit LIDs to be used with the Event input method; this allows the client to control a number of active LIDs simultaneously while concurrently performing graphical output.
- *Echo output functions,* which allow values to be echoed on a given CGI Virtual Device when the source of such values was not the given device.
- *Input and echoing inquiry functions,* which provide access to the description tables and state lists concerned with input and echoing.

5.7 Overview of ISO/IEC 9636-6: Raster

ISO/IEC 9636-6 defines a set of functions for creating, modifying, retrieving, and displaying information stored as pixel data below the CGI. This functionality is divided into the following areas:

- *Raster control functions,* including functions for the creation and deletion of bitmaps, and the selection of drawing and display bitmaps, and the control of raster transparency and mapped bitmap expansion.
- *Raster attribute functions,* for setting particular attributes which have significance with other graphical output, defined in ISO/IEC 9636-3 and ISO/IEC 9636-6, when used in conjunction with raster functionality.
- *Raster operation functions,* including display and retrieval of pixel array data, and various forms of bitmap manipulation operations (bitblts) including movement, combination, and replication of bitmap regions.
- *Raster inquiry functions,* which provide access to the description tables and state lists defined in this part of ISO/IEC 9636.

6 Profiles

Profiles are the means of standardizing CGI implementations having different functional scopes in order to suit a variety of users.

Profiles represent a minimum level of capability that an implementation of a CGI Virtual Device shall provide, and a maximum level of capability that a client program can expect when utilizing such an implementation. An implementation may exceed the minimum requirements of a given profile. In this way a CGI implementation can support multiple profiles, each of which reflects a subset of the capabilities implemented.

Each profile contains sufficient function support and resources for it to be used effectively for its intended purpose independently of other profiles. Given any two profiles, a single CGI Virtual Device may be implemented that satisfies both profiles. This means that a profile may not be defined with any restrictions or assumptions which could prevent this Profile Compatibility Principle.

In general, profiles may impose requirements on resource availability and functional capability which may be interpreted as demands on a memory resource. For implementations which use dynamic memory allocation, the interpretation of the profile may be taken in the sense of total memory needed to satisfy all such requirements. In this way, the same fixed memory resource can be allocated in different manners to satisfy different sets of requirements for different profiles.

Profiles exist for INPUT, OUTPUT and OUTIN Virtual Devices.

6.1 Profile definition

ISO/IEC 9636 defines two types of Profiles, the Foundation Profile and the Constituency Profile.

- Foundation Profiles specify a minimally conforming level of CGI Virtual Device capability. Foundation Profiles may also prove useful for various combinations of device class and processing environments.
- Certain classes of users of ISO/IEC 9636 may require special consideration of their needs in the definition of Virtual Device support. These users may be served by the Constituency Profiles. Such Profiles consist of a particular list of CGI functions and levels of resource support. All Constituency Profiles, either included in ISO/IEC 9636 or registered according to ISO/IEC TR 9973: 1988, shall be supersets of at least one of the standard Foundation Profiles. This means that a Constituency Profile shall include support for all of a Foundation Profile's required functions and meets that Foundation Profile's resource requirements. The CGI Standard defines Profiles for the following constituencies:
  - GKS implementations;
  - CGM interpreters;
  - Advanced 1-Way (rich output devices).

A Constituency Profile may be specified to be the union of two other Profiles. The meaning of this union is as follows: Profile C is the union of Profiles A and B if an implementation which satisfies C also satisfies A and B; furthermore, any implementation which fails to satisfy C also fails to satisfy at least one of A or B.

The following, 6.4, 6.5, 6.6, and 6.7, describe the list of functions and resource requirements of the current Foundation and Constituency Profiles.

The following information is contained in the definition of a Profile:

a) A list of functions which are required to be present. Such a list shall be complete in the sense that a Virtual Device implementing only those functions can be used effectively for its intended purpose. It shall not contain functions that are useless in the absence of any function that is not listed.

b) A statement of the minimum support requirements. Note that these minimum values represent the maximum capability a client program may expect to be available in addition to the default values.

c) Restrictions on the behaviour of particular functions when latitude is otherwise allowed by ISO/IEC 9636.

d) A list of required GDP, ESCAPE, and GET ESCAPE functions. These functions shall themselves be registered before they can be included in a Profile.

e) A name is used to identify the Profile, whether the Profile is defined in ISO/IEC 9636 or is registered at a later date.

Subject to approval by the Registration Authority, constituencies may impose other requirements in Profiles, so long as such requirements do not violate any of the rules for Profiles stated in ISO/IEC 9636. Where ISO/IEC 9636 allows latitude in the behaviour of a function or a feature (e.g. the shape of the pick aperture, dynamic modification of bundles, supported bitmap formats) there is a corresponding description table entry stating the implemented behaviour. For these functions or features, ISO/IEC 9636 indicates the preferred behaviour, which provides the highest quality or promotes the most efficient use. Within a description table entry specifying a particular behaviour in a case of allowed latitude, the enumerated value which signifies the preferred behaviour (normally the last entry in the list) is underlined. A Profile shall not require a behaviour which conflicts with the preferred behaviour, i.e. a less efficient or lower quality implementation. A Profile shall not require a particular behaviour where ISO/IEC 9636 explicitly states that the behaviour is implementation-dependent (e.g. the shape of line end caps). A Profile shall not specify a state list default value if this is mandated by ISO/IEC 9636, nor where ISO/IEC 9636 specifies such a default as implementation-dependent. However, if the default is subject to allowed latitude, then a Profile may specify it's default value, which may differ from the preferred default. (For example VDC Type.)

A contrasting approach is taken by ISO/IEC 9636 where there is provision of a range of optional capabilities for a function or feature (e.g. interior styles, hatch styles). In this case, the implementation support for the function or feature is recorded in a description table and a Profile may require a minimum support level for this function or feature.

The allowed latitudes of primitive functions are described in ISO/IEC 9636-3, clause 3.

It is recommended that where an implementation can support more than one behaviour of a set of allowed latitudes, that a private ESCAPE function is used to modify the behaviour, that a private state list entry records the current setting, and that a private GET ESCAPE function provide an inquiry of this state list entry.

6.1.1 Constituency Profile registration

The Profiles included in ISO/IEC 9636 are not expected to satisfy all future needs. Other Profiles may be registered, when required, in the ISO Register of Graphical Items.

6.2 Foundation Profiles

The Foundation Profiles <input class> INPUT provides a minimal set of functions for an input Virtual Device using two-way communications. Similarly, the 2-WAY OUTPUT Foundation Profile provides a minimal set of functions for an output Virtual Device using two-way communications. All of these Profiles include inquiry of the Virtual Device capabilities.

The 1-WAY OUTPUT Profile provides a minimum set of functions for an OUTPUT or OUTIN Virtual Device, in situations where inquiry is not possible because communication is one-way only. This may occur quite frequently in practice. For example:

- When the I/O connection is physically incapable of two-way communication.
- When the device is accessed via a network or spooler that does not provide two-way communication.
- When the output to a device is being pipelined.
- When the client chooses to operate in a 1-way output mode because the overhead in synchronizing output and response is high.
- When the implementation does not support soliciting functions.

The following Foundation Profiles are defined:

2-WAY OUTPUT
1-WAY OUTPUT
LOCATOR INPUT
STROKE INPUT
VALUATOR INPUT
CHOICE INPUT
STRING INPUT
RASTER INPUT

6.3 Constituency Profiles

6.3.1 GKS Constituency Profiles

The GKS Constituency Profiles are designed to provide reasonable support for implementing the various GKS (ISO 7942) INPUT, OUTPUT and OUTIN workstations for the different standard GKS levels. The defined GKS Profiles and their corresponding CGI Virtual Device class are shown in table 2.

Table 2 – GKS Constituency Profiles

Profile Name	GKS Level	CGI Virtual Device
GKS OUTPUT-0	0	OUTPUT or OUTIN
GKS OUTPUT-1	1,2	OUTPUT or OUTIN
GKS INPUT-b	0b, 1b, 2b	INPUT or OUTIN
GKS INPUT-c	0c, 1c, 2c	INPUT or OUTIN
GKS OUTIN-0b	0b	OUTIN
GKS OUTIN-0c	0c	OUTIN
GKS OUTIN-1b	0b,1b, 2b	OUTIN
GKS OUTIN-1c	0c,1c, 2c	OUTIN

6.3.2 CGM Constituency Profile

The CGM Constituency Profile is defined to provide reasonable support for implementing interpreters for basic CGM (ISO 8632). The BASIC CGM Constituency Profile's corresponding CGI device class is OUTPUT.

6.3.3 Advanced 1-Way Output Constituency Profile

The ADVANCED 1-WAY OUTPUT Constituency Profile is designed to provide reasonable support for an advanced output device. It guarantees correct interpretation of most primitives and attributes (except for colour) of the CGI, but does not support inquiries.

6.4 Foundation Profile definition

Each Foundation Profile described provides the required information as defined in 6.1.

6.4.1 2-WAY OUTPUT Foundation Profile

List of required functions

Virtual Device Management:

INITIALIZE	TERMINATE
EXECUTE DEFERRED ACTIONS	PREPARE DRAWING SURFACE
END PAGE	

Coordinate Space Control:

VDC EXTENT	DEVICE VIEWPORT

Primitives:

POLYLINE	TEXT

Attributes:

LINE TYPE	CHARACTER HEIGHT

Foundation Profile definition

Device Identity Description Table:
INQUIRE DEVICE IDENTIFICATION

Output Device Description Table:
INQUIRE DEVICE DESCRIPTION

Function and Profile Support Description Table:
LOOKUP FUNCTION SUPPORT
LOOKUP PROFILE SUPPORT

Control Description Table:
INQUIRE DEVICE CONTROL CAPABILITY

Line Description Table:
INQUIRE LIST OF AVAILABLE LINE TYPES

Text Description Table:
INQUIRE LIST OF AVAILABLE CHARACTER HEIGHTS

General Attributes and Output Control State List:
INQUIRE OBJECT CLIPPING

List of ESCAPE, GDP and GET ESCAPE functions required

No ESCAPE, GET ESCAPE or GDP functions are required for this Profile.

Resource requirements

Available Line Types	(solid, dash, dot, dash-dot)

Definition of restrictions

No additional restriction on the behaviour of any functions.

6.4.2 1-WAY OUTPUT Foundation Profile

This Foundation Profile provides the same output functional capability as the 2-WAY OUTPUT Profile except for support of inquiries.

List of required functions

Virtual Device Management:
INITIALIZE
EXECUTE DEFERRED ACTIONS
END PAGE
TERMINATE
PREPARE DRAWING SURFACE

Coordinate Space Control:
VDC EXTENT
DEVICE VIEWPORT

Primitives:
POLYLINE
TEXT

Attributes:
LINE TYPE
CHARACTER HEIGHT

List of ESCAPE, GDP and GET ESCAPE functions required

No ESCAPE, GET ESCAPE or GDP functions are required for this Profile.

Resource requirements

Available Line Types	(solid, dash, dot, dash-dot)

Definition of restrictions

No additional restrictions on the behaviour of any functions.

6.4.3 <input class> INPUT Foundation Profile

Foundation Profiles are defined for CGI Virtual Devices which contain at least one LID of the following CGI input classes: LOCATOR, STROKE, VALUATOR, CHOICE, STRING and RASTER. For each Profile, the specific input class is substituted for each occurrence of <input class>.

NOTE – There is no Foundation Profile for PICK (which would require segments) or for GENERAL.

List of required functions

Virtual Device Management:

INITIALIZE	TERMINATE

Input Control:

INITIALIZE LOGICAL INPUT DEVICE	RELEASE LOGICAL INPUT DEVICE
ECHO CONTROLS	

Request and Sample:

REQUEST <input class>

For CGI input classes STROKE, STRING, and RASTER:

GET ADDITIONAL <input class> DATA

Device Identity Description Table:

INQUIRE DEVICE IDENTIFICATION

Output Device Description Table:

INQUIRE DEVICE DESCRIPTION

Function and Profile Support Description Table:

LOOKUP FUNCTION SUPPORT	LOOKUP PROFILE SUPPORT

Input Description Table:

INQUIRE INPUT CAPABILITY

Class-Independent Logical Input Device Description Table:

INQUIRE COMMON INPUT DEVICE PROPERTIES	INQUIRE LIST OF SUPPORTED ECHO TYPES
INQUIRE LIST OF SUPPORTED PROMPT TYPES	
INQUIRE LIST OF SUPPORTED ACKNOWLEDGEMENT TYPES	

List of ESCAPE, GDP and GET ESCAPE functions required

No ESCAPE, GET ESCAPE or GDP functions are required for these Profiles.

Resource requirements

For Input Class STROKE:	
Number of Points	(64)
For Input Class STRING:	
String Buffer Size	(72)
Available Input Character Sets	(1) based on ISO 646

Definition of restrictions

No additional restrictions on the behaviour of any functions.

6.5 GKS Profile definition

All GKS Profiles described provide the required information as defined in 6.1.

6.5.1 GKS OUTPUT-0 Profile

GKS OUTPUT-0 Profile has been designed to satisfy the requirements for implementing a workstation for a GKS level 0 implementation. It includes the 2-WAY OUTPUT Foundation Profile.

GKS Profile definition

List of required functions

Virtual Device Management:

- INITIALIZE
- TERMINATE
- EXECUTE DEFERRED ACTIONS
- PREPARE DRAWING SURFACE
- END PAGE

Coordinate Space Control:

- VDC EXTENT
- DEVICE VIEWPORT
- DEVICE VIEWPORT SPECIFICATION MODE

Errors:

- DEQUEUE ERROR REPORT
- ERROR HANDLING CONTROL

Miscellaneous Control:

- STATE LIST INQUIRY SOURCE

Primitives:

- POLYLINE
- POLYMARKER
- TEXT
- POLYGON
- CELL ARRAY

Attributes:

- LINE BUNDLE INDEX
- MARKER BUNDLE INDEX
- LINE TYPE
- MARKER TYPE
- LINE WIDTH
- MARKER SIZE
- TEXT BUNDLE INDEX
- FILL BUNDLE INDEX
- TEXT FONT INDEX
- TEXT PRECISION
- CHARACTER EXPANSION FACTOR
- CHARACTER SPACING
- CHARACTER HEIGHT
- CHARACTER ORIENTATION
- TEXT PATH
- TEXT ALIGNMENT
- INTERIOR STYLE

If the CGI device supports colour, then the following functions are required:

- LINE COLOUR
- MARKER COLOUR
- TEXT COLOUR
- FILL COLOUR

If the CGI device supports hatching, then the following function is required:

- HATCH INDEX

If the CGI device supports pattern, then the following functions are required:

- PATTERN INDEX
- FILL REFERENCE POINT
- PATTERN SIZE

General Attribute and Output Control:

- CLIP INDICATOR
- CLIP RECTANGLE

The following function is required if colour is supported:

- COLOUR TABLE
- ASPECT SOURCE FLAGS

Retrieval:

- GET TEXT EXTENT

Raster Operation:

If the CGI device is a raster device able to return this information, the following functions are required:

- GET PIXEL ARRAY
- GET PIXEL ARRAY DIMENSIONS

Device Identity Description Table:

- INQUIRE DEVICE IDENTIFICATION

Output Device Description Table:

- INQUIRE DEVICE DESCRIPTION

Function and Profile Support Description Table:

LOOKUP FUNCTION SUPPORT	LOOKUP PROFILE SUPPORT

Control Description Table:
INQUIRE DEVICE CONTROL CAPABILITY

Control State List:

INQUIRE CONTROL STATE	INQUIRE VDC TO DEVICE MAPPING

Primitive Support Description Table:
LOOKUP GDP SUPPORT

Line Description Table:

INQUIRE LINE CAPABILITIES	INQUIRE LIST OF AVAILABLE LINE TYPES
INQUIRE LIST OF AVAILABLE SCALED LINE WIDTHS	

Marker Description Table:

INQUIRE MARKER CAPABILITY	INQUIRE LIST OF AVAILABLE MARKER TYPES
INQUIRE LIST OF AVAILABLE SCALED MARKER SIZES	

Text Description Table:

INQUIRE TEXT CAPABILITY	INQUIRE LIST OF AVAILABLE TEXT FONTS

INQUIRE LIST OF AVAILABLE CHARACTER EXPANSION FACTORS
INQUIRE LIST OF AVAILABLE CHARACTER HEIGHTS

Fill Description Table:

INQUIRE FILL CAPABILITY	INQUIRE LIST OF AVAILABLE HATCH STYLES

Output Control Description Table:
INQUIRE COLOUR CAPABILITIES

Line Attribute State List:

INQUIRE LINE ATTRIBUTES	INQUIRE LINE REPRESENTATION

NOTE – INQUIRE LINE REPRESENTATION is required to determine the content of the predefined bundles.

Marker Attribute State List:

INQUIRE MARKER ATTRIBUTES	INQUIRE MARKER REPRESENTATION

NOTE – INQUIRE MARKER REPRESENTATION is required to determine the content of the predefined bundles.

Text Attribute State List:

INQUIRE TEXT ATTRIBUTES	INQUIRE TEXT REPRESENTATION

NOTE – INQUIRE TEXT REPRESENTATION is required to determine the content of the predefined bundles.

Fill Attribute State List:

INQUIRE FILL ATTRIBUTES	INQUIRE FILL REPRESENTATION

NOTE – INQUIRE FILL REPRESENTATION is required to determine the content of the predefined bundles.

The following function is required if pattern is supported:
INQUIRE PATTERN

General Attributes and Output Control State List:

INQUIRE OBJECT CLIPPING	LOOKUP ASPECT SOURCE FLAGS

The following function is required if colour is supported:
INQUIRE LIST OF COLOUR TABLE ENTRIES

NOTE – GKS allows some functional capabilities to be not supported. However, the entry points for these functions should be provided by the implementation and a class 4 error generated if the client attempts to use them via a CGI language binding interface.

List of ESCAPE, GDP and GET ESCAPE functions required

No ESCAPE, GET ESCAPE or GDP functions are required by this Profile. However, where the implementation presents a procedural library interface then the entry points for GDP, ESCAPE, and GET ESCAPE shall be present.

Resource requirements

Number of Predefined Line Bundles	(5)
Available Line Types	(solid, dash, dot, dash-dot)
Number of Predefined Marker Bundles	(5)
Available Marker Types	(dot, plus, asterisk, circle, cross)
Number of Predefined Text Bundles	(2)
Available STRING Precision Text Fonts	(1) with:
Supported Horizontal Text Alignments	(NORMAL HORIZONTAL, LEFT, CENTRE, RIGHT)
Supported Vertical Text Alignments	(NORMAL VERTICAL, CAP, TOP, HALF, BOTTOM, BASE)
Available CHARACTER Precision Text Fonts	(1) with:
Supported Horizontal Text Alignments	(NORMAL HORIZONTAL, LEFT, CENTRE, RIGHT)
Supported Vertical Text Alignments	(NORMAL VERTICAL, CAP, TOP, HALF, BOTTOM, BASE)
Number of Points for Polygon	(128)
Number of Predefined Fill Bundles	(5) bundles to be distinguishable
Available Interior Styles	(HOLLOW)
Available Hatch Styles	(3) if HATCH is supported
Number of Predefined Patterns	(1) if PATTERN is supported

Definition of restrictions

GKS requires that predefined bundle elements are contiguous and starts at the beginning of the table (i.e. 0 for colour table, 1 for the others) and that all required predefined bundle elements have different and distinguishable representations. There is no distinction between fonts and character sets in GKS. So, a GKS font may correspond to a pair (CGI font, character set).

6.5.2 GKS OUTPUT-1 Profile

GKS OUTPUT-1 Profile has been designed to satisfy the requirements for implementing a workstation for a GKS level 1 or 2 implementation. It includes the 2-WAY OUTPUT Foundation Profile.

List of required functions

Virtual Device Management:

INITIALIZE	TERMINATE
EXECUTE DEFERRED ACTIONS	DEFERRAL MODE
PREPARE DRAWING SURFACE	END PAGE

Coordinate Space Control:

VDC EXTENT	DEVICE VIEWPORT
DEVICE VIEWPORT SPECIFICATION MODE	

Errors:

DEQUEUE ERROR REPORT	ERROR HANDLING CONTROL

Miscellaneous Control:

MESSAGE	STATE LIST INQUIRY SOURCE

Primitives:

POLYLINE	POLYMARKER
TEXT	POLYGON
CELL ARRAY	

Attributes:

LINE BUNDLE INDEX	MARKER BUNDLE INDEX
LINE TYPE	MARKER TYPE
LINE WIDTH	MARKER SIZE
TEXT BUNDLE INDEX	FILL BUNDLE INDEX
TEXT FONT INDEX	TEXT PRECISION
CHARACTER EXPANSION FACTOR	CHARACTER SPACING
CHARACTER HEIGHT	CHARACTER ORIENTATION
TEXT PATH	TEXT ALIGNMENT

INTERIOR STYLE

If the CGI device supports colour, then the following functions are required:

LINE COLOUR	MARKER COLOUR
TEXT COLOUR	FILL COLOUR

If the CGI device supports hatching, then the following function is required:

HATCH INDEX

If the CGI device supports patterns, then the following functions are required:

PATTERN INDEX	FILL REFERENCE POINT
PATTERN SIZE	

General Attribute and Output Control Functions:

CLIP INDICATOR	CLIP RECTANGLE

The following function is required if colour is supported:

COLOUR TABLE

LINE REPRESENTATION	MARKER REPRESENTATION
TEXT REPRESENTATION	FILL REPRESENTATION
ASPECT SOURCE FLAGS	PATTERN TABLE

Retrieval:

GET TEXT EXTENT

Raster Operation:

If the CGI device is a raster device able to return this information, the following functions are required:

GET PIXEL ARRAY	GET PIXEL ARRAY DIMENSIONS

Segment Manipulation:

CREATE SEGMENT	CLOSE SEGMENT
DELETE SEGMENT	DELETE ALL SEGMENTS
RENAME SEGMENT	DRAW ALL SEGMENTS
IMPLICIT SEGMENT REGENERATION MODE	RESET REGENERATION PENDING

Segment Attribute:

SEGMENT VISIBILITY	SEGMENT TRANSFORMATION
SEGMENT HIGHLIGHTING	SEGMENT DISPLAY PRIORITY

Device Identity Description Table:

INQUIRE DEVICE IDENTIFICATION

Output Device Description Table:

INQUIRE DEVICE DESCRIPTION

Function and Profile Support Description Table:

LOOKUP FUNCTION SUPPORT	LOOKUP PROFILE SUPPORT

Control Description Table:

INQUIRE DEVICE CONTROL CAPABILITY

Control State List:

INQUIRE CONTROL STATE	INQUIRE VDC TO DEVICE MAPPING

Primitive Support Description Table:

LOOKUP GDP SUPPORT

Line Description Table:

INQUIRE LINE CAPABILITIES	INQUIRE LIST OF AVAILABLE LINE TYPES
INQUIRE LIST OF AVAILABLE SCALED LINE WIDTHS	

Marker Description Table:

INQUIRE MARKER CAPABILITY	INQUIRE LIST OF AVAILABLE MARKER TYPES
INQUIRE LIST OF AVAILABLE SCALED MARKER SIZES	

Text Description Table:

INQUIRE TEXT CAPABILITY	INQUIRE LIST OF AVAILABLE TEXT FONTS

INQUIRE LIST OF AVAILABLE CHARACTER EXPANSION FACTORS
INQUIRE LIST OF AVAILABLE CHARACTER HEIGHTS

Fill Description Table:
INQUIRE FILL CAPABILITY
INQUIRE LIST OF AVAILABLE HATCH STYLES

Output Control Description Table:
INQUIRE COLOUR CAPABILITIES

Segment Description Table:
INQUIRE SEGMENT CAPABILITY

Line Attribute State List:
INQUIRE LINE ATTRIBUTES
INQUIRE LINE REPRESENTATION

Marker Attribute State List:
INQUIRE MARKER ATTRIBUTES
INQUIRE MARKER REPRESENTATION

Text Attribute State List:
INQUIRE TEXT ATTRIBUTES
INQUIRE TEXT REPRESENTATION

Fill Attribute State List:
INQUIRE FILL ATTRIBUTES
INQUIRE FILL REPRESENTATION

The following function is required if pattern is supported:
INQUIRE PATTERN DIMENSIONS
INQUIRE PATTERN

General Attributes and Output Control State List:
INQUIRE OBJECT CLIPPING
LOOKUP ASPECT SOURCE FLAGS

The following function is required if colour is supported:
INQUIRE LIST OF COLOUR TABLE ENTRIES

Segment State List:
INQUIRE SEGMENT STATE

Individual Segment State List:
INQUIRE INDIVIDUAL SEGMENT STATE

NOTE – GKS allows some functional capabilities to be not supported. However, the entry points for these functions should be provided by the implementation and a class 4 error generated if the client attempts to use them via a CGI language binding interface.

List of ESCAPE, GDP and GET ESCAPE functions required

No ESCAPE, GET ESCAPE or GDP functions are required by this Profile. However, where the implementation presents a procedural library interface then the entry points for GDP, ESCAPE, and GET ESCAPE shall be present.

Resource requirements

Number of Characters for Message	(80)
Number of Predefined Line Bundles	(5)
Number of Settable Line Bundles	(20)
Available Line Types	(solid, dash, dot, dash-dot)
Number of Predefined Marker Bundles	(5)
Number of Settable Marker Bundles	(20)
Available Marker Types	(dot, plus, asterisk, circle, cross)
Number of Predefined Text Bundles	(6)
Number of Settable Text Bundles	(20)
Available STRING Precision Text Fonts	(1) with:
Supported Horizontal Text Alignments	(NORMAL HORIZONTAL, LEFT, CENTRE, RIGHT)
Supported Vertical Text Alignments	(NORMAL VERTICAL, CAP, TOP, HALF, BOTTOM, BASE)
Available CHARACTER Precision Text Fonts	(1) with:
Supported Horizontal Text Alignments	(NORMAL HORIZONTAL, LEFT, CENTRE, RIGHT)
Supported Vertical Text Alignments	(NORMAL VERTICAL, CAP, TOP, HALF, BOTTOM, BASE)
Available STROKE Precision Text Fonts	(2) with:

Supported Horizontal Text Alignments	(NORMAL HORIZONTAL, LEFT, CENTRE, RIGHT)
Supported Vertical Text Alignments	(NORMAL VERTICAL, CAP, TOP, HALF, BOTTOM, BASE)
Number of Points for Polygon	(128)
Number of Predefined Fill Bundles	(5) GKS requires these bundles to be distinguishable
Number of Settable Fill Bundles	(10)
Available Interior Styles	(HOLLOW)
Available Hatch Styles	(3) if HATCH is supported
Number of Predefined Patterns	(1) if PATTERN is supported
Number of Settable Patterns	(10) if PATTERN is supported

Definition of restrictions

GKS requires that the indices of predefined bundles be continuous and start at the beginning of the table (i.e. 0 for colour table, 1 for the others) and that all required predefined bundle elements shall have different and distinguishable representations. There is no distinction between fonts and character sets in GKS. So, a GKS font may correspond to a pair (CGI font, character set).

6.5.3 <input class> GKS INPUT-b Profile

This Profile provides support for an input workstation of a GKS level 0b, 1b or 2b. Any occurrence of <input class> should be replaced by one of the following CGI input classes: LOCATOR, STROKE, VALUATOR, CHOICE, and STRING.

List of required functions

Virtual Device Management:
INITIALIZE — TERMINATE

Errors:
DEQUEUE ERROR REPORTS — ERROR HANDLING CONTROL

Miscellaneous Control:
STATE LIST INQUIRY SOURCE

Input Control:
INITIALIZE LOGICAL INPUT DEVICE — RELEASE LOGICAL INPUT DEVICE
ECHO CONTROLS — PUT CURRENT <input class> MEASURE
ECHO DATA — <input class> DEVICE DATA

Request and Sample:
REQUEST <input class>

For CGI input classes STROKE and STRING:
GET ADDITIONAL <input class> DATA

Device Identity Description Table:
INQUIRE DEVICE IDENTIFICATION

Output Device Description Table:
INQUIRE DEVICE DESCRIPTION

Function and Profile Support Description Table:
LOOKUP FUNCTION SUPPORT — LOOKUP PROFILE SUPPORT

Input Description Table:
INQUIRE INPUT CAPABILITY

Class-Independent Logical Input Device Description Table:
INQUIRE COMMON INPUT DEVICE PROPERTIES — INQUIRE LIST OF SUPPORTED ECHO TYPES
INQUIRE LIST OF SUPPORTED PROMPT TYPES
INQUIRE LIST OF SUPPORTED ACKNOWLEDGEMENT TYPES

Class-Specific Logical Input Device Description Table:
INQUIRE <input class> CAPABILITIES

Class-Independent Logical Input Device State:

INQUIRE COMMON LOGICAL INPUT DEVICE STATE	INQUIRE ECHO DATA RECORD
INQUIRE INPUT DEVICE DATA RECORD	

Class-Specific Logical Input Device State:
INQUIRE <input class> STATE

List of ESCAPE and GET ESCAPE functions required

No ESCAPE or GET ESCAPE functions are required by this Profile. However, where the implementation presents a procedural library interface then the entry points for ESCAPE or GET ESCAPE shall be present.

Resource requirements

A GKS Input workstation shall contain at least one input device. If it contains several, all <input class> functions shall be present for each input device.

For Input Class STROKE:

Number of Points	(64)

For Input Class STRING:

String Buffer Size	(72)
Available Input Character Sets	(1) based on ISO 646

Definition of restrictions

None

6.5.4 <input class> GKS INPUT-c Profile

This Profile provides support for an input workstation of a GKS level 0c, 1c or 2c. Any occurrence of <input class> should be replaced by one of the following CGI input classes: LOCATOR, STROKE, VALUATOR, CHOICE, and STRING.

List of required functions

Virtual Device Management:

INITIALIZE	TERMINATE

Errors:

DEQUEUE ERROR REPORTS	ERROR HANDLING CONTROL

Miscellaneous Control:
STATE LIST INQUIRY SOURCE

Input Control:

INITIALIZE LOGICAL INPUT DEVICE	RELEASE LOGICAL INPUT DEVICE
ECHO CONTROLS	ECHO DATA
PUT CURRENT <input class> MEASURE	<input class> DEVICE DATA

Request and Sample:

REQUEST <input class>	SAMPLE <input class>
SAMPLING STATE	

For CGI input classes STROKE and STRING:
GET ADDITIONAL <input class> DATA

Event Input:

INITIALIZE EVENT QUEUE	RELEASE EVENT QUEUE
ENABLE EVENTS	DISABLE EVENTS
EVENT QUEUE BLOCK CONTROL	FLUSH EVENTS
AWAIT EVENT	DEQUEUE <input class> EVENT

Device Identity Description Table:
INQUIRE DEVICE IDENTIFICATION

Output Device Description Table:

INQUIRE DEVICE DESCRIPTION

Function and Profile Support Description Table:

LOOKUP FUNCTION SUPPORT
LOOKUP PROFILE SUPPORT

Input Description Table:

INQUIRE INPUT CAPABILITY

Class-Independent Logical Input Device Description Table:

INQUIRE COMMON INPUT DEVICE PROPERTIES
INQUIRE LIST OF SUPPORTED ECHO TYPES
INQUIRE LIST OF SUPPORTED PROMPT TYPES
INQUIRE LIST OF SUPPORTED ACKNOWLEDGEMENT TYPES

Class-Specific Logical Input Device Description Table:

INQUIRE <input class> CAPABILITIES

Class-Independent Logical Input Device State:

INQUIRE COMMON LOGICAL INPUT DEVICE STATE
INQUIRE ECHO DATA RECORD
INQUIRE INPUT DEVICE DATA RECORD

Class-Specific Logical Input Device State:

INQUIRE <input class> STATE

Event Input State:

INQUIRE EVENT INPUT STATE

List of ESCAPE and GET ESCAPE functions required

No ESCAPE or GET ESCAPE functions are required by this Profile. However, where the implementation presents a procedural library interface then the entry points for ESCAPE or GET ESCAPE shall be present.

Resource requirements

A GKS Input workstation shall contain at least one input device. If it contains several, all <input class> functions shall be present for each input device.

For Input Class STROKE:	
Number of Points	(64)
For Input Class STRING:	
String Buffer Size	(72)
Available Input Character Sets	(1) based on ISO 646

Definition of restrictions

None.

6.5.5 GKS OUTIN-0b Profile

This Profile provides support for an OUTIN workstation of a GKS level 0b. This Profile is obtained simply by the union of GKS OUTPUT-0 Profile and one or more <input class> GKS INPUT-b Profile.

6.5.6 GKS OUTIN-0c Profile

This Profile provides support for an OUTIN workstation of a GKS level 0c. This Profile is obtained simply by the union of GKS OUTPUT-0 Profile and one or more <input class> GKS INPUT-c Profile.

6.5.7 GKS OUTIN-1b Profile

This Profile provides support for an OUTIN workstation of a GKS level 1b and 2b. This Profile is obtained simply by the union of GKS OUTPUT-1 Profile and one or more <input class> GKS INPUT-b Profile, with the inclusion of the following functions if pick input is supported:

PICK IDENTIFIER
SEGMENT DETECTABILITY

SEGMENT PICK PRIORITY

and all functions with <input class> replaced by PICK.

6.5.8 GKS OUTIN-1c Profile

This Profile provides support for an OUTIN workstation of a GKS level 1c and 2c. This Profile is obtained simply by the union of GKS OUTPUT-1 Profile and one or more <input class> GKS INPUT-c Profile, with the inclusion of the following functions if pick input is supported:

PICK IDENTIFIER	SEGMENT DETECTABILITY
SEGMENT PICK PRIORITY	

and all functions with <input class> replaced by PICK.

6.6 CGM Profile definition

The CGM Profile described provides the required information as defined in 6.1.

6.6.1 BASIC CGM Profile

This Profile provides support for an interpreter of metafiles which conform to ISO 8632 (CGM). It includes the 1-WAY OUTPUT Foundation Profile.

List of required functions

Virtual Device Management:

INITIALIZE	TERMINATE
PREPARE DRAWING SURFACE	EXECUTE DEFERRED ACTIONS
END PAGE	

Coordinate Space Control:

VDC TYPE	VDC INTEGER PRECISION REQUIREMENT
VDC REAL PRECISION REQUIREMENTS	VDC EXTENT
DEVICE VIEWPORT	DEVICE VIEWPORT SPECIFICATION MODE
DEVICE VIEWPORT MAPPING	

Miscellaneous Control:

MESSAGE

Primitives:

POLYLINE	DISJOINT POLYLINE
CIRCULAR ARC 3 POINT	CIRCULAR ARC CENTRE
ELLIPTICAL ARC	POLYMARKER
TEXT	RESTRICTED TEXT
APPEND TEXT	POLYGON
POLYGON SET	RECTANGLE
CIRCLE	CIRCULAR ARC 3 POINT CLOSE
CIRCULAR ARC CENTRE CLOSE	ELLIPSE
ELLIPTICAL ARC CLOSE	CELL ARRAY

Attributes:

LINE BUNDLE INDEX	MARKER BUNDLE INDEX
LINE TYPE	MARKER TYPE
LINE WIDTH	MARKER SIZE
LINE COLOUR	MARKER COLOUR
TEXT BUNDLE INDEX	FILL BUNDLE INDEX
TEXT FONT INDEX	INTERIOR STYLE
TEXT PRECISION	FILL COLOUR

CHARACTER EXPANSION FACTOR	HATCH INDEX
CHARACTER SPACING	PATTERN INDEX
TEXT COLOUR	FILL REFERENCE POINT
CHARACTER HEIGHT	PATTERN SIZE
CHARACTER ORIENTATION	EDGE BUNDLE INDEX
TEXT PATH	EDGE TYPE
TEXT ALIGNMENT	EDGE WIDTH
CHARACTER SET INDEX	EDGE COLOUR
ALTERNATE CHARACTER SET INDEX	EDGE VISIBILITY
CHARACTER CODING ANNOUNCER	

General Attribute and Output Control:

CLIP INDICATOR	CLIP RECTANGLE
LINE WIDTH SPECIFICATION MODE	EDGE WIDTH SPECIFICATION MODE
MARKER SIZE SPECIFICATION MODE	COLOUR SELECTION MODE
COLOUR VALUE EXTENT	AUXILIARY COLOUR
TRANSPARENCY	COLOUR TABLE
ASPECT SOURCE FLAGS	PATTERN TABLE
FONT LIST	CHARACTER SET LIST

List of ESCAPE, GDP and GET ESCAPE functions required

No ESCAPE, GET ESCAPE or GDP functions are required by this Profile. However, where the implementation presents a procedural library interface then the entry points for GDP, ESCAPE, and GET ESCAPE shall be present.

Resource requirements

Supported Device Viewport Specification Modes	(FRACTION OF DISPLAY SURFACE, MILLIMETRES WITH SCALE FACTOR, PHYSICAL DEVICE UNITS)
Number of Characters for Message	(80)
Number of Predefined Line Bundles	(5)
Available Line Types	(solid, dash, dot, dash-dot, dash-dot-dot)
Number of Predefined Marker Bundles	(5)
Available Marker Types	(dot, plus, asterisk, circle, cross)
Available Scaled Marker Sizes	(continuous)
Number of Predefined Text Bundles	(2)
Available STRING Precision Text Fonts	(1) with:
Supported Horizontal Text Alignments	(NORMAL HORIZONTAL, LEFT, CENTRE, RIGHT)
Supported Vertical Text Alignments	(NORMAL VERTICAL, CAP, TOP, HALF, BOTTOM, BASE)
Number of Points for Polygon	(128)
Number of Predefined Fill Bundles	(5)
Available Interior Styles	(HOLLOW, SOLID, HATCH, PATTERN, EMPTY)
Available Hatch Styles	(horizontal, vertical, positive slope, negative slope, horizontal/vertical crosshatch, positive slope/negative slope crosshatch)
Number of Predefined Patterns	(1)
Available Edge Types	(solid, dash, dot, dash-dot, dash-dot-dot)
Number of Predefined Edge Bundles	(5)

Definition of restrictions

CGM recommends that all required predefined bundle elements shall have different and distinguishable representations. Since this Profile is only one-way and does not include inquiries, it is mandatory that the indices of predefined bundles be contiguous and start at the beginning of the table (i.e. index 0 for colour table, index 1 for others).

6.7 Other Constituency Profile definitions

One other Constituency Profile is defined in ISO/IEC 9636. Others Profiles may be registered.

6.7.1 ADVANCED 1-WAY OUTPUT Profile

This Profile provides support for a device which cannot respond to inquiries, but guarantees an important richness of supported capabilities. It includes the 1-WAY OUTPUT Foundation Profile.

List of required functions

Virtual Device Management:

INITIALIZE	TERMINATE
EXECUTE DEFERRED ACTIONS	PREPARE DRAWING SURFACE
END PAGE	

Coordinate Space Control:

VDC EXTENT	DEVICE VIEWPORT
DEVICE VIEWPORT SPECIFICATION MODE	DEVICE VIEWPORT MAPPING

Primitives:

POLYLINE	DISJOINT POLYLINE
CIRCULAR ARC 3 POINT	CIRCULAR ARC CENTRE
ELLIPTICAL ARC	CONNECTING EDGE
POLYMARKER	
TEXT	RESTRICTED TEXT
APPEND TEXT	POLYGON
POLYGON SET	RECTANGLE
CIRCLE	CIRCULAR ARC 3 POINT CLOSE
CIRCULAR ARC CENTRE CLOSE	ELLIPSE
ELLIPTICAL ARC CLOSE	
CELL ARRAY	

Attributes:

LINE BUNDLE INDEX	LINE TYPE
LINE WIDTH	LINE COLOUR
MARKER BUNDLE INDEX	MARKER TYPE
MARKER SIZE	MARKER COLOUR
TEXT BUNDLE INDEX	TEXT FONT INDEX
TEXT PRECISION	CHARACTER EXPANSION FACTOR
CHARACTER SPACING	TEXT COLOUR
CHARACTER HEIGHT	CHARACTER ORIENTATION
TEXT PATH	TEXT ALIGNMENT
CHARACTER SET INDEX	ALTERNATE CHARACTER SET INDEX
FILL BUNDLE INDEX	INTERIOR STYLE
FILL COLOUR	HATCH INDEX
PATTERN INDEX	FILL REFERENCE POINT
PATTERN SIZE	EDGE BUNDLE INDEX
EDGE TYPE	EDGE WIDTH
EDGE COLOUR	EDGE VISIBILITY

General Attribute and Output Control Functions:

CLIP INDICATOR	CLIP RECTANGLE
COLOUR TABLE	LINE WIDTH SPECIFICATION MODE
EDGE WIDTH SPECIFICATION MODE	MARKER SIZE SPECIFICATION MODE
LINE REPRESENTATION	MARKER REPRESENTATION
TEXT REPRESENTATION	FILL REPRESENTATION
EDGE REPRESENTATION	ASPECT SOURCE FLAGS
BEGIN FIGURE	END FIGURE
NEW REGION	

List of ESCAPE, GDP and GET ESCAPE functions required

No ESCAPE, GET ESCAPE or GDP functions are required for this Profile.

Resource requirements

Number of Points for Disjoint Polyline	(128)
Number of Predefined Line Bundles	(5)
Available Line Types	(solid, dash, dot, dash-dot, dash-dot-dot)
Available Scaled Line Widths	(continuous)
Number of Predefined Marker Bundles	(5)
Available Marker Types	(dot, plus, asterisk, circle, cross)
Available Scaled Marker Sizes	(continuous)
Number of Predefined Text Bundles	(5)
Available STRING Precision Text Fonts	(1) with:
Supported Horizontal Text Alignments	(NORMAL HORIZONTAL, LEFT, CENTRE, RIGHT)
Supported Vertical Text Alignments	(NORMAL VERTICAL, CAP, TOP, HALF, BOTTOM, BASE)
Available CHARACTER Precision Text Fonts	(1) with:
Supported Horizontal Text Alignments	(NORMAL HORIZONTAL, LEFT, CENTRE, RIGHT)
Supported Vertical Text Alignments	(NORMAL VERTICAL, CAP, TOP, HALF, BOTTOM, BASE)
Available STROKE Precision Text Fonts	(1) with:
Supported Horizontal Text Alignments	(NORMAL HORIZONTAL, LEFT, CENTRE, RIGHT)
Supported Vertical Text Alignments	(NORMAL VERTICAL, CAP, TOP, HALF, BOTTOM, BASE)
Number of Points for Polygon	(128)
Number of Predefined Fill Bundles	(5)
Available Interior Styles	(HOLLOW, SOLID, HATCH, EMPTY)
Available Hatch Styles	(horizontal, vertical, positive slope, negative slope, horizontal/vertical crosshatch, positive slope/negative slope crosshatch)
Available Edge Types	(solid, dash, dot, dash-dot, dash-dot-dot)
Available Scaled Edge Widths	(continuous)
Number of Predefined Edge Bundles	(5)

Definition of restrictions

None.

7 Classification and designation

7.1 Implementation conformance

ISO/IEC 9636 specifies the functional capability of the Computer Graphics Interface. Where an implementation claims to support a function defined in ISO/IEC 9636, the behaviour of the implementation shall match the function definition.

An implementation of a CGI interpreter that controls a physical device or a physical device that presents a CGI interpreter interface conforms to ISO/IEC 9636 if all of the following conditions are met:

a) All implemented CGI functions shall behave as specified by ISO/IEC 9636;

b) All implemented functions shall conform to at least one of the encodings or bindings specified in one of the related standards: ISO/IEC 9637 and ISO/IEC 9638;

c) One of the following conditions are met:

   1) A CGI interpreter that supports output shall satisfy at least one of the output Profiles defined in clause 6 of this part of ISO/IEC 9636;

   2) A CGI interpreter that supports input shall satisfy at least one of the input Profiles defined in clause 6 of this part of ISO/IEC 9636;

   3) A CGI interpreter that supports input and output shall satisfy at least one of the input and one of the output Profiles defined in clause 6 of this part of ISO/IEC 9636.

NOTE – A CGI interpreter may also support additional Profiles and/or functions specified in ISO/IEC 9636.

d) The relationship among implemented CGI functions and CGI states conforms to the specification in ISO/IEC 9636;

e) The list of functions supported shall not contain CGI functions that are meaningless in the absence of any function that is not implemented;

f) A mechanism shall be provided that enables a client to detect when extensions are being used (see 7.3). (This mechanism is not available to implementations that conform to the 1-WAY OUTPUT Profile.)

An implementation of a generator of a CGI data stream conforms to ISO/IEC 9636 if all of the following conditions are met:

a) All generated functions shall conform to at least one of the encodings specified in ISO/IEC 9637; if an implementation of a CGI generator presents a language binding interface to a CGI client then all implemented functions shall conform to at least one of the bindings specified in ISO/IEC 9638;

b) One of the following conditions are met:

   1) A CGI generator that supports output shall satisfy at least one of the output Profiles defined in clause 6 of this part of ISO/IEC 9636;

   2) A CGI generator that supports input shall satisfy at least one of the input Profiles defined in clause 6 of this part of ISO/IEC 9636;

   3) A CGI generator that supports input and output shall satisfy at least one of the input and one of the output Profiles defined in clause 6 of this part of ISO/IEC 9636.

NOTE – A CGI generator may also support additional Profiles and/or functions specified in ISO/IEC 9636.

c) The list of functions supported shall not contain CGI functions that are meaningless in the absence of any function that is not implemented;

d) A CGI generator shall fully support all implemented soliciting functions and be able to recognize the responses received from its downstream interface.

7.2 Client conformance

A client of the CGI that calls a CGI procedural interface or generates a CGI data stream, or a CGI data stream from an unknown source, conforms to ISO/IEC 9636 if it meets the following conditions:

a) The client or data stream shall use only one of the encodings or bindings specified in one of the related standards: ISO/IEC 9637 and ISO/IEC 9638;

b) The client or data stream shall use only those functions defined in ISO/IEC 9636;

c) To conform to a Profile in ISO/IEC 9636, the client shall use only those functions and related constraints contained in that Profile.

A generator client (see 4.2.1.2) of the CGI that itself presents a CGI interface conforms to ISO/IEC 9636 if it meets the conditions (a)-(c) above and also meets the following conditions:

a) Any function presented at the client's upstream interface shall be faithfully reproduced at the downstream interface;

b) Any response received at the downstream interface to a soliciting function that arrived at the upstream interface shall be faithfully returned over the client's upstream interface.

7.3 Extensions

Extensions are defined as those functions available and documented in a certain implementation, but not specified in ISO/IEC 9636.

An implementation which satisfies the constraints of 7.1 and provides additional non-standard functions shall satisfy the following criteria to be a conforming implementation:

a) Provide a mechanism to detect the use of extensions by implementing class 8 error detection, that is, the use of private functions and features;

b) Follow the rules and guidelines for ISO/IEC 9636 and its corresponding data encodings or language bindings;

c) Clearly specify the extensions and document them as extensions;

d) Ensure that extensions do not cause client-conforming programs to work improperly.

7.4 Inquiry

Inquiry functions are supplied by ISO/IEC 9636; for implementations of the CGI which only satisfy the 1-WAY OUTPUT Foundation Profile, no inquiry functions are required. Where inquiry is supported, only those functions that return information from implemented description tables and state lists are required.

Every implementation shall provide YES or NO responses to LOOKUP PROFILE SUPPORT for those Profiles specified in ISO/IEC 9636, except for implementations providing only 1-WAY OUTPUT Profiles. For any other Profile, registered or not, a response of UNRECOGNIZED is acceptable.

7.5 Parsing

All CGI implementations supporting a data stream encoding must be able to parse all function representations, regardless of whether the function is supported. All standard data stream encodings of the CGI shall be designed to permit recognition of the end of function representation. A function representation of an unsupported function shall be discarded without causing disruption to the CGI state. The CGI implementation shall in such circumstances generate a class 4 error.

Parsing

The data stream encodings shall be designed such that an implementation of a CGI interpreter is able to recognize that the function is a soliciting function, and is consequently able to return at least the appropriate response opcode and a response validity parameter of INVALID.

For CGI implementations that use a procedural language binding, it is required that the functions LOOKUP FUNCTION SUPPORT and LOOKUP PROFILE SUPPORT have entry points. For further recommendations, see annex C.

Annex A

(normative)

Function identifiers

A.1 Function identifiers

This annex specifies the assigned function identifiers for all standardized CGI functions.

Private functions shall be assigned negative integer identifiers.

A.1.1 CGI function identifier list

Function Identifier	**Function Name**
ISO/IEC 9636-2	
1	INITIALIZE
2	TERMINATE
3	EXECUTE DEFERRED ACTIONS
4	DEFERRAL MODE
5	PREPARE DRAWING SURFACE
6	END PAGE
7	VDC TYPE
8	VDC INTEGER PRECISION REQUIREMENT
9	VDC REAL PRECISION REQUIREMENTS
10	VDC EXTENT
11	DEVICE VIEWPORT
12	DEVICE VIEWPORT SPECIFICATION MODE
13	DEVICE VIEWPORT MAPPING
14	DRAWING SURFACE CLIP RECTANGLE
15	DRAWING SURFACE CLIP INDICATOR
16	DEQUEUE ERROR REPORTS
17	ERROR HANDLING CONTROL
18	INTEGER PRECISION REQUIREMENT
19	REAL PRECISION REQUIREMENTS
20	INDEX PRECISION REQUIREMENT
21	COLOUR PRECISION REQUIREMENT
22	COLOUR INDEX PRECISION REQUIREMENT
23	CLIENT SPECIFIED NAME PRECISION REQUIREMENT
24	MESSAGE
25	ESCAPE
26	GET ESCAPE
27	STATE LIST INQUIRY SOURCE
28	INQUIRE DEVICE IDENTIFICATION
29	INQUIRE DEVICE DESCRIPTION
30	LOOKUP FUNCTION SUPPORT
31	LOOKUP PROFILE SUPPORT
32	INQUIRE LIST OF PROFILE SUPPORT INDICATORS
33	INQUIRE SUPPORTED VDC TYPES
34	INQUIRE DEVICE CONTROL CAPABILITY
35	LOOKUP ESCAPE SUPPORT
36	LOOKUP GET ESCAPE SUPPORT
37	INQUIRE CONTROL STATE

Function Identifier	Function Name
38	INQUIRE CURRENT PRECISIONS REQUIREMENTS
39	INQUIRE VDC TO DEVICE MAPPING
40	INQUIRE ERROR HANDLING
41	INQUIRE MISCELLANEOUS CONTROL STATE
ISO/IEC 9636-3	
42	POLYLINE
43	DISJOINT POLYLINE
44	CIRCULAR ARC 3 POINT
45	CIRCULAR ARC CENTRE
46	CIRCULAR ARC CENTRE REVERSED
47	ELLIPTICAL ARC
48	CONNECTING EDGE
49	POLYMARKER
50	TEXT
51	RESTRICTED TEXT
52	APPEND TEXT
53	POLYGON
54	POLYGON SET
55	RECTANGLE
56	CIRCLE
57	CIRCULAR ARC 3 POINT CLOSE
58	CIRCULAR ARC CENTRE CLOSE
59	ELLIPSE
60	ELLIPTICAL ARC CLOSE
61	CELL ARRAY
62	GENERALIZED DRAWING PRIMITIVE (GDP)
63	LINE BUNDLE INDEX
64	LINE TYPE
65	LINE WIDTH
66	LINE COLOUR
67	LINE CLIPPING MODE
68	MARKER BUNDLE INDEX
69	MARKER TYPE
70	MARKER SIZE
71	MARKER COLOUR
72	MARKER CLIPPING MODE
73	TEXT BUNDLE INDEX
74	TEXT FONT INDEX
75	TEXT PRECISION
76	CHARACTER EXPANSION FACTOR
77	CHARACTER SPACING
78	TEXT COLOUR
79	CHARACTER HEIGHT
80	CHARACTER ORIENTATION
81	TEXT PATH
82	TEXT ALIGNMENT
83	CHARACTER SET INDEX
84	ALTERNATE CHARACTER SET INDEX
85	CHARACTER CODING ANNOUNCER
86	FILL BUNDLE INDEX
87	INTERIOR STYLE
88	FILL COLOUR
89	HATCH INDEX
90	PATTERN INDEX

Function identifiers

Function Identifier	Function Name
91	FILL REFERENCE POINT
92	PATTERN SIZE
93	EDGE BUNDLE INDEX
94	EDGE TYPE
95	EDGE WIDTH
96	EDGE COLOUR
97	EDGE CLIPPING MODE
98	EDGE VISIBILITY
99	CLIP INDICATOR
100	CLIP RECTANGLE
101	LINE WIDTH SPECIFICATION MODE
102	EDGE WIDTH SPECIFICATION MODE
103	MARKER SIZE SPECIFICATION MODE
104	COLOUR SELECTION MODE
105	COLOUR VALUE EXTENT
106	BACKGROUND COLOUR
107	AUXILIARY COLOUR
108	TRANSPARENCY
109	COLOUR TABLE
110	LINE REPRESENTATION
111	MARKER REPRESENTATION
112	TEXT REPRESENTATION
113	FILL REPRESENTATION
114	EDGE REPRESENTATION
115	DELETE BUNDLE REPRESENTATION
116	ASPECT SOURCE FLAGS
117	PATTERN TABLE
118	DELETE PATTERN
119	FONT LIST
120	CHARACTER SET LIST
121	SAVE PRIMITIVE ATTRIBUTES
122	RESTORE PRIMITIVE ATTRIBUTES
123	DELETE PRIMITIVE ATTRIBUTE SAVE SET
124	BEGIN FIGURE
125	END FIGURE
126	NEW REGION
127	GET TEXT EXTENT
128	INQUIRE PRIMITIVE SUPPORT LEVELS
129	LOOKUP GDP SUPPORT
130	INQUIRE GDP ATTRIBUTES
131	INQUIRE LINE CAPABILITY
132	INQUIRE LIST OF AVAILABLE LINE TYPES
133	INQUIRE LIST OF AVAILABLE SCALED LINE WIDTHS
134	INQUIRE MARKER CAPABILITY
135	INQUIRE LIST OF AVAILABLE MARKER TYPES
136	INQUIRE LIST OF AVAILABLE SCALED MARKER SIZES
137	INQUIRE TEXT CAPABILITY
138	INQUIRE LIST OF AVAILABLE CHARACTER SETS
139	INQUIRE LIST OF AVAILABLE TEXT FONTS
140	INQUIRE FONT CAPABILITIES
141	INQUIRE LIST OF AVAILABLE CHARACTER EXPANSION FACTORS
142	INQUIRE LIST OF AVAILABLE CHARACTER SPACINGS
143	INQUIRE LIST OF AVAILABLE CHARACTER HEIGHTS
144	INQUIRE LIST OF AVAILABLE CHARACTER ORIENTATIONS
145	INQUIRE FILL CAPABILITY

Function Identifier	Function Name
146	INQUIRE LIST OF AVAILABLE HATCH STYLES
147	INQUIRE EDGE CAPABILITY
148	INQUIRE LIST OF AVAILABLE EDGE TYPES
149	INQUIRE LIST OF AVAILABLE SCALED EDGE WIDTHS
150	INQUIRE COLOUR CAPABILITY
151	INQUIRE CIE CHARACTERISTICS
152	INQUIRE MAXIMUM NUMBER OF SIMULTANEOUSLY SAVED ATTRIBUTE SETS
153	INQUIRE ARRAY OF SUPPORTED CHARACTER CODING ANNOUNCERS
154	INQUIRE LINE ATTRIBUTES
155	INQUIRE LIST OF LINE BUNDLE INDICES
156	INQUIRE LINE REPRESENTATION
157	INQUIRE MARKER ATTRIBUTES
158	INQUIRE LIST OF MARKER BUNDLE INDICES
159	INQUIRE MARKER REPRESENTATION
160	INQUIRE TEXT ATTRIBUTES
161	INQUIRE LIST OF TEXT BUNDLE INDICES
162	INQUIRE TEXT REPRESENTATION
163	INQUIRE FILL ATTRIBUTES
164	INQUIRE PATTERN DIMENSIONS
165	INQUIRE PATTERN
166	INQUIRE LIST OF PATTERN INDICES
167	INQUIRE LIST OF FILL BUNDLE INDICES
168	INQUIRE FILL REPRESENTATION
169	INQUIRE EDGE ATTRIBUTES
170	INQUIRE LIST OF EDGE BUNDLE INDICES
171	INQUIRE EDGE REPRESENTATION
172	INQUIRE OUTPUT STATE
173	INQUIRE OBJECT CLIPPING
174	INQUIRE LIST OF ATTRIBUTE SET NAMES IN USE
175	INQUIRE COLOUR STATE
176	INQUIRE LIST OF COLOUR TABLE ENTRIES
177	INQUIRE FONT LIST
178	INQUIRE CHARACTER SET LIST
179	LOOKUP ASPECT SOURCE FLAGS
ISO/IEC 9636-4	
180	GET NEW SEGMENT IDENTIFIER
181	CREATE SEGMENT
182	REOPEN SEGMENT
183	CLOSE SEGMENT
184	COPY SEGMENT
185	DELETE SEGMENT
186	DELETE ALL SEGMENTS
187	RENAME SEGMENT
188	DRAW ALL SEGMENTS
189	IMPLICIT SEGMENT REGENERATION MODE
190	RESET REGENERATION PENDING
191	PICK IDENTIFIER
192	SEGMENT VISIBILITY
193	SEGMENT TRANSFORMATION
194	SEGMENT HIGHLIGHTING
195	SEGMENT DISPLAY PRIORITY
196	SEGMENT DETECTABILITY
197	SEGMENT PICK PRIORITY
198	SIMULATE PICK