INTERNATIONAL STANDARD

ISO/IEC 7942-1

Second edition
1994-11-01

Information technology — Computer graphics and image processing — Graphical Kernel System (GKS) —

Part 1:
Functional description

Technologies de l'information — Infographie et traitement d'image — Système graphique GKS —

Partie 1: Description fonctionnelle

Reference number
ISO/IEC 7942-1:1994(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 JTC1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as International Standard requires approval by at least 75% of the national bodies casting a vote.

International Standard ISO/IEC 7942-1 was prepared by Joint Technical Committee ISO/IEC JTC1, *Information technology, Sub-Committee 24, Computer graphics and image processing.*

This second edition cancels and replaces the first edition (ISO 7942:1985), which has been technically revised.

ISO/IEC 7942 consists of the following parts, under the general title *Information technology – Computer graphics and image processing – Graphical Kernel System (GKS):*

Part 1: Functional description

Part 2: NDC metafile

Part 3: Audit trail

Part 4: Picture part archive

Annexes A and B form an integral part of this part of ISO/IEC 7942. Annexes C, D, E, F, G, and H are for information only.

Introduction

The Graphical Kernel System (GKS) provides a set of functions for computer graphics programming that can be used by a range of applications. The main motivations for standardization are to improve portability of programs and to define a basic methodology. Portability is achieved by providing all the capabilities necessary in a device independent way. The following principles are used in specifying GKS:

a) Design: the three goals are consistency of approach, compatibility with related standards and orthogonal functionality where possible.

b) Functionality: the goals are completeness with the minimal set of functions. Organization of functions should be such as to achieve compact programs. Richness should be provided by utilities and toolkits on top of GKS rather than non-standard extensions to GKS.

c) Clarity: the underlying concepts should be easily understood, especially by the application programmer. To achieve this, GKS is defined using a small set of functions with precise specification of the data structures that define the state of GKS and the effect that functions have on this state.

d) Error handling: all errors caused by incorrect function invocations or internal failures are logged with the application having control over the action taken.

e) Separation of device dependent functionality: in GKS there is a clear separation between the functionality that is device dependent and device independent. GKS has the concept of a workstation which deals with all device dependent functions.

f) Implementation: GKS should be realizable in a wide range of host languages. The support GKS requires from the operating system should not be excessive. GKS should be efficient to implement on commonly available hardware and it should be easy to produce a robust product.

This International Standard (known informally as "GKS-94") has a strong relationship to the principles and functionality of ISO 7942:1985 (known informally as "GKS-85"). However, it has been updated to provide additional functionality, cleaner concepts and an International Standard for modern hardware. Minor changes made include some additional primitives, some minor extensions to input, a clearer definition of the NDC picture and simplification of deferral.

A nameset attribute has been added and this is used as a selection criterion for a range of functions. As a result, the equivalent of the ISO 7942:1985 segment (picture part) no longer needs attributes separate from the primitive attributes. The segment facilities of ISO 7942:1985 are provided in terms of the new picture part store.

The ISO 7942:1985 metafile is replaced by an audit trail and an NDC metafile based on the Computer Graphics Metafile (ISO/IEC 8632).

Information technology – Computer graphics and image processing - Graphical Kernel System (GKS) – Part 1: Functional description

1 Scope

This part of ISO/IEC 7942 specifies a set of functions for computer graphics programming, the Graphical Kernel System (GKS). It provides functions for two dimensional graphical output, the storage and dynamic modification of pictures, and operator input. GKS functions and datatypes are specified independently of programming languages.

GKS establishes a system for device independent graphics programming by separating picture composition and interaction from the realization of the pictures on a specific output device and the input devices used by the operator.

This International Standard is applicable to a wide range of applications that produce two dimensional pictures on vector or raster graphical devices in monochrome or colour. Operator interaction is allowed with these pictures.

2 Normative references

The following standards contain provisions which, through reference in this text, constitute provisions of this part of ISO/IEC 7942. 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 7942 are encouraged to investigate the possibilities of applying the most recent editions of the standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards.

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

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

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

ISO 6093:1985, *Information processing - Representation of numerical values in character strings for information interchange.*

ISO/IEC 8632:1992, *Information technology - Computer graphics - Metafile for transfer and storage of picture description information*

- *Part 1 : Functional specification.*
- *Part 2 : Character encoding.*
- *Part 3 : Binary encoding.*
- *Part 4 : Clear text encoding.*

ISO/IEC 9541:1991, *Information technology - Font information interchange.*

3 Definitions

For the purposes of this part of ISO/IEC 7942, the following definitions apply.

3.1 picture: A spatially structured sequence of output primitives destined for storage or display on a workstation. The following pictures exist: NDC picture, logical picture, realized picture.

3.1.1 NDC picture: The picture in which graphical output is composed by the application program and with which the operator interacts using graphical input devices.

3.1.2 logical picture: The picture constructed from the NDC picture for a particular workstation in which logical attributes are bound to output primitives.

3.1.3 realized picture: The picture constructed from the logical picture for a particular workstation by binding the precise definition of colours specified either directly or indirectly. The realized picture is displayed on the workstation's display space.

3.2 workstation: A display space and associated input peripherals.

3.2.1 display surface: The physical area on a display device onto which display spaces are mapped.

3.2.2 display space: The area available for displaying images on a particular workstation.

3.3 coordinate system: The metric used to specify graphical output and positional input. The coordinate systems are world coordinates, normalized device coordinates, logical device coordinates and device coordinates. All coordinate systems are cartesian.

3.3.1 world coordinates (WC): The coordinates used by the application program to define output primitives and positional input.

3.3.2 normalized device coordinates (NDC): The coordinates used to define the NDC picture and picture parts.

3.3.3 logical device coordinates (LDC): The coordinates used to define the logical picture.

3.3.4 device coordinates (DC): The coordinates used to define realized pictures on a workstation.

3.4 transformation: A process for transforming one coordinate system to another.

3.4.1 normalization transformation: A window-to-viewport transformation that maps positions in WC to NDC.

3.4.2 normalization transformation number: An identifier of a particular normalization transformation.

3.4.3 view orientation: A transformation that transforms the NDC coordinate system to one more appropriate for the object to be viewed.

3.4.4 view mapping: A transformation applied after view orientation to map to LDC coordinates.

3.4.5 workstation transformation: A workstation window-to-viewport mapping that maps positions in LDC to device coordinates, preserving aspect ratio.

3.5 scissoring: Restricting the appearance of an output primitive on a display by either clipping or shielding.

3.5.1 view scissoring: Scissoring applied to output primitives in a view after view orientation and view mapping. View scissoring may be postponed until after the workstation transformation.

3.6 picture part store: A collection of picture parts.

3.6.1 picture part: A sequence of output primitives with associated attributes.

3.7 state list: A data structure whose entries specify the current values of variables relating to GKS as a whole (GKS state list) or to a specific workstation (workstation state list).

3.8 description table: A data structure whose entries specify the capabilities of an implementation (GKS description table) or a type of workstation (generic workstation description table) or an instance of a type of workstation (workstation description table).

3.9 backdrop: Graphical output that appears on the workstation but is not part of the NDC picture, logical picture, or realized picture. The backdrop arrives at the display device on a separate channel from the realized picture. It is device dependent how the two appear relative to each other.

3.9.1 router: A switch which controls the destination to which output primitives are dispatched.

3.10 output primitive: A basic graphical element used to construct the NDC picture and picture parts.

3.11 audit trail: The external entity for representing the sequential flow of information across the interface between the application and GKS.

3.12 output primitive classes: A classification associated with output primitives. A set of logical attributes apply to all output primitive types in a class. The output primitive classes are curve, marker, area, character, image, design. The output primitive, generalized drawing primitive, does not belong to any class.

3.12.1 curve: A class of output primitive which generates a set of curves. The output primitives in the class are set of polyline, set of nurb and set of conic section.

3.12.1.1 set of polyline: A type of output primitive consisting of a set of polylines each of which consists of a connected sequence of straight lines between specified points.

3.12.1.2 set of nurb: A type of output primitive consisting of a set of curves each of which is defined by a nurb.

3.12.1.2.1 nurb: A curve defined by a non-uniform B-spline. Both rational and non-rational versions are allowed.

3.12.1.3 set of conic section: A type of output primitive consisting of a set of curves each of which is defined by a conic section.

3.12.2 marker: A class of output primitive which generates a set of symbols. The only output primitive in the class is polymarker.

3.12.2.1 polymarker: A type of output primitive consisting of a set of positions at which symbols are placed.

3.12.3 area: A class of output primitive which generates a set of areas with edges. The output primitives in the class are set of fill area, set of closed nurb, set of elliptic sector, set of elliptic segment and set of elliptic disc.

3.12.3.1 set of fill area: A type of output primitive consisting of a set of areas each of which is defined by a closed sequence of connected points.

3.12.3.2 set of closed nurb: A type of output primitive consisting of a set of areas each of which is defined by a closed nurb.

3.12.3.2.1 closed nurb: A nurb which has the first point connected to the end point to produce a closed curve if the two points are not coincident.

3.12.3.3 set of elliptic sector: A type of output primitive consisting of a set of areas each of which is defined by an elliptic arc. Each elliptic arc area is closed by lines from the ends of the arc to the centre of the ellipse.

3.12.3.4 set of elliptic segment: A type of output primitive consisting of a set of areas each of which is defined by an elliptic arc. Each elliptic arc area is closed by joining the ends of the arc.

3.12.3.5 set of elliptic disc: A type of output primitive consisting of a set of areas each of which is defined by a complete ellipse.

3.12.3.6 edge: The boundaries of closed regions defined in the area class of output primitives.

3.12.4 character: A class of output primitive which generates a sequence of glyphs. The only output primitive in the class is text.

3.12.4.1 text: A type of output primitive consisting of a sequence of glyphs defined relative to a position.

Definitions

3.12.4.1.1 glyph: A recognizable abstract graphic symbol which is independent of any specific design (ISO/IEC 9541).

3.12.4.1.2 glyph body: A rectangle used by a font designer to define a glyph shape (see figure 1). All glyph bodies in a font have the same height.

3.12.4.1.3 glyph shape: The set of information in a glyph representation used for displaying the dimensions and positioning of the glyph shape (ISO/IEC 9541).

3.12.4.1.4 glyph representation: The glyph shape and glyph metrics associated with a specified glyph in a font resource (ISO/IEC 9541).

3.12.4.1.5 topline: A horizontal line at the top of a glyph body (see figure 1) which is just above the upper limit of all glyph shapes in a font. Ascenders and accents are below the topline.

3.12.4.1.6 capline: A horizontal line within a glyph body (see figure 1) which, for many glyph definitions, has the appearance of being the upper limit of the glyph shape. An ascender may pass above this line and in some languages an additional mark (for example an accent) over the glyph may be defined above this line. All caplines in a font are in the same position in the glyph bodies.

3.12.4.1.7 baseline: A horizontal line within a glyph body (see figure 1) which, for many glyph definitions, has the appearance of being a lower limit of the glyph shape. A descender passes below this line. All baselines in a font are in the same position in the glyph bodies.

3.12.4.1.8 halfline: A horizontal line between the capline and the baseline within the glyph body (see figure 1), about which a horizontal string of glyphs in a font would appear centrally placed in the vertical direction. All halflines in a font are in the same position in the glyph bodies.

3.12.4.1.9 bottomline: A horizontal line at the bottom of the glyph body (see figure 1) which is just below all descenders in a font. All bottomlines in a font are in the same position in the glyph bodies.

3.12.4.1.10 centreline: A vertical line bisecting the glyph body (see figure 1).

3.12.4.1.11 leftline: A vertical line at the left of a glyph body (see figure 1) which is to the left of all glyph shapes in a font.

3.12.4.1.12 rightline: A vertical line at the right of a glyph body (see figure 1) which is to the right of all glyph shapes in a font.

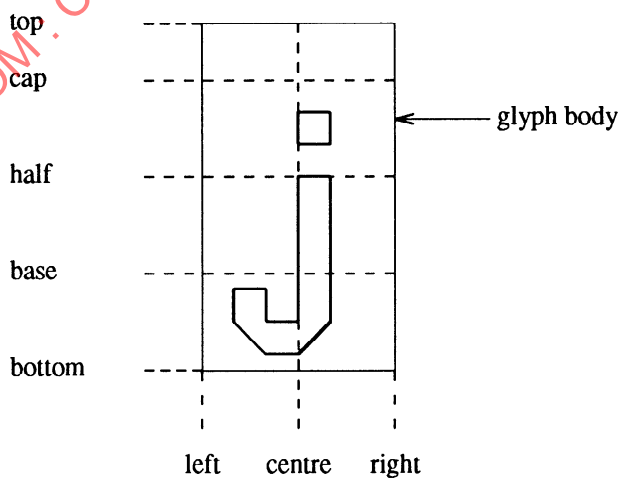

Figure 1 - Glyph coordinate system

3.12.4.1.13 font: A collection of glyph images having the same basic design (ISO/IEC 9541).

3.12.4.1.14 font resource: A collection of glyph representations together with description and font metric information which are relevant to the collection of glyph representations as a whole

(ISO/IEC 9541).

3.12.4.1.15 posture: The extent to which the shape of a glyph or set of glyphs appear to incline, including any consequent design or form change (ISO/IEC 9541).

3.12.4.1.16 escapement Movement of the current position on the presentation surface after a glyph representation is imaged (ISO/IEC 9541).

3.12.5 image: A class of output primitive which generates images. The only output primitive in the class is cell array.

3.12.5.1 cell array: A type of output primitive defined by a rectangular grid of equal size rectangular cells, each having a single colour.

3.12.6 design: A class of output primitive which generates graphical output by extruding a tiling through a stencil. Stencils are stored in the stencil store. Tilings are stored in the tiling store.

3.12.6.1 stencil: A set of shapes. Stencils may be constructed from area boundaries (which may be defined by closed paths), closed contours and other stencils.

3.12.6.1.1 closed path: A sequence of paths where the end of one path is the start of the next and where the last point is the same as the first.

3.12.6.1.2 path: A sequence of points used to specify the locus between two end points.

3.12.6.1.3 contour: A stencil which is defined as the outline surrounding a path.

3.12.6.2 tiling: A sequence of tiling components.

3.12.6.2.1 tiling component: A replicated tile.

3.12.6.2.2 tile: A pattern defined as a curve, area or design.

3.12.7 generalized drawing primitive (GDP): A type of output primitive used to address special output requirements.

3.13 attribute: A particular property that applies to an output primitive.

3.13.1 identification attributes: A class of attribute used to name output primitives.

3.13.1.1 nameset: An identification attribute (in the form of a set of names) of an output primitive.

3.13.1.2 pick identifier: An identification attribute of an output primitive.

3.13.1.3 selection criterion: A rule for choosing elements from a sequence of output primitives. Selection criteria are expressed as operations on namesets.

3.13.2 NDC attributes: A class of attribute bound to primitives when they are created which completes the geometric definitions of primitives.

3.13.2.1 local transformation: An NDC attribute which defines a transformation in NDC space which is applied to the geometry of a primitive to position individual output primitives.

3.13.2.2 global transformation: An NDC attribute which defines a transformation in NDC space which is applied to the geometry of a primitive after the local transformation. Global transformations are used to apply a transformation to a complete object.

3.13.2.3 NDC scissor: A set of scissors applied to the NDC picture and backdrop after the local and global transformations. NDC scissoring may be postponed until after the workstation transformation.

3.13.3 source attributes: A class of attribute bound to primitives when they are created which are used to determine the values of the logical attributes which will be applied to primitives at the logical level. Bundle indices and attribute source flags are source attributes.

3.13.3.1 bundle index: An index into a particular bundle table.

3.13.3.2 attribute source flag (ASF): A flag for each logical attribute, which determines whether the value of a logical attribute of a primitive at the logical level will be taken from the logical attribute bound to the primitive at the NDC level, or from the bundle table using the bundle index bound at the NDC level. Attribute source flags for each of the logical attributes of a primitive are bound to the

primitive at the NDC level.

3.13.3.3 bundle table: A workstation dependent table with entries accessed by a bundle index. Each table entry defines the values of a set of logical attributes to be associated with a class of output primitive. Whether these values are bound to particular output primitives at the logical level depends on the settings of the corresponding attribute source flags.

3.13.4 logical attributes: A class of attribute which defines the appearance of an output primitive in the logical picture. Logical attributes may have different values for different workstations.

3.13.4.1 colour specifier: A colour type and colour value. Colour may be specified either directly or indirectly. If the colour type is indirect, the colour value is a colour index, otherwise the colour type defines a colour model and the colour value defines the value of the colour in that model. The realized picture defines the precise colour of primitives by substituting the appropriate definitions for indirectly specified colours.

3.13.4.2 colour array specifier: A colour type and colour value. If the colour type is indirect, the colour value is an array of colour indices. Otherwise the colour type specifies a colour model and the colour value is an array of values defining the colours in that model.

3.13.4.3 colour index: An index into a particular colour table.

3.13.4.4 colour table: A workstation dependent table. Each table entry defines, by a colour model name and associated values, the colour corresponding to a particular value of the colour index.

3.14 logical input device: An abstraction of one or more physical devices that delivers logical input values to the program. Specific logical input devices may be of class locator, stroke, valuator, choice, pick, string and composite.

3.14.1 logical input value: A measure value delivered by a logical input device.

3.14.1.1 measure: A value which is determined by one or more physical input devices and a mapping from the values delivered by the physical devices.

3.14.1.2 trigger: A condition which determines when a logical input value is delivered by a logical input device.

3.14.2 locator: A class of logical input device providing a position in world coordinates and a normalization transformation number.

3.14.3 stroke: A class of logical input device providing a sequence of positions in world coordinates and a normalization transformation number.

3.14.4 valuator: A class of logical input device providing a real number.

3.14.5 choice: A class of logical input device providing a positive integer.

3.14.6 pick: A class of logical input device providing the identification attributes of an output primitive.

3.14.7 string: A class of logical input device providing a character string.

3.14.8 composite: A class of logical input device built up from previously defined devices.

3.14.9 prompt: An indication to the operator that a specific logical input device is available.

3.14.10 echo: The immediate notification to the operator of the current value of an input device.

The following alphabetical list gives the subclause of each GKS definition.

Definitions

4 Conformance

4.1 Specification

The set of functions known as GKS shall be as described in clauses 5, 6, 7, 8, 9, 10, 11, 12, 13, and 14 and annexes A and B. In an implementation all graphical capabilities of a display space that can be addressed by GKS functions shall be used only via GKS.

4.2 Registration[1)]

For certain parameters of the functions, GKS defines value ranges as being reserved for registration. The meanings of these values will be defined using the established procedures described in ISO/IEC TR 9973: 1988. These procedures do not apply to values and value ranges defined as being workstation or implementation dependent; these values and value ranges are not standardized.

1) For the purpose of this part of ISO/IEC 7942 and according to the rules for the designation and operation of registration authorities in the ISO/IEC Directives, the ISO and IEC Councils have designated the following as the registration authority: National Institute of Standards and Technology (National Computer Systems Laboratory), The Registration Authority for Graphical Items, A-266 Technology Building, Gaithersurg, MD 20899.

5 Concepts

5.1 Pictures

A central concept is the *picture* which is a spatially structured sequence of output primitives destined for storage or display on a workstation. The *NDC picture* (Normalized Device Coordinate picture) is where graphical output is composed and with which the operator interacts using graphical input devices (see figure 2) on one or more workstations. Alternatively, graphical output can be dispatched directly to a backdrop in which case it cannot be interacted with.

For each workstation, a number of views of the NDC picture can be selected. Each view has all the logical attributes bound to create the *logical picture* as the set of views. Finally, the precise colour definitions are bound to create the *realized picture* for display on the workstation's display space. Changes to the NDC picture will be reflected in changes to the logical and realized pictures.

The operator interacts with the set of views in the realized picture. Where these overlap, it is possible to define which view takes priority in terms of operator interactions.

5.2 Output primitive classes and attributes

Output primitive classes are abstractions of basic actions a device can perform such as drawing curves, rendering areas and writing character strings. There may be several different types of output primitive in each class. Associated with output primitives are *attributes* which define additional properties of the primitive. For example, the type of curve to be output is specified by a primitive attribute.

There are four classes of primitive attributes:

a) identification

b) NDC

c) source

d) logical

Identification attributes can be used to partition output primitives into sets for a variety of purposes. The main identification attribute is the *nameset*.

NDC attributes complete the geometric definition of the output primitive in the NDC coordinate space. All output primitives have a *local transformation,* a *global transformation* and an *NDC scissor* as NDC attributes.

Source attributes determine the source of values for the attributes of the primitives in the logical picture.

Logical attributes define the appearance of the output primitives in the logical picture to be displayed on the workstation.

When an output primitive is created, identification, NDC, logical and source attributes are bound to it.

Attributes of output primitives in the NDC picture may be changed.

5.3 Workstations

GKS defines a *workstation* as an abstract *display space* on a device's *display surface* and associated input peripherals. Multiple workstations may be in operation together. Workstations may have no associated input peripherals or no display space.

A *selection criterion* based on the namesets of the output primitives in the NDC picture and backdrop defines what subset of the NDC picture and backdrop is displayed on a workstation. A second set of selection criteria decide how many views of each output primitive selected are displayed on the workstation.

For each workstation, the NDC picture is transformed into a *logical picture* on the workstation partitioned into views. The logical picture is defined in *logical device coordinates* (LDC). Output primitives in the logical

picture have all logical attributes and relevant NDC attributes bound to them.

Logical attributes are bound to output primitives in the NDC picture. The values bound to the equivalent output primitive in the logical picture depend on the source attributes. They are either the values of the logical attributes bound to the primitives in the NDC picture or the values extracted from workstation bundle tables. The attribute related to colour is the *colour specifier*.

The logical picture is transformed into the *realized picture* which is realized on the workstation display spacc. The realized picture is obtained by binding the required colour information, for indirectly specified colours (pointed at by the colour indices), to the output primitives in the logical picture. Figure 3 shows the NDC picture and picture part store. Figure 4 shows the logical and the realized pictures.

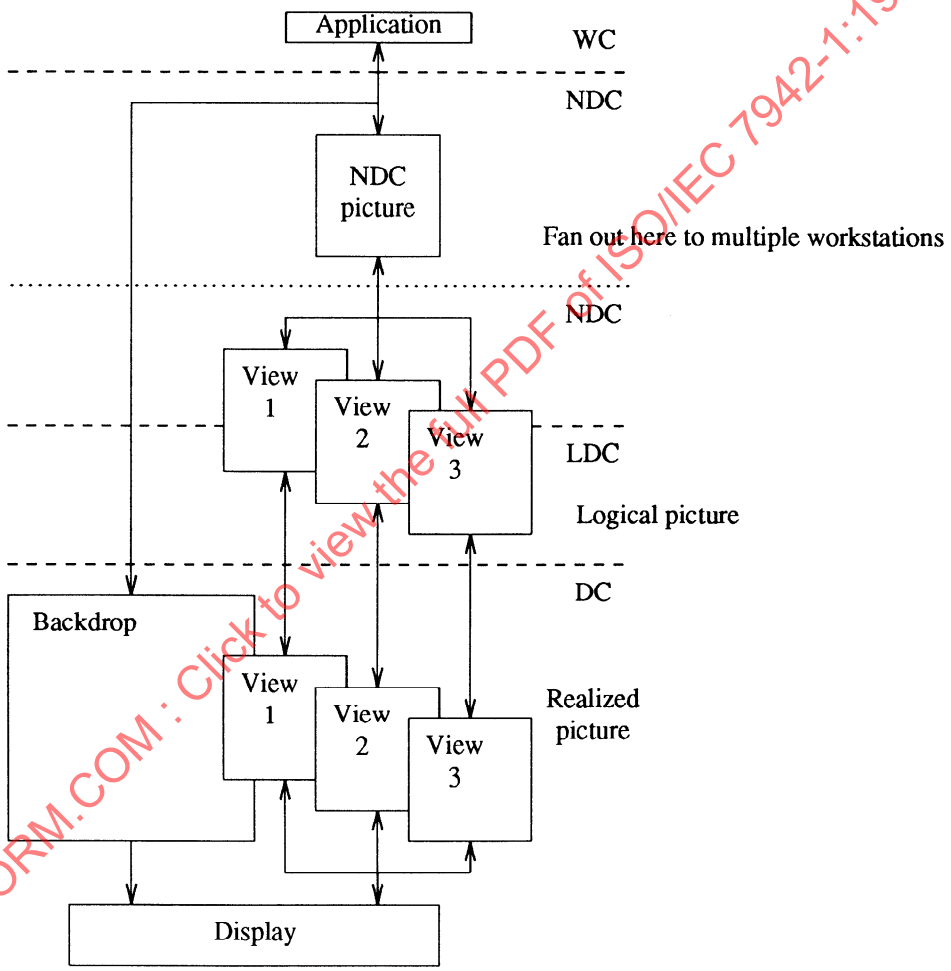

Figure 2 - GKS overview

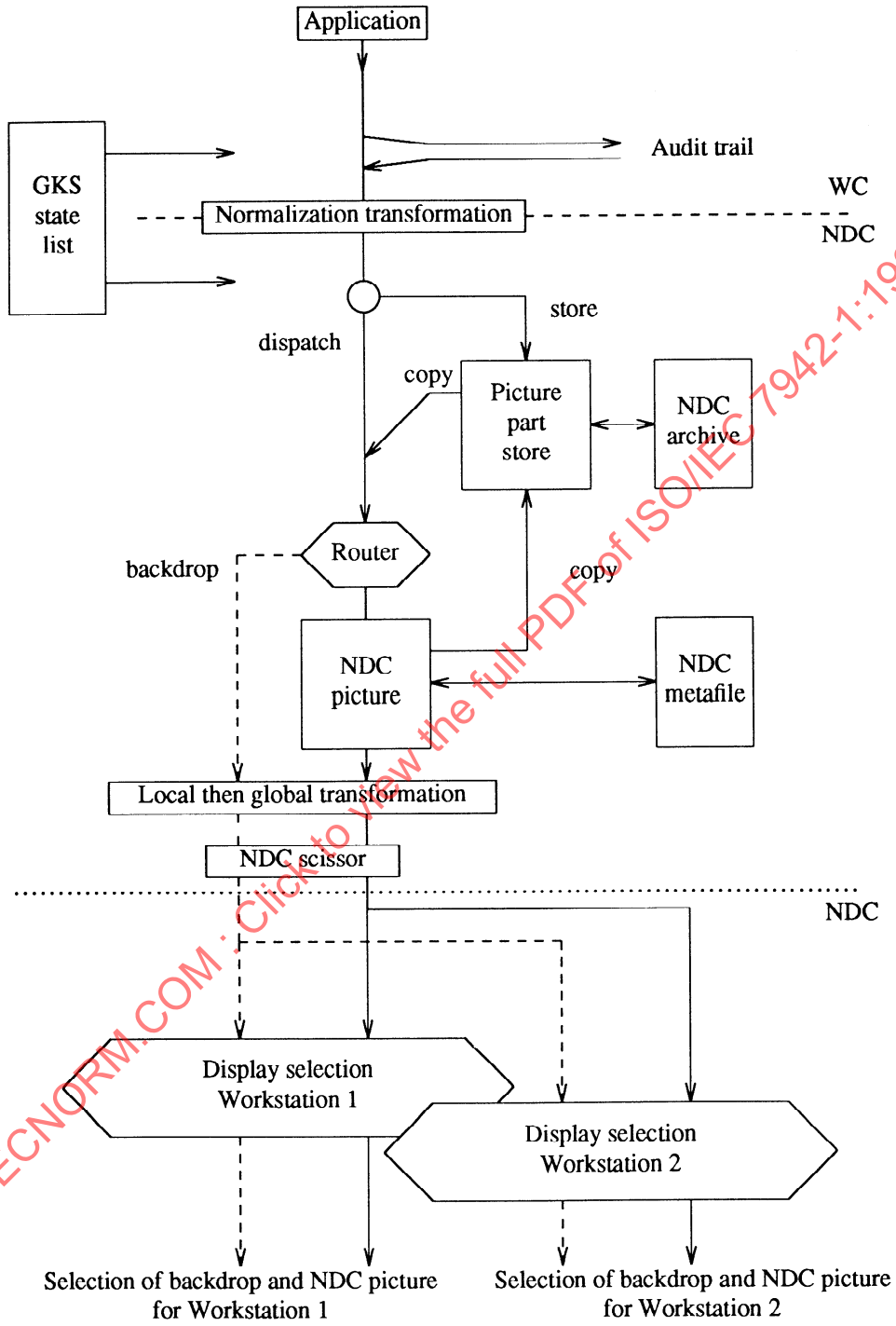

Figure 3 - GKS NDC picture

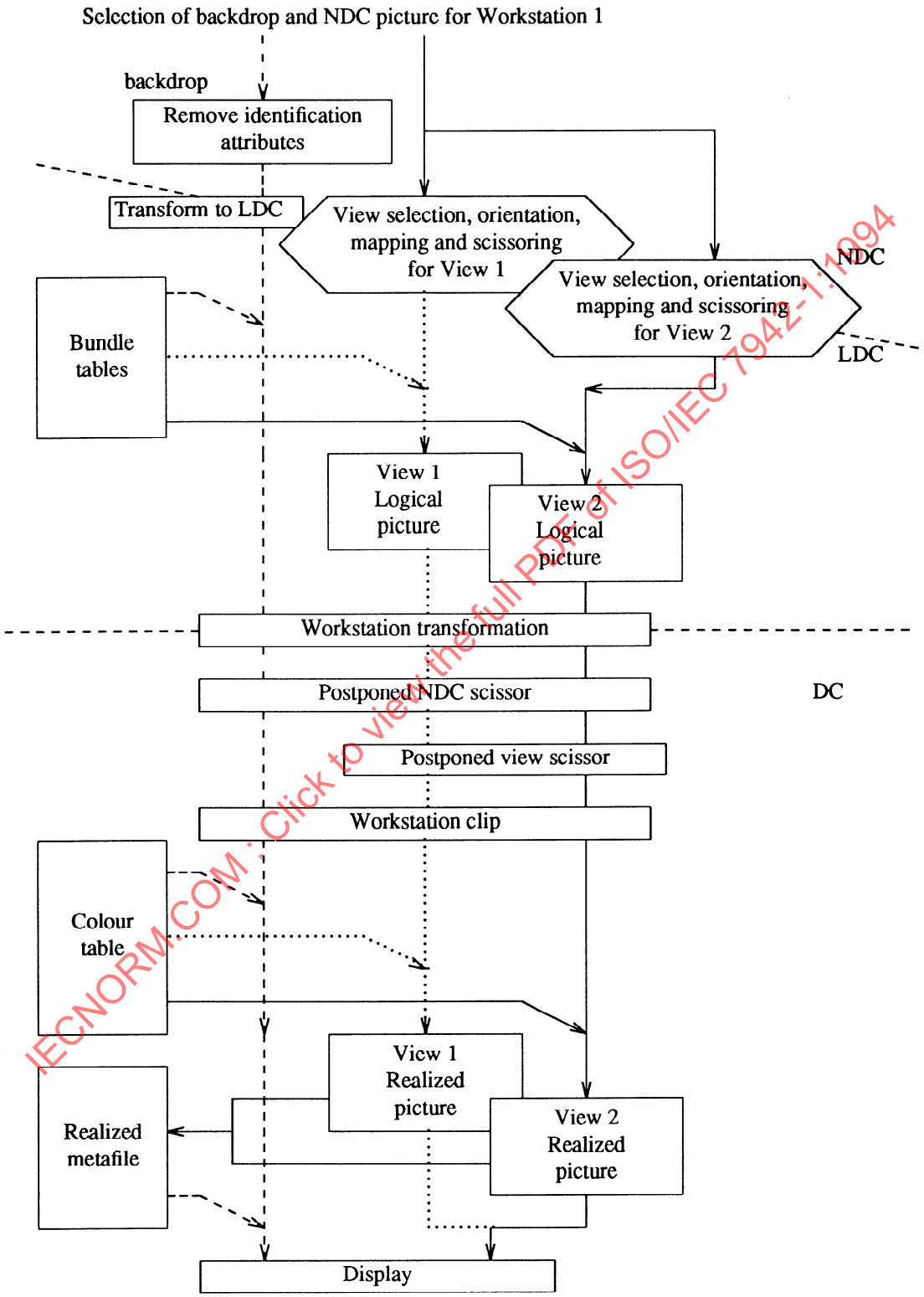

Figure 4 - GKS logical and realized pictures

5.4 Coordinate systems and transformations

Output primitives are defined by the application in a *world coordinate* system. The world coordinates are specified by the application. More than one world coordinate system may be specified.

The application specifies the transformation from world coordinates to *normalized device coordinates* (NDC). NDC is a workstation independent coordinate system.

The application specifies the orientation and mapping from NDC to LDC for each view on a workstation.

The display space of a workstation has a *device coordinate* system (DC) associated with it. The logical picture is defined in LDC and the realized picture in DC. The application specifies the mapping from LDC to DC for each workstation.

An output primitive and its NDC attributes are mapped from WC to NDC by a *normalization transformation*. The resulting geometry of an output primitive in NDC space is transformed by the local transformation and then by the global transformation. The output primitive may appear in several views on a workstation with different orientation and mapping from NDC to LDC. The workstation transformation performs the mapping from LDC to DC.

5.5 Logical input devices

A *logical input device* is an abstraction of one or more physical devices. An application can define how it receives *logical input values* from logical input devices.

Logical input devices are divided into classes dependent on the datatype of the logical input value. All logical input devices of one class deliver logical input values of the same datatype.

Three operating modes are defined for every logical input device. The operating modes specify whether the operator or the application has the initiative in controlling input.

5.6 Picture part store

Sequences of output primitives with associated attributes may be defined as a *picture part* and retained in *picture part store*. Positions associated with output primitives and attributes are stored in NDC coordinates. The NDC picture and the backdrop may be augmented by adding the sequence of output primitives in a picture part. The global and local transformation attributes of output primitives in a picture part may be changed as the picture part is added to the NDC picture, thus enabling several instances of a picture part to appear in the NDC picture. The namesets of output primitives in a picture part may be changed as the picture part is added to the NDC picture. Picture parts may also be created by selecting output primitives from the NDC picture.

5.7 State lists

State lists precisely describe the current state of an application in its use of graphics. Two types of state list exist. The *GKS state list* describes the current state of the NDC environment as shown in figure 3 and the input queue. The *workstation state list* describes the current state of a workstation. Several workstation state lists may be in use at the same time.

5.8 Description tables

The details of a particular GKS implementation are defined in a set of description tables. The *GKS description table* contains information about the specific implementation of GKS. A *generic workstation description table* defines the characteristics of a type of workstation and a *specific workstation description table* defines the characteristics of an instance of a generic workstation type.

5.9 Metafiles and archives

The contents of the NDC picture and realized picture may be stored and retrieved from metafiles.

The picture part store may be archived.

5.10 Routing to the backdrop

A *backdrop* is defined. Output primitives may be *routed* to the backdrop or the NDC picture. Output primitives in the backdrop appear on the workstation but are not part of the NDC, logical, or realized pictures. Output primitives in the backdrop are rendered using the relevant NDC and logical attributes. On creation, an output primitive is routed to the backdrop and selected for display on workstations. Once the selection has been made, the identification attributes for primitives in the backdrop are removed so that it is not possible for the operator to interact with information displayed in the backdrop. Attributes are bound to the primitives before they are displayed on the workstations. Primitives in the backdrop are transformed from NDC to LDC by an identity transformation.

The realized picture and backdrop are conceptually dispatched to the display of the workstation on separate channels. Their relative positioning is display dependent. Anything that destroys the integrity of the backdrop will cause it to be deleted from the display (see 7.10).

5.11 Audit trail

Audits of the sequence of GKS functions invoked by an application may be captured selectively in *audit trails*. User records may also be added. Functions are provided to playback audits for inquiry and execution of the output functions. An audit can be used to store the GKS functions that define a backdrop for later playback.

5.12 Segments

Segment functionality is retained in this part of ISO/IEC 7942 for compatibility with ISO 7942:1985 (see 10.1). It is achieved by naming, picture parts and selection criteria which give a more powerful means of selecting data for display.

An open segment is defined as a collection of output primitives in the NDC picture identified by a mapping of the segment name. On closure, the output primitives that make up the segment are stored as a picture part in picture part store with a picture part name that is a mapping of the segment name. Segment attributes are specified by selection criteria and primitive attributes.

5.13 Compatibility with ISO 7942:1985

Each GKS implementation shall ensure that it is possible to provide the functionality of ISO 7942:1985. Annex G illustrates one way of achieving it.

A GKS implementation has to support a special workstation type called WISS for compatibility with ISO 7942:1985. A WISS workstation is handled as in ISO 7942:1985 as far as state list entries are concerned, and for segment operations.

It is anticipated that this compatibility requirement will not be required for the next edition of GKS.

5.14 Summary

The main concepts are all contained in figures 3 and 4. Tables specific to single primitive classes have not been shown. Figure 3 shows the application accessing GKS (through a language binding). The commands generated by the application can be stored in an audit trail and replayed later. The commands can change values in the GKS state list (define text height, select normalization transformation etc) and these are either stored in world or NDC coordinates depending on the specific item. This is why the GKS state list straddles the normalization transformation. The geometric information defining the output primitive is transformed to NDC by the selected normalization transformation and attribute values are bound to the primitive from the GKS state list. Output primitives can either be dispatched for output or stored as a part of a picture part in picture part store. For long term storage (between sessions), it is possible to archive some or all of the picture part store.

Output consists of output primitives either generated by the application or copied from picture part store. The router directs these output primitives to the backdrop or to the NDC picture. The backdrop is provided so that large background pictures can be output without the need for local storage. In general, output primitives will

be added to the NDC picture. These will have attributes associated with them defining both a local and a global transformation and the NDC scissor to be applied. The output primitive in the NDC picture selected for display will have its geometry transformed first by the local transformation and then by the global transformation before being scissored by the NDC scissor. The local transformation can be used in the creation of objects while the global transformation is usually used for the positioning of objects made up of a number of output primitives. The NDC scissor may be postponed until after the workstation transformation if scissoring of the output primitives' shape rather than locus is required.

The NDC picture can be captured in an NDC metafile.

A selection criterion on the identification attributes of the output primitives in the NDC picture or backdrop defines which output primitives will be displayed on each workstation.

Figure 4 shows what happens to the set of output primitives selected for display on a specific workstation. Each output primitive selected from the NDC picture can appear in several views on a workstation. A second selection criterion selects the set of output primitives to appear in each view. For each view, the output primitives can be reorientated and mapped to a different position in LDC space. The values of logical attributes (from the bundle tables) may replace the values bound in the NDC picture, depending on the values of the source attributes, to create the logical picture.

The backdrop has the identification attributes of its output primitives removed to ensure that the operator cannot interact with the backdrop. No view selection takes place. As can be seen by the dashed lines emanating from the bundle table and colour table, the backdrop has the relevant bundle table information bound to it in the same way that the output primitives from the NDC picture have the information bound to create the logical picture. An identity NDC to LDC transformation is applied. The backdrop has the workstation transformation and clip applied and the colour information for indirectly specified colours is extracted from the colour table for primitives for which colour is specified indirectly.

The logical picture in LDC coordinates is transformed and clipped to produce the realized picture in DC coordinates. It is possible to postpone both the NDC scissor and view scissor to take place after the transformation of the logical picture and before workstation clipping. At this stage, colour information for indirectly specified colours is extracted from the colour table so that the realized picture is completely defined ready for display.

It is possible to store the realized picture as a sequence of output primitives in the realized metafile. Realized metafiles can be read back into the backdrop.

6 The Graphical Kernel System

6.1 Initialization

To activate GKS, the function OPEN GKS is invoked. This defines a set of data structures (called *state lists* and *description tables*) which define the characteristics of the implementation of GKS in use. State lists are dynamic and may change during program execution. Description tables define the characteristics of a particular system. The following data structures are initialized on GKS being opened:

a) Operating state list: defines the state of GKS.

b) GKS description table: gives information about the workstations and fonts available.

c) GKS state list: provides information about the state of GKS as the execution of a program progresses. On OPEN GKS being invoked, it is set up with predefined default values.

d) Generic workstation description tables: these each describe the characteristics of a type of workstation. One is provided for each type in the GKS implementation accessed.

Inquiry functions are provided to access all the information in the data structures. The functions SAVE GKS STATE LIST and RESTORE GKS STATE LIST allow parts of the GKS state list to be saved by the application and restored.

To deactivate GKS, the function CLOSE GKS is invoked. The data structures listed above no longer exist.

6.2 Graphical output

6.2.1 Output primitive classes

Output primitives are generated by GKS. Output primitives may be stored in the picture part store, or dispatched to the router for routing to the NDC picture or backdrop.

The basic geometry of the output primitive is part of its definition. Additional properties of an output primitive are defined by primitive *attributes* as described in 6.2.2.

Output primitives are divided into the following output primitive classes:

a) CURVE: generates a set of curves
b) MARKER: generates a set of symbols
c) AREA: generates a set of areas with edges
d) CHARACTER: generates a sequence of glyphs
e) IMAGE: generates an array of cells with individual colours
f) DESIGN: generates graphical output by extruding a tiling through a stencil

There is one kind of output primitive which does not belong to any particular class called the *generalized drawing primitive* (GDP) (see 8.8).

6.2.2 Output primitive attributes

Output primitives have identification, NDC, source and logical attributes. All the relevant primitive attributes (see 11.1.5) are bound to an output primitive when it is created by invoking the function CREATE OUTPUT PRIMITIVE. For the backdrop, identification attributes are removed after output primitives have been selected for display on a workstation so that it is not possible for the operator to interact with the backdrop. The function SET PRIMITIVE ATTRIBUTE defines all the attribute values in the GKS state list that may be bound to a primitive on creation. One value is set at a time by specifying the attribute and the new attribute value to be associated with it. This value is bound on creation to all subsequent primitives to which it refers until the value is reset by another invocation of SET PRIMITIVE ATTRIBUTE for this specific attribute. All

attributes of output primitives in the current NDC picture may be changed by the function SET NDC PICTURE PRIMITIVE ATTRIBUTE.

All output primitives have the *nameset* and *pick identifier* identification attributes associated with them.

Output primitives that make up the current NDC picture can have their NAMESET attributes changed by invoking one of the functions ADD SET OF NAMES TO NDC PICTURE, or REMOVE SET OF NAMES FROM NDC PICTURE. The functions ADD SET OF NAMES TO NAMESET and REMOVE SET OF NAMES FROM NAMESET allow the current nameset value in the GKS state list to be modified.

The PICK IDENTIFIER attribute provides an additional level of identification used in input (see 9.2).

All output primitives have the following three NDC attributes associated with them:

a) LOCAL TRANSFORMATION

b) GLOBAL TRANSFORMATION

c) SCISSOR SET

For some classes of output primitives, additional NDC attributes are defined. NDC attributes complete the geometric definition of output primitives.

The geometry of the primitives is transformed by the LOCAL TRANSFORMATION matrix and then the GLOBAL TRANSFORMATION matrix before NDC scissoring takes place. The two transformations are defined with respect to the NDC coordinate space. Global transformations allow transformations to be applied to complete objects while local transformations can be used to position individual output primitives.

Scissoring of the NDC picture restricts the appearance of each output primitive either by clipping or shielding. NDC scissoring is specified by the SCISSOR SET attribute which consists of a set of named scissors. Scissors are named by scissor identifiers. A scissor consists of a clipping indicator, clipping rectangle set, shielding indicator and shielding rectangle set. Clipping and shielding rectangles are defined in NDC. Clipping and shielding are activated by the clipping indicator and shielding indicator respectively. When NDC scissoring takes place depends on the scissor mode set by invoking the SET SCISSOR MODE function. If scissor mode is set to LOCUS, NDC scissoring occurs after the local and global transformations have been applied and, conceptually, before the output primitive is sent to the relevant workstations. A part of an output primitive is visible if its locus is outside all shield rectangles and inside at least one clip rectangle. If the clip rectangle set is empty and the clip indicator is CLIP, all of an output primitive is visible. If scissor mode is set to SHAPE, NDC scissoring is postponed until after the workstation transformation. Any relevant transformations are applied to the NDC scissor so that its effect, if postponed, is the same as when it is applied immediately except for the differences due to scissoring on shape rather than locus.

The functions ADD SET OF SCISSORS TO NDC PICTURE and REMOVE SET OF SCISSORS FROM NDC PICTURE allow a specified set of scissors to be added to or removed from the SCISSOR SET attribute of a group of primitives in the NDC picture. The function SET NDC PICTURE PRIMITIVE ATTRIBUTE can be used to replace the SCISSOR SET attribute bound to a group of primitives.

The functions ADD SET OF SCISSORS TO SCISSOR SET and REMOVE SET OF SCISSORS FROM SCISSOR SET allow the current scissor set attribute value in the GKS state list to be modified.

Current values of NDC attributes associated with specific primitives are defined in world coordinates and are stored in the GKS state list in world coordinates. When these NDC attributes are bound to their respective primitives, the values are subject to the same transformations as the geometric data contained in the definition of the primitive.

Source attributes are bound to an output primitive when the primitive is created. Each class of primitive has two source attributes, a set of ATTRIBUTE SOURCE FLAGS (ASFs) which determine the source of logical attributes which are applied in the logical picture, and a bundle index for that class of primitive. Each bundle index value is an index into a bundle table for that class of primitive, which exists on each workstation (stored in the workstation state list). The values in a particular bundle (or entry in a bundle table) may be different for different workstations. The ASFs control whether the values of the logical attributes of a primitive at the logical level are taken from the values bound to the primitive at the NDC level or from the bundle table. A full

description of logical attributes is given in clause 8.

6.3 Normalization transformations

The application programmer can compose a graphical picture from separate entities each of which, conceptually, is defined with its own world coordinate system (WC). The relative positioning of the separate entities is defined by having a single normalized device coordinate space (NDC) onto which all the defined world coordinate systems are mapped. A set of normalization transformations defines the mappings from the world coordinate systems onto the single normalized device coordinate space, which can be regarded as a workstation independent abstract viewing space. On creation, output primitives have world coordinate positions transformed to NDC coordinates before being dispatched to the router or stored in the picture part store. A single normalization transformation is current at any one time and this is used for the transformation from WC to NDC.

A normalization transformation is specified by invoking the function SET WINDOW AND VIEWPORT which defines the limits of an area in the world coordinate system (window) which is to be mapped onto a specified area of the normalized device coordinate space (viewport). Window and viewport limits specify rectangles parallel to the coordinate axes in WC and NDC (see figure 5). The rectangles include their boundaries. The normalization transformation performs a mapping from WC onto NDC that includes translation and differential scaling with positive scale factors for the two axes.

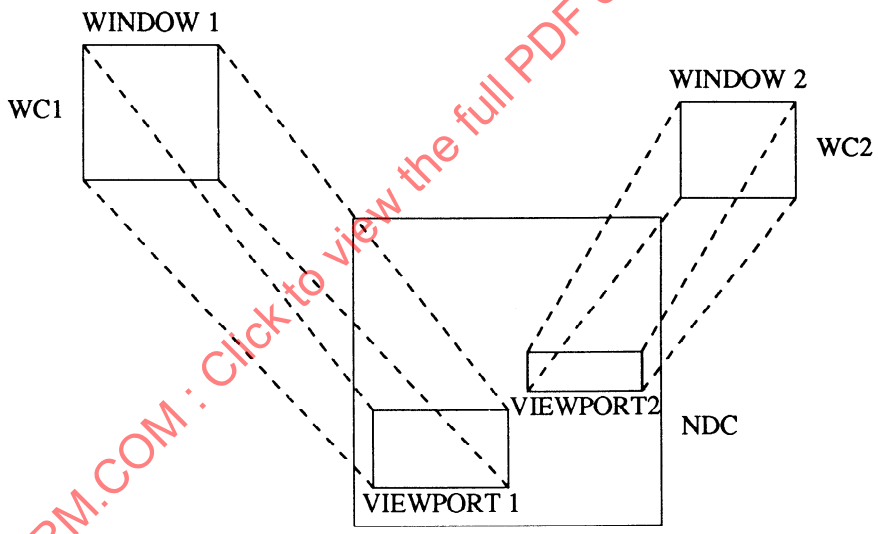

Figure 5 - Normalization transformations

Although NDC space conceptually extends to infinity, an implementation may support only a restricted range of NDC. Coordinates in the closed range [-7,7] × [-7,7] are always handled.

Each normalization transformation is identified by a *normalization transformation number* which is a non-negative integer. The normalization transformation with normalization transformation number 0 is the identity transformation which maps [0,1] × [0,1] in world coordinates to [0,1] × [0,1] in normalized device coordinates. It cannot be changed.

Initially, all other normalization transformations are set to a default transformation which is the same as normalization transformation number 0. Different transformations can be specified at any time when GKS is open. Since GKS provides a number of different normalization transformations, it is possible for the application program to specify them prior to outputting the graphical picture. The separate entities in the picture are output by selecting a particular normalization transformation before outputting the associated graphical

primitives. However, specifying a normalization transformation while the graphical output is taking place is allowed.

A normalization transformation can be selected by SET NORMALIZATION TRANSFORMATION NUMBER, and it will be used for all output until another is selected. By default, normalization transformation number 0 is selected.

6.4 Picture part store

6.4.1 Picture part creation

Output primitives on creation may be stored in a picture part or dispatched to the router. When an OPEN PICTURE PART function is invoked, primitives are stored in the picture part and dispatching ceases. When CLOSE PICTURE PART is invoked, the sequence of primitives that make up the picture part is added to the picture part store. Subsequent primitives are dispatched to the router. The picture part is identified by a unique, application specified picture part name. Primitives in the picture part have attributes bound to them in the same way as primitives dispatched to the router.

6.4.2 Picture part functions

The following functions are provided to manipulate picture parts:

a) RENAME PICTURE PART: changes the name of the picture part specified to a new name.

b) DELETE PICTURE PART: removes a picture part from the picture part store.

c) REOPEN PICTURE PART: reopens a picture part. Output primitives created are added in sequence to the end of the specified picture part until CLOSE PICTURE PART is invoked.

d) APPEND PICTURE PART: adds one picture part to the end of the other picture part. The output primitives can be altered in a similar way to COPY PICTURE PART FROM PICTURE PART STORE (6.4.3).

6.4.3 Adding picture parts

A picture part can be dispatched to the router by invoking COPY PICTURE PART FROM PICTURE PART STORE. In sequence, each output primitive in the picture part has its global and local transformation attributes replaced, or pre or post concatenated by the specified transformation matrices, and the specified set of names is added to the NAMESET attribute already associated with the output primitive. This allows several instances of a picture part to appear in the NDC picture in different positions and these can be differentiated by their primitives having different NAMESET attributes. The ability to change or reset the global and local transformation attributes allows the incoming primitives to be consistent with the primitives already in the NDC picture. Depending on a 'scissor select' parameter, either the current scissor set is bound to the primitives copied or the scissor set bound to the primitives in the picture part store is used.

Utility functions (EVALUATE TRANSFORMATION MATRIX and ACCUMULATE TRANSFORMATION MATRIX) are available to assist the application in establishing a transformation matrix.

6.4.4 Picture part archiving

A picture part can be archived by invoking the function ARCHIVE PICTURE PART. Each archive consists of a set of picture parts with names local to the archive. On archiving, an archive name is supplied. A picture part can be retrieved from the archive by invoking the function RETRIEVE PICTURE PART FROM ARCHIVE. On retrieval, a picture part name must be supplied. An error occurs if there is a conflict between an existing name in the archive or picture part store and the name supplied.

6.5 The NDC picture

6.5.1 Introduction

The NDC picture at any time consists of a sequence of output primitives which have been added to the picture in one of three ways:

a) on creation outside the definition of a picture part;

b) by copying from a picture part;

c) on input from an NDC metafile.

The order of appearance of output primitives in the NDC picture is important in that if two output primitives overlap, the second primitive in the sequence may obscure some part of the first.

Each workstation is responsible for selecting the subset of the NDC picture to be viewed and rendering it.

6.5.2 NDC picture operations

The following operations can be applied to the NDC picture:

a) DELETE PRIMITIVES FROM NDC PICTURE: the primitives whose NAMESET attribute in the NDC picture satisfies the specified selection criterion are deleted from the NDC picture.

b) REMOVE SET OF NAMES FROM NDC PICTURE: the specified names are removed from the NAMESET attributes of all primitives in the NDC picture which satisfy the specified selection criterion.

c) ADD SET OF NAMES TO NDC PICTURE: the specified names are added to the NAMESET attributes of all primitives in the NDC picture which satisfy the specified selection criterion.

d) SET NDC PICTURE PRIMITIVE ATTRIBUTE: the primitives satisfying the specified selection criterion have the value of the specified attribute changed.

e) REMOVE SET OF SCISSORS FROM NDC PICTURE: the specified scissors are removed from the SCISSOR SET attributes of all primitives in the NDC picture which satisfy the specified selection criterion.

f) ADD SET OF SCISSORS TO NDC PICTURE: the specified scissor set is added to the SCISSOR SET attributes of all primitives in the NDC picture which satisfy the specified selection criterion.

g) COPY NDC PICTURE TO PICTURE PART STORE: the primitives satisfying the specified selection criterion are copied to the picture part store as the specified picture part. The NAMESET attribute associated with each primitive has the specified set of names removed.

h) REORDER NDC PICTURE: The subsequence of primitives satisfying the source selection criterion are moved in front of the first position in the remaining subsequence of primitives satisfying the reference selection criterion (BACK) or after the last position in the remaining subsequence of primitives satisfying the reference selection criterion (FRONT). The relative order of primitives which satisfy the selection criterion is not changed by this operation.

6.5.3 NDC metafiles

It is useful to be able to capture a subsequence of primitives of the NDC picture satisfying a selection criterion, and store it away for future use or transmission to other systems. The function COPY NDC PICTURE TO NDC METAFILE will store a subsequence of the NDC picture as a picture on the specified metafile adding a set of names to the NAMESET attribute of each primitive of the subsequence. The picture can be recovered at a later time and added to the current NDC picture by invoking COPY NDC METAFILE PICTURE TO NDC PICTURE. A set of names may be added to the NAMESET attribute of each primitive recovered. Adding a set of names to the NAMESET attribute allows the primitives captured or recovered to be identified. The backdrop cannot be sent to an NDC metafile.

6.6 Selection criterion

Functions in GKS select subsequences of output primitives (for example, subsequences of the NDC picture or of a picture part) according to a *selection criterion*. A typical function of this type is ADD SET OF NAMES TO NDC PICTURE. A primitive will be included in the subsequence if its NAMESET attribute satisfies the selection criterion.

Selection criteria are constructed from comparison operations and logical operations. The comparison operations allowed are:

a) CONTAINS

b) ISIN

c) EQUALS

d) SELECTALL

e) REJECTALL

The first three operations specify a nameset which is compared with the nameset of each output primitive in the sequence.

All namesets satisfy the SELECTALL selection criterion and no namesets satisfy the REJECTALL criterion. These selection criteria can thus be used to select or reject all primitives in a sequence respectively.

More than one comparison operation can be included in the selection criterion by using the following logical operations:

f) AND

g) OR

h) NOT

The selection criterion CONTAINS (RED,GREEN), for example, will select all output primitives having namesets containing both names RED and GREEN. The selection criterion CONTAINS (RED,GREEN) OR CONTAINS (BLUE,YELLOW) will select those primitives with namesets containing either RED and GREEN or BLUE and YELLOW.

The selection criterion ISIN (RED,GREEN) will select those primitives whose namesets are subsets of the set (RED,GREEN). In this case that is primitives with the namesets (RED), (GREEN), (RED,GREEN) and the empty nameset.

A precise definition of selection criteria is given in 11.1.18. The sequence of primitives from which a selection is to be made are tested one by one against the selection criterion and, if accepted, are included in the resulting sequence and, if rejected, are excluded.

6.7 Graphical input

6.7.1 Introduction to logical input devices

An application program obtains graphical input from an operator by controlling the activity of one or more logical input devices, which deliver logical input values to the program.

The *device identifier* defines the class of logical input device and how it relates to physical devices on a workstation. The logical input device class determines the type of logical input value delivered. The classes and the logical input values they provide are:

a) LOCATOR: a position in world coordinates and a normalization transformation number.

b) STROKE: a sequence of points in world coordinates and a normalization transformation number.

c) VALUATOR: a real number.

d) CHOICE: a CHOICE status and a positive integer which represents a selection from a number of choices.

e) PICK: a PICK status, a nameset and a pick identifier.

f) STRING: a character string.

g) COMPOSITE: an application defined logical input device which delivers values that are composites of values of the above types.

Each logical input device may be operated in three modes, called *operating modes*. At any time a logical input device is in one, and only one, of the modes set by the invocation of the function SET LOGICAL INPUT DEVICE MODE. The three operating modes are REQUEST, SAMPLE and EVENT. Input from devices is obtained in different ways depending on the mode as follows.

h) REQUEST: A specific invocation of REQUEST INPUT causes an attempt to read a logical input value from a specified logical input device. This can only occur when the logical input device is in REQUEST mode. GKS waits until the input is entered by the operator or a break action is performed by the operator. The break action is dependent on the logical input device and on the implementation. If a break occurs, the logical input value is not valid.

i) SAMPLE: A specific invocation of SAMPLE INPUT causes GKS, without waiting for an operator action, to return the current logical input value of a specified logical input device. This can only occur when the logical input device is in SAMPLE mode.

j) EVENT: GKS maintains one input queue containing temporally ordered event reports. An event report contains the identification of a logical input device and a logical input value from that device. Event reports are generated asynchronously, by operator action only, from input devices in EVENT mode.

The application program can remove the oldest event report from the queue and examine its contents. The application can also flush from the queue all event reports from a specified logical input device.

A specific logical input device is said to be taking part in an interaction during the whole time that it is in SAMPLE or EVENT mode, but, when it is in REQUEST mode, only during the execution of a REQUEST INPUT function for that device. Alternatively, an interaction with the device can be said to be underway during that time. Many devices on many workstations may be taking part in interactions simultaneously.

6.7.2 Logical input device model

To describe the precise actions of the logical input devices, it is first necessary to describe their relationship with physical input devices, using the concept of measures and triggers.

A logical input device contains a measure, a trigger, an initial value, a prompt and echo type, an echo area and a data record containing details about the prompt and echo type. A logical input device's measure and trigger are parts of the implementation of the workstation containing the logical input device. Initial value, prompt and echo type, echo area, and data record can be supplied by the application program.

The measure of a logical input device is a value determined by one or more physical input devices together with a 'measure mapping'. More than one measure may simultaneously be determined by a single physical device; a separate measure mapping applies for each measure. A measure can be seen as the state of an independent, active process (a measure process). Each state corresponds exactly with a logical input value.

The current state of the measure process (i.e. the device's measure) is available as a logical input value. Whenever the device is taking part in an interaction, the measure process is in existence. Under other conditions, this process does not exist.

While the measure process exists, if echoing is required, information indicating the current state of the measure process is provided to the operator.

The trigger of a logical input device is a physical input device or a set of them together with a 'trigger mapping'. The operator can use a trigger to indicate significant moments in time. At these moments, the trigger is said to 'fire'. A single operator action (for example, pressing a button) causes the firing of not more than one trigger. Several logical input devices may refer to the same trigger.

A trigger can be seen as an independent, active process (a trigger process) that sends a message to one or more recipients when it fires. A logical input device is a recipient of its trigger if there is a pending REQUEST for it or if it is in EVENT mode. Both of these conditions can be true simultaneously for different logical input devices. If there is at least one recipient for a trigger, the trigger process is in existence. Under other conditions this process does not exist.

If a REQUEST for a logical input device is pending when the device's trigger fires, the measure of that device is used to satisfy the REQUEST (see figure 6). If one or more devices containing a given trigger are in EVENT mode when the trigger fires, the identifications of those devices and their measure values are passed to the input queue mechanism as separate event reports. The input queue mechanism is described in detail in 6.7.4.

When a trigger firing succeeds in satisfying a REQUEST, or adding event records to the input queue, GKS provides to the operator an acknowledgement the form of which depends on the implementation of the logical input device. The acknowledgement is not controllable by a GKS function.

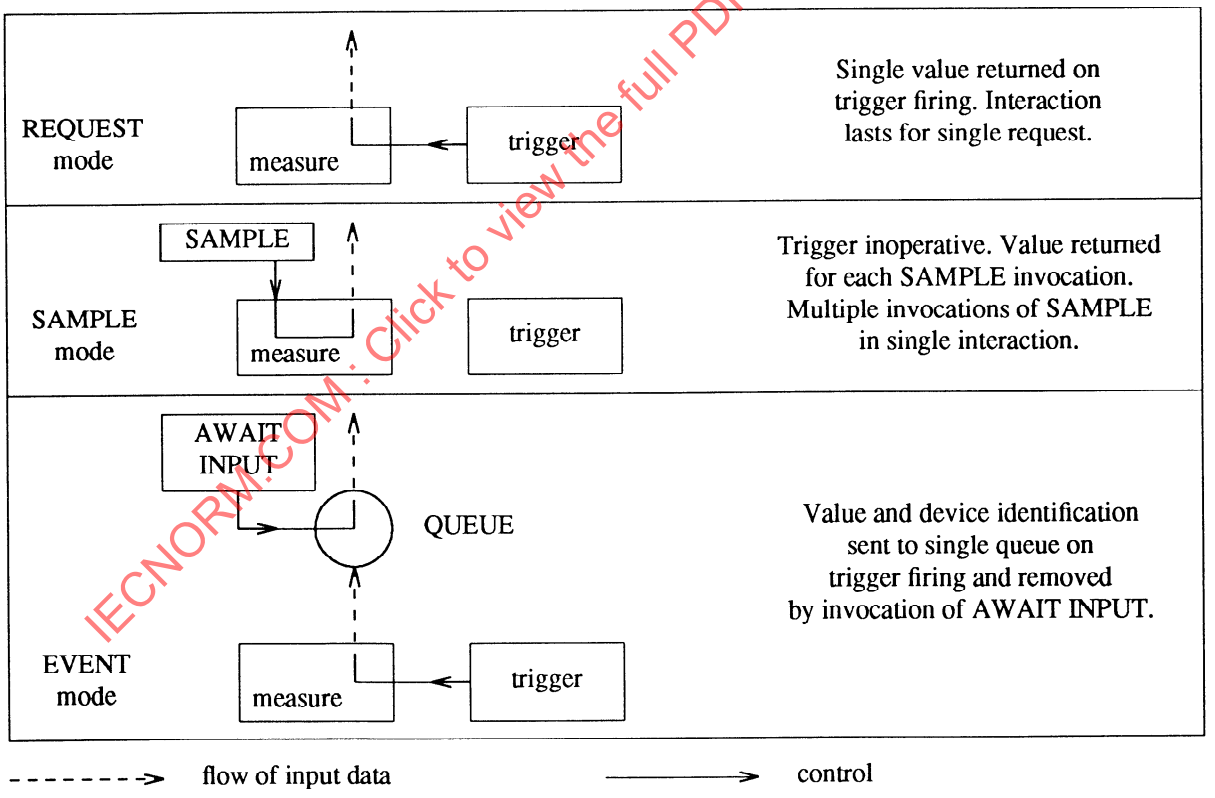

Figure 6 - Operating modes using measures and triggers

6.7.3 Operating modes of logical input devices

The mode of a logical input device can be changed by invoking the appropriate SET LOGICAL INPUT DEVICE MODE function.

After an invocation of SET LOGICAL INPUT DEVICE MODE with the parameter REQUEST, no measure process exists for the specified device and the device's identifier is not on its trigger's list of recipients. After an invocation with the parameter EVENT, a newly initiated measure process is in existence for the specified device and the device's identifier is on its trigger's list of recipients. After an invocation with the parameter SAMPLE, a newly initiated measure process is in existence for the specified device, but the device's identifier is not on its trigger's list of recipients.

Initially a logical input device is in REQUEST mode. While a device is in REQUEST mode, a logical input value can be obtained by invoking the appropriate REQUEST INPUT function. The effects of doing so are:

a) To create a measure process for the specified device and set its initial value. Echoing is performed by the measure process if echoing is on for the specified device.

b) To add the device's identifier to its trigger's list of recipients. If the list was previously empty, the trigger process is started.

c) To suspend GKS until the trigger of the specified device fires, or the operator invokes the break facility.

d) If the trigger fired, to set the logical input value to the current state of the measure process.

e) To destroy the measure process.

f) To remove the device's identifier from its trigger's list of recipients. If this list becomes empty, the trigger process is destroyed.

g) If the trigger fired, to return the logical input value and the status OK, otherwise to return the status NONE.

While a logical input device is in SAMPLE mode, a logical input value can be obtained by invoking the SAMPLE INPUT function. The effect of doing so is to set the logical input value to the current state of the measure process without waiting for a trigger firing.

While a logical input device is in EVENT mode, logical input values are added as event reports to the input queue, and can be obtained in sequence by invoking AWAIT INPUT. When an event report is generated, the state of the measure process depends on the input device class. For all classes other than STRING (and any STRING component of a COMPOSITE class), the state of the measure process is unchanged. For STRING class input, it is reset to its initial value.

Figure 6 shows the effect of every operating mode on the measure and trigger of a logical input device.

6.7.4 Input queue and current event report

The input queue contains zero or more event reports. Event reports contain pairs of values (device identifier, logical input value) resulting from trigger firings. Event reports may be added to the input queue when logical input devices in EVENT mode are triggered by the operator. Events can be removed from the input queue by invocations of AWAIT INPUT (see figure 6) and FLUSH DEVICE EVENTS.

When a trigger that is part of one or more logical input devices in EVENT mode fires, the resulting event reports are entered into the queue. A set of event reports, one for each device is added to the input queue, if and only if there is room for the whole set of simultaneous event reports. AWAIT INPUT returns all the simultaneous event reports together.

If there is not room in the queue for all event reports when a trigger fires, input queue overflow has occurred. Input queue overflow is not reported to the application program immediately. It is reported via the error mechanism during the next invocation of any GKS function that can remove event reports from the input queue (AWAIT INPUT, FLUSH DEVICE EVENTS, and CLOSE WORKSTATION (see 7.3)). The input queue has to be emptied before further event reports will be added. Between the detection of input queue overflow and the next time AWAIT INPUT is invoked with the input queue empty, no events are generated by trigger firings and thus no acknowledgements are provided. (This permits the application program to

determine how many events were in the queue when overflow occurred by calling AWAIT INPUT with zero timeout.)

When the 'input queue overflow' error is reported, the trigger causing the overflow is indicated by placing into the error state list the identification of any one of the logical input devices using that trigger which was in EVENT mode at the time the overflow was detected.

AWAIT INPUT, if the queue is not empty, removes the first event report and returns it to the application program. If the queue is empty, AWAIT INPUT suspends execution until an event report is queued or until the specified timeout period has elapsed.

FLUSH DEVICE EVENTS removes all event reports for a specific device from the input queue.

6.8 Inquiry functions

Inquiry functions return values directly from or derived from the various state lists and description tables. The data types of the values and the default values of the entries are summarized in clause 11.

The inquiry functions are designed in such a way that they do not cause any errors to be generated. Inquiry functions for values that may be logically unavailable have an output parameter, 'error indicator', that determines whether or not the other returned values are valid. The availability parameter is of the type integer and, in the event of the other values not being available, returns an error number, which identifies the appropriate GKS error condition. The same error numbers are used for inquiry functions as for other functions and thus the standard list of error messages should be consulted. If GKS is not in the proper state, then the error number appropriate to this condition is the one returned, even if there are other reasons for the values being unavailable. If the values are available, zero is returned in the error indicator parameter.

For all values except zero the returned output values are implementation dependent. The description of each inquiry function lists the error indicator values that the function may return.

6.9 Error handling

For each GKS function, a finite number of error situations is specified, any of which will cause the ERROR HANDLING function to be invoked. The ERROR HANDLING function defined invokes an ERROR LOGGING function which appends an error message and identification of the GKS function (which caused the error) to the error file before returning to the ERROR HANDLING function.

This two stage invocation of error handling allows the ERROR HANDLING function provided to be replaced by the application while still having access to services provided by the ERROR LOGGING function.

Each GKS function checks that it is used appropriately and that the values of input parameters are valid before attempting to execute the function. At least the first error detected will cause the ERROR HANDLING function to be invoked.

The application supplied ERROR HANDLING function is only allowed to invoke GKS inquiry functions, the ERROR LOGGING function and the EMERGENCY CLOSE GKS function. Inquiry functions are not allowed to generate errors.

If the application detects errors outside GKS and regains control, it can invoke the EMERGENCY CLOSE GKS function which will attempt to save as much of the graphical information as possible. GKS itself may invoke the EMERGENCY CLOSE GKS function if it gets into difficulty.

6.10 Special interfaces between GKS and the application program

A uniform escape mechanism for allowing access to installation and hardware specific features (a 'standard way of being non-standard') is provided by means of the ESCAPE function. Although the use of this mechanism reduces the portability of the application program, it does so in an easily identifiable manner.

The ESCAPE function does not generate graphical output; by contrast, the GENERALIZED DRAWING PRIMITIVE may generate graphical output not otherwise generated by GKS.

6.11 Backdrop

By default, routing is to the NDC picture.

Invoking the function ROUTE with the parameter BACKDROP will cause output primitives to be routed directly to the backdrop. Invoking the function ROUTE with parameter NDC will cause output to be added to the NDC picture again.

Primitives, as they are routed to the backdrop, have their identification attributes removed after selection for display on a workstation and, therefore, cannot be picked.

6.12 Audit and playback

The function AUDIT can be used to control the capture of GKS functions executed by an application. The parameters to the AUDIT function are the name of the audit trail and the operation to be performed (which, for the OPEN operation, includes the specification of the connection to be established). Multiple audits are allowed simultaneously. The initialization operation OPEN should be performed before an audit is started. The operation BEGIN starts the recording of audit trail information and END stops it. BEGIN and END operations may be invoked several times during an audit so that a selection of the GKS functions invoked are recorded. The operation CLOSE stops the audit and releases the audit trail file.

For each GKS function executed, the function name and a list of the parameter values are stored in the set of open audits when recording is enabled. For functions, such as input and inquiry, both the parameter values input to the function and the output values generated by the function are recorded. Non-graphical information supplied by the user can also be added to the audit by the function WRITE USER RECORD TO AUDIT. This can be used to add information such as times of entries or the association of application specific data with the graphical information.

To playback a previously recorded audit, four functions are provided (PLAYBACK, READ ITEM FROM AUDIT, READ ITEM FUNCTION NAME FROM AUDIT, PROCESS AUDIT ITEM). The function PLAYBACK is used to OPEN and CLOSE the session for the specified audit trail replay. The information stored on the audit trail consists of a sequence of items, one per GKS function invoked. The function READ ITEM FROM AUDIT returns the name of the function executed and a list of parameter values associated with that execution of the function. The function READ ITEM FUNCTION NAME FROM AUDIT just returns the name of the function. The function PROCESS AUDIT ITEM identifies the audit and has a parameter which takes the values SKIP or DO. The SKIP operation changes the item pointed at to the next on the specified audit trail. Initially, the audit trail is positioned at the first item. The DO operation has a null effect for those functions not concerned with output. For functions related to output (described in 12.2, 12.3, 12.4, 12.5, 12.6, 12.7, 12.8, 12.10, 13.1 and 14), the effect is as if the GKS function was executed directly at this point with the parameters defined in the audit trail. The PROCESS AUDIT ITEM function changes the item pointed at to the next item in either case of the operation.

The function AUDIT invoked with the operation parameter BEGIN or END will cause the corresponding item to be added to the audit trail. These can be used to identify the individual parts of the overall audit trail.

7 Workstation dependent control

7.1 Introduction

When a workstation is opened, a selection criterion is established for the workstation which selects part of the NDC picture and backdrop for rendering on the workstation. For each output primitive, the local and global transformations are applied and then either the NDC scissor is applied before rendering takes place or it is postponed until after the workstation transformation. The operation of rendering the NDC picture on a workstation proceeds in three stages. In the first stage, the selection criterion for each view on the workstation is used to define how many views the output primitive will appear in. For each of these, the output primitive is first reorientated, then mapped to a new position and locus in LDC space; finally it is either scissored by the view scissor, or the view scissor is postponed until after the workstation transformation and the NDC scissor is applied. In the second stage, any values of logical attributes required from the workstation dependent bundle tables are bound and the resulting primitives are rendered to produce the logical picture (see figure 4). In the logical picture, colour is not resolved beyond colour specifier or colour array specifier. The third stage applies the workstation transformation (and if postponed, applies the NDC scissor followed by the view scissor) and workstation clip, and resolves colour specifiers (replacing colour indices by the colours defined in the colour table for the workstation) to produce the realized picture which is displayed on the workstation display space. The realized picture may be sent to a metafile. The updating of the logical and realized pictures is continuous. Any change to the NDC picture produces a change as soon as possible in the logical picture. The realized picture will have any colour or workstation transformation changes made as soon as possible also. However, updating of the logical and realized pictures may be suspended in order that a group of updates can be made to appear atomic.

Rendering the backdrop on a workstation is similar. The output primitives routed to the backdrop have any values of logical attributes required from the workstation dependent bundle tables bound to them. The resulting primitives are rendered and the workstation transformation and clip applied. Finally, the colour specifiers are resolved to produce the backdrop which is displayed. The backdrop cannot be sent to a metafile. The backdrop cannot be suspended.

7.2 Workstation characteristics

For every type of workstation present in a GKS implementation a generic workstation description table exists which describes the standard capabilities and characteristics of the workstation. On opening a workstation, a specific workstation description table is created for that workstation containing information derived from: the generic workstation description table, the device itself, the application through the workstation specific information provided with the workstation type, and possibly other implementation dependent sources. The content of the specific workstation description table may change at any time while the workstation is open. Such changes are generated by events external to GKS. The application program can inquire which generic capabilities are available before the workstation is open. The specific capabilities can be inquired while the workstation is open. The application program can adapt its behaviour dependent on this information. If capabilities are requested that a particular workstation does not provide, or are not yet available, a standard reaction is defined.

The function INQUIRE SPECIFIC WORKSTATION DESCRIPTION TABLE ENTRY can be used to inquire the value of a workstation description table entry whilst the workstation is open. There is an entry state flag associated with each workstation description table entry for each workstation, which takes the values CHANGED and UNCHANGED. Whenever an entry in a specific workstation description table changes, the corresponding flag is set to CHANGED (for example, in a windowing environment when, as a result of an operator action, the display is resized). The value of the flag associated with a particular entry in the specific workstation description table of an open workstation is returned by the function INQUIRE SPECIFIC WORKSTATION DESCRIPTION TABLE ENTRY STATE. The function RESET SPECIFIC

WORKSTATION DESCRIPTION TABLE ENTRY STATE enables the application program to reset any particular flag to UNCHANGED.

An abstract graphical workstation with maximum capabilities

a) has one rectangular and addressable display space;

b) permits the specification and use of smaller display spaces than the maximum while guaranteeing that no display image is generated outside the specified display space;

c) renders the NDC picture to produce the logical picture, using (possibly workstation dependent) logical attributes, and binds indirectly specified colour values to produce the realized picture;

d) has one or more logical input devices for each input class;

e) permits REQUEST, SAMPLE and EVENT type input;

f) allows independent setting of logical input devices in REQUEST, SAMPLE or EVENT mode.

Actual workstations may provide more capabilities than those listed in the workstation description table. These cannot be used by GKS. However, if the workstation itself provides sufficient intelligence, the additional capabilities can be accessed via the ESCAPE function or GENERALIZED DRAWING PRIMITIVE, or used locally by the workstation operator. As an example, if a workstation has two display surfaces, the operator can switch locally from one to the other without notifying GKS or the application program. More than one display space can be controlled by GKS only by defining a separate workstation for each display space.

7.3 Selecting a workstation

The application program references a workstation by means of a workstation identifier. Connection to a particular workstation is established by the function OPEN WORKSTATION, which associates the workstation identifier with a workstation specifier and workstation type. The current state of each open workstation is kept in a workstation state list. Specified entries in the workstation state list can be saved and restored by invoking the functions SAVE WORKSTATION STATE LIST and RESTORE WORKSTATION STATE LIST.

Connections are released by the function CLOSE WORKSTATION.

7.4 Selection criteria

Selection criteria are provided to control which primitives in the NDC picture are displayed on a workstation and the highlighting and detectability of the primitives displayed. When a selection criterion is changed, the sequence of primitives in the NDC picture is examined and those that satisfy the new selection criterion are rendered again to produce a new realized picture for display or storage. First, the display selection criterion selects the sequence of primitives for display. The highlighting and detectability selection criteria determine which of this sequence of primitives is highlighted or made detectable.

For output primitives routed to the backdrop, only the display selection criterion has any effect and selects primitives for display. They are not highlighted or made detectable.

Clause 10 describes the segment and workstation activation facilities which are explicitly defined in order to provide compatibility with ISO 7942:1985. These facilities make use of the nameset and selection criteria mechanisms and consequently it is necessary to reserve a part of each selection criterion for manipulation by GKS. This is explained in clause 10. The function SET WORKSTATION SELECTION CRITERION sets the application specifiable part of the current selection criterion, without modifying the system controlled part.

When a workstation is opened, the selection criteria for display, highlighting and detectability are each set to REJECTALL. Consequently, no primitives will be selected for display on a workstation, unless the application first defines an appropriate selection criterion.

7.5 Viewing

Each output primitive selected for display on a workstation may appear in one or many views on that workstation. Initially, view 0 has the default selection criterion of SELECTALL and all other views have the default selection criterion of REJECTALL. Thus, initially all output primitives selected for display appear in view 0 and no other view. The function SET VIEW SELECTION CRITERION can redefine the selection criterion for each view on the workstation.

The function SET VIEW defines the orientation, mapping and scissoring to be applied to each output primitive selected for the view. *View orientation* effectively moves the NDC coordinate system to one more appropriate for the set of output primitives to be viewed. *View mapping* provides a mapping of the set of output primitives to a new position in LDC space. Finally, the *view scissor* parameter provides an optional view scissor on the output primitives being viewed. View scissoring either occurs here or may be delayed until after the workstation transformation and any delayed NDC scissoring.

View 0 provides an identity mapping from NDC to LDC, empty clipping and shielding rectangle sets and clipping and shielding indicators with values NOCLIP and NOSHIELD respectively. These parameters of view 0 cannot be changed.

Two utilities are provided for the construction of the view orientation and mapping matrices. The function EVALUATE VIEW ORIENTATION MATRIX defines a matrix that will move the origin of the set of output primitives and redefine the Y direction. The function EVALUATE VIEW MAPPING MATRIX defines a matrix that performs a window to viewport mapping from NDC space to LDC space.

The NDC picture is displayed through views in order of view priority. The primitives satisfying the selection criterion of the lowest priority view are displayed first in temporal order. Primitives in views of higher priority are displayed after those of lower priority. View priority is set by the function SET VIEW PRIORITY. View priority also affects LOCATOR and STROKE input measures (see 9.2).

7.6 Workstation transformations

An application can select independently for each open workstation some part of the LDC space in the range $[0,1] \times [0,1]$ to be displayed somewhere in the workstation display space. A particular workstation transformation is a mapping from LDC space onto the device coordinates (DC) for that workstation. Thus picture composition can be achieved using the normalization transformations. View orientation and view mapping allow different views of the NDC picture to be assembled in the logical picture. The workstation transformation allows different areas of the logical picture to be viewed on different workstations. For example, a correctly scaled drawing could be output to a plotter and simultaneously some part of the drawing could be displayed on the surface of an interactive terminal.

The units of device coordinates are metres on a device capable of producing a precisely scaled image (for example, on most plotters) and appropriate workstation dependent units otherwise (for example, on a display unit with unknown monitor size). In either case the device coordinate system maps onto the display space in the following way:

a) the DC origin is at the bottom left corner of the display space;

b) the device coordinate units are related to the display space in such a way that a square in device coordinates appears as a square on the display surface;

c) x and y increase to the right and upwards respectively.

On some devices, device coordinate units do not coincide with the addressable units used by the display, for example if the addressable units do not satisfy the above conditions.

The current size of the display space in device coordinate units is recorded in the specific workstation description table (see 7.2).

A default workstation transformation is provided when the workstation is opened. The workstation transformation can be redefined at any time while the workstation is open. The workstation transformation is a uniform mapping from LDC onto DC and thus performs translation and *equal* scaling with a positive scale factor

for the two axes. A workstation transformation is specified by defining the limits of an area in the LDC system within the range [0,1] × [0,1] (workstation window) which is to be mapped onto a specified area of the device coordinate space (workstation viewport). Workstation window and workstation viewport limits specify rectangles parallel to the coordinate axes in LDC and DC. The rectangles include their boundaries. To ensure that no output outside the workstation window is displayed, GKS clips at the workstation window boundaries, and this clipping cannot be disabled. As the workstation window is defined to be somewhere in the LDC range [0,1] × [0,1], this ensures that the only part of LDC space that can be viewed on any workstation lies in the range [0,1] × [0,1]. If the workstation window and workstation viewport have different aspect ratios, the scaling specified would be different on each axis if the workstation window was mapped onto the workstation viewport in its entirety. To ensure equal scaling on each axis, the workstation transformation maps the workstation window onto the largest rectangle that fits within the workstation viewport such that:

d) aspect ratio is preserved;

e) the lower left-hand corner of the workstation window is mapped to the lower left-hand corner of the workstation viewport.

Thus, space is left unused at the top or right side of the workstation viewport if the aspect ratios of the workstation window and workstation viewport are different.

When a display is addressed through a window manager, the window provided by the window manager is analogous to the display surface of a conventional display. The limits of the display space can be inquired from the workstation description table. The workstation viewport is then defined relative to the display space in the usual way. The window manager is required to retain the contents of the display surface and do damage repair correctly.

All workstation transformations are set by default to map LDC space [0,1] × [0,1] onto the whole of the workstation display space. If the display space is not square, the same rules as above apply to achieve equal scaling on each axis.

Workstation transformations are changed by the SET WORKSTATION WINDOW AND VIEWPORT function.

7.7 Output primitives

Each output primitive has four classes of attributes (identification, source, NDC and logical). When a primitive is created, the current values of all the attributes of the primitive in the GKS state list are bound to the primitive. The values of attributes of primitives in the NDC picture can be changed by editing (see 6.5.2).

Each workstation has a set of *bundle tables,* one per primitive class. A *bundle index* is defined for each class of bundle table. Each bundle table entry contains values for a set of logical attributes. For example, the curve bundle table has a set of entries each of which contains values for CURVETYPE, CURVEWIDTH SCALE FACTOR and CURVE COLOUR SPECIFIER accessed via the curve bundle table index.

Values for each of the logical attributes are also present in the GKS state list. The GKS state list also contains entries for an index value for each bundle table and sets of *attribute source flags* (ASFs), one for each logical attribute and grouped by primitive class. The initial values of all the ASFs are INDIVIDUAL. The bundle index and set of ASFs for a particular primitive class constitute the source attributes of that primitive class.

At the workstation, values of logical attributes of output primitives are determined in the following way:

a) for all logical attributes for which the ASF is INDIVIDUAL, the values already bound to the primitive are used directly;

b) for all logical attributes for which the ASF is BUNDLED, the values bound to the primitive are replaced by the values of the corresponding attributes in the bundle table entry indexed by the bundle index bound to the primitive.

Figure 7 shows diagrammatically how the binding takes place. The logical attributes for a particular primitive are bound to a primitive on creation from the GKS state list. At the same time, the attribute source flags (represented here by arrows, up representing INDIVIDUAL, down representing BUNDLED) together with the

class index into the bundle table are also bound. All these attributes are available for editing in the NDC picture. In the creation of the logical picture, the index into the bundle table selects a bundle table entry. The direction of the arrow decides whether the INDIVIDUAL values already bound to the primitive are retained, or the values in the bundle table entry are bound to the primitive.

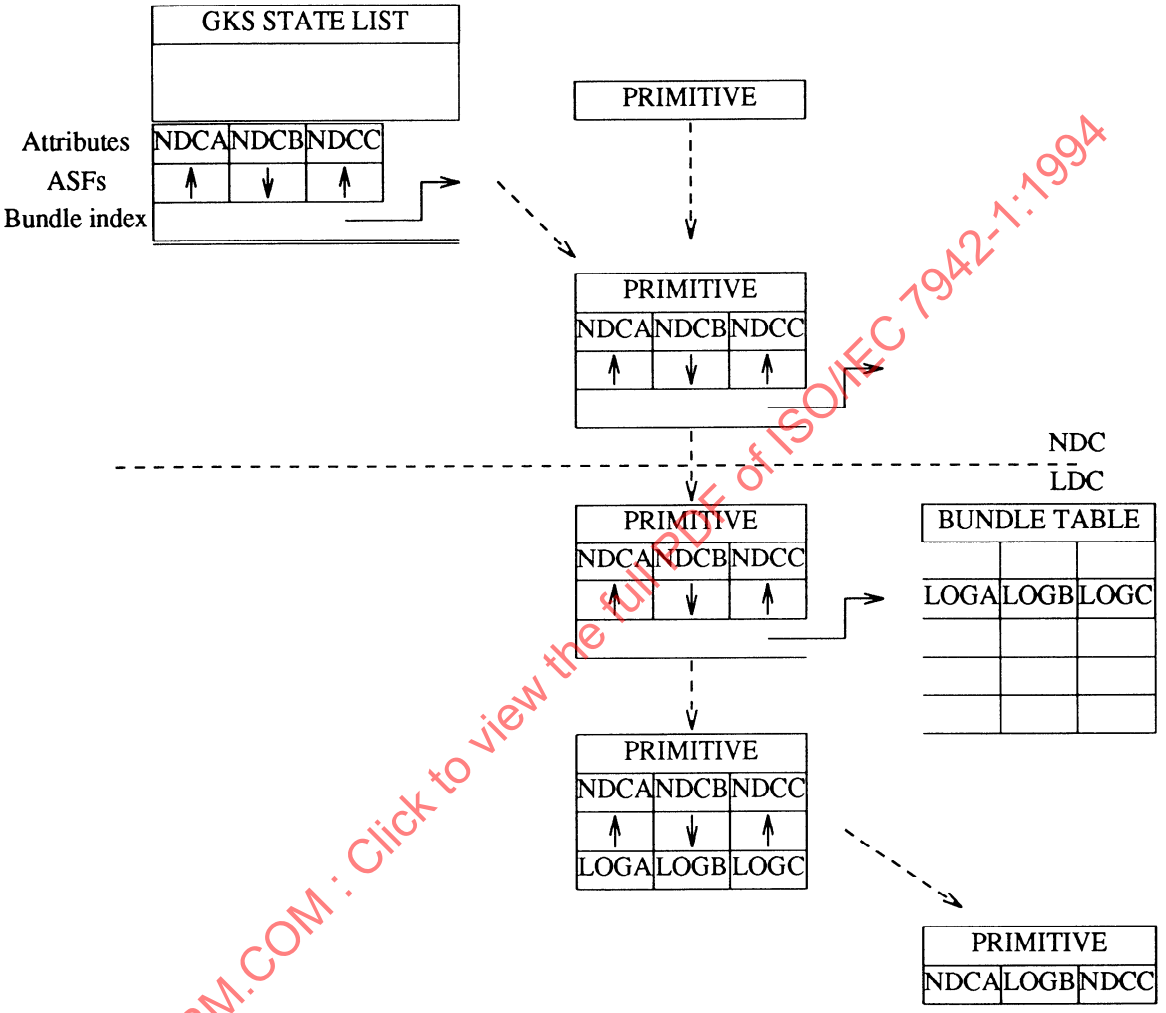

Figure 7 - Effect of different ASF settings

7.8 Colour

Colour is specified in a number of different situations. It may be an attribute of a primitive, in which case it is specified by a colour specifier. It may be part of a pattern for area primitives, or part of the primitive itself (namely for CELL ARRAY); in both these cases, colour is specified by a colour array specifier.

Colour specifiers may be of two colour types. If the colour type is indirect the colour specifier contains a *colour index*, an index into the workstation *colour table*. Alternatively the colour specifier contains a colour model name together with a specification of colour coordinates in the colour space of that model. Colours in the workstation colour table are defined by a colour model name and colour coordinates in that model.

A *colour array specifier* specifies an array of colours. This may be either an array of colour indices or a colour model name and an array of colour coordinates in that colour model.

Colours are associated with a colour index by the function SET REPRESENTATION (see 7.9). The function INQUIRE WORKSTATION STATE LIST ENTRY can be used to determine the colour value associated with a colour index. The parameters of these functions describe colour as a specification of colour coordinates in the colour space of the specified colour model. The number of coordinates necessary to specify colour, and their interpretation, depends on the colour model.

Four colour models, RGB, CIELUV, HSV and HLS are predefined. The colour models RGB and CIELUV shall be provided. Note that some enumeration values are reserved for registration. The four predefined colour models are described in Annex E.

On each workstation, there is one colour table into which all the colour indices point. The size of the colour table is workstation dependent but entries 0 and 1 always exist. Entry 0 corresponds to the background colour. The background colour is the colour of the display space after it has been cleared. Entry 1 is the default foreground colour and entries higher than 1 correspond to alternative foreground colours. INQUIRE SPECIFIC WORKSTATION DESCRIPTION TABLE ENTRY can be used to determine the predefined colour representations for a workstation. The colour coordinates are returned in an appropriate colour model for the workstation.

When colours are realized they are mapped to the nearest available on the workstation. Some workstations do not have the capability to change the background colour, and in this case the mapping of a specific colour to the nearest available for the background colour may be different from the mapping of the same colour for the foreground colours.

The 'colour characteristics of display' entry in the workstation description table describes the colour capabilities of the display and provides information that enables the application to make best use of the colour capabilities of the workstation. For workstations capable of producing precise colours the 'colour characteristics of display' entry provides sufficient information to the application to enable precise colour specifications to be set up. The information returned is workstation dependent.

Some workstations are not capable of displaying colours (for example, workstations only capable of displaying colours with equal red, green, and blue intensities or workstations only capable of displaying colours which are different intensities of the same colour); these are referred to as monochrome workstations. Whether a workstation is capable of colour is recorded in the 'colour available' entry in the workstation description table. If the 'colour available' entry is set to MONOCHROME, the intensity is computed from the colour values in a workstation dependent way (see Travis, Annex F).

The function CONVERT COLOUR can be used to convert colour specifiers to colour coordinates in a specified colour model. The conversion is performed according to the conversion algorithms used by a specified workstation and hence can be used to determine how a particular colour specifier will be realized by a particular workstation.

7.9 Setting representations

The function SET REPRESENTATION defines an entry in a bundle table, the pattern table or the colour table of a workstation. Some standard definitions for data entries are contained in the workstation description table and are used as initial values. The application program can select a standard definition or can define the value of a specific entry explicitly. Only the most commonly used (or anticipated) combinations of values need be predefined for each workstation. At least these predefined entries with indices up to the minimum number of predefined entries are distinguishable from each other. Other combinations of values can be set by the SET REPRESENTATION function, possibly after inquiring the workstation capabilities. The tables which are on every workstation are:

curve bundle table
marker bundle table
area bundle table
  pattern table
character bundle table

colour table

The values in these tables may be changed at any time. Changes only effect the rendering of primitives in the logical picture. If the change of representation were to have any effect on the backdrop, the backdrop is removed from this workstation. It is possible to set an entry to undefined as a means of recovering storage.

7.10 Removing a backdrop

The backdrop on a workstation can be removed by invoking the function REMOVE BACKDROP. (The backdrop can also be removed by invoking the function CLEAR WORKSTATION.) Changes to the workstation state list can change the NDC picture. If such changes occurred while similar output was routed to the backdrop, the backdrop is removed since it is not always possible to update the backdrop.

7.11 Visual effect state

The function SET WORKSTATION VISUAL EFFECTS allows the application to suppress and resume updating of the logical and realized pictures. This allows a group of updates to be made to appear atomic.

When visual effects state is SUPPRESS, changes have no effect on the logical and realized pictures. When routed to the backdrop, any primitive should be routed to the open workstations irrespective of the workstations' visual effects state. When visual effects state is ALLOW, updating of the logical and realized pictures is continuous. When the visual effects state is changed from SUPPRESS to ALLOW, changes made whilst the visual effects state was SUPPRESS now produce effects in the logical and realized pictures.

A function GET WORKSTATION STATUS allows the application to ascertain when all the suppressed effects (for example, from a group of updates) have been accomplished.

7.12 Realized metafile

The contents of the realized picture may be captured and stored for future use or transmission to another system. The function COPY REALIZED PICTURE TO REALIZED METAFILE will store the realized picture as a picture on the specified metafile. Blank pictures may be inserted in the realized metafile. This can be achieved by invoking the function COPY BLANK REALIZED PICTURE TO REALIZED METAFILE. The picture can be recovered at a later time and added to the current backdrop by invoking COPY REALIZED METAFILE PICTURE TO BACKDROP. The backdrop cannot be captured in a realized metafile. The external representation of the realized metafile is not defined in this International Standard.

7.13 Logical input devices

7.13.1 Introduction

A workstation may have one or more logical input devices for each input class. A logical input device is identified by a logical input device identifier which consists of a workstation identifier, a device class and the device number within that class on that particular workstation.

The application program has some degree of control over logical input devices in that it can supply an initial value for the device, set an area of the display space for the display of prompt and echos and select prompting and echoing techniques to be used.

The application program can also define the composition of compound logical input devices in terms of the *measures* and *triggers* provided by the workstation. A measure is a value which is determined by one or more physical input devices and a mapping from the values delivered by the physical devices. A trigger is a

condition which determines when a logical input value is delivered by a logical input device.

7.13.2 Initialization of logical input devices

Logical input devices may only be initialized when in REQUEST mode. The INITIALIZE LOGICAL INPUT DEVICE function provides the following information to a device via the workstation state list (if the INITIALIZE LOGICAL INPUT DEVICE function is not invoked, then default values apply):

a) An initial value appropriate to the device. If the initial value violates the rules, an error occurs and the workstation state list is unchanged.

b) A prompt and echo type that selects the prompting technique and, if echoing is on, the echoing technique for a logical input device. An implementation dependent prompt and echo type (type 1) is required for all logical input devices. Further prompt and echo types are defined but not required. Prompt and echo types above those are reserved for registration (see 4.2). Prompt and echo types less than 0 are device dependent.

c) An echo switch that controls whether echoing is on or off.

d) An echo area (xmin,xmax,ymin,ymax) in device coordinates. Input device implementations may use the echo area for certain prompt and echo types to display prompts or echoes. The specified echo area need not be within the current viewport. An input device implementation is not required to use the specified echo area; in a windowing environment echoing may be provided outside the display space.

e) A data record. Some input classes have mandatory control values in the data record. Some prompt and echo types within an input class also have mandatory control values in the data record. These values occupy well defined places in the data record. In any data record used in initializing an input device, values mandatory for the input class, if any, appear first, followed by values mandatory to the prompt and echo type, if any. Depending on the device and prompt and echo type, the data record may contain other (additional) information.

When a logical input device is REQUESTed, or when it is set to EVENT or SAMPLE mode, its measure is set to the initial value from the workstation state list, unless this is not a valid measure for the device. If it is not a valid measure for the device, the measure is set to a device dependent value, except for PICK devices, for which the measure is set to NOPICK.

Prompt and echo types describe both the prompt, which informs the operator that the device is available, and the echo, which informs the operator of the state of the measure. In addition, an implementation dependent acknowledgement of successful trigger firings is provided.

7.13.3 Definition of logical and composite input devices

The application program can construct logical input devices of composite types using the function DEFINE LOGICAL INPUT DEVICE. Each workstation supporting input devices provides a number of measures and triggers that map onto physical devices associated with the workstation. The application can specify which of these measures and triggers are to be used in constructing the measure and trigger processes for a specified device. The value returned by such a device is a sequence of values of basic or constructed types. Satisfaction of any of the trigger conditions specified causes the corresponding logical input device's trigger to fire.

For example, an input device could be defined which returns two measures from a single physical device such as a mouse. The first measure could be derived from the mouse's position and the second measure could be derived from the current positions of the buttons on the mouse.

7.14 Sending messages to a workstation

The MESSAGE function allows a character string to be sent to a workstation. The application program has no control over the position and appearance of the character string and an implementation is allowed to place the string on a device distinct from, but associated with, the workstation. For example, in a windowing environment the message could appear in a pop-up window.

8 Output primitives

8.1 Introduction

Clause 6 contains a description of the overall capabilities of the output primitive classes and their attributes. This clause gives details of the output primitives and attributes in each class. The logical attributes for each class are given in the following table. The individual subclauses give their meaning.

Output primitive class	Logical attributes
CURVE	CURVETYPE CURVEWIDTH SCALE FACTOR CURVE COLOUR SPECIFIER
MARKER	MARKERTYPE MARKERSIZE SCALE FACTOR MARKER COLOUR SPECIFIER
AREA	INTERIOR STYLE INTERIOR STYLE INDEX INTERIOR COLOUR SPECIFIER EDGE FLAG EDGETYPE EDGEWIDTH SCALE FACTOR EDGE COLOUR SPECIFIER
CHARACTER	CHARACTER FONT AND PRECISION CHARACTER EXPANSION FACTOR CHARACTER SPACING CHARACTER COLOUR SPECIFIER
IMAGE	
DESIGN	

8.2 Curve output primitives

8.2.1 Curve primitives

The output primitives in the class curve are:

a) SET OF POLYLINE: generates a set of curves each of which is a sequence of connected lines defined by a point sequence.

b) SET OF NURB: generates a set of curves each of which is a nurb.

c) SET OF CONIC SECTION: generates a set of curves each of which is a conic section.

A nurb path is defined as follows. The nurb is specified by its order k (degree $k-1$), the knot sequence $(T_1, \cdots, \cdots, T_{n+k})$, where n is the number of control points, and the set of control points $\{\mathbf{P}_i\}^n$, a set of weights $\{W_i\}^n$, a minimum t_{min} and maximum t_{max} of an independent parameter t. For non-rational control points, the weights $\{W_i\}$ are all taken as 1. The knot sequence must satisfy (simultaneously):

$$T_k \leq t_{min} \leq t \leq t_{max} \leq T_{n+1}$$

$$T_1 \leq T_2 \cdots \leq T_{n+k-1} \leq T_{n+k}$$

The curve is the (infinite) set of points $\mathbf{P}$ for all possible values of t between t_{min} and t_{max}.

$$\{\mathbf{P}(t) : t_{min} \le t \le t_{max} \text{ and } \mathbf{P}(t) = \frac{\sum_{i=1}^{n} \mathbf{P}_i W_i B_{i,k}(t)}{\sum_{i=1}^{n} W_i B_{i,k}(t)}\}$$

where the Basis Functions $B_{i,k}$ are defined (using the knot sequence) as:

$$B_{i,0}(t) = \begin{cases} 1 & \text{if } T_i \le t < T_{i+1} \text{ and } T_i < T_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

$$B_{i,p}(t) = \frac{(t - T_i) B_{i,p-1}(t)}{(T_{i+p-1} - T_i)} + \frac{(T_{i+p} - t) B_{i+1,p-1}(t)}{(T_{i+p} - T_{i+1})}$$

where p=1..k. In the case of multiple identical knot values some denominators evaluate to 0, in such cases, as part of the above definition the indeterminate quantity $\frac{0}{0}$ is considered to be 0.

For the SET OF CONIC SECTION primitive, each conic section is defined by a 3 × 3 matrix, A, which specifies the conic and two points which define the start and end points of the conic section. The equation of the conic is:

$$\mathbf{X}^T A \mathbf{X} = 0 \quad \textit{where} \quad \mathbf{X} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

The last parameter of the conic section definition is a sense flag which indicates whether it is the clockwise or anti-clockwise section that is required when the conic is a circle or an ellipse. A number of utility functions EVALUATE CIRCULAR ARC 3 POINT, EVALUATE CIRCULAR ARC CENTRE, EVALUATE ELLIPTIC ARC, EVALUATE HYPERBOLIC ARC, and EVALUATE PARABOLIC ARC are provided (see 12.14) to define the conic section for circular, elliptic, hyperbolic and parabolic arcs. For EVALUATE CIRCULAR ARC CENTRE, EVALUATE ELLIPTIC ARC, and EVALUATE HYPERBOLIC ARC the start and end points of the arc are defined as the intersection of the conic with lines from the centre in the directions of specified start and end vectors.

8.2.2 Curve attributes

The curve logical attributes are CURVETYPE, CURVEWIDTH SCALE FACTOR and CURVE COLOUR SPECIFIER.

The defined curvetypes 1 to 5 are solid, dashed, dotted, dashed-dotted, and dashed-dotted-dotted. The curvetype specifies a sequence of line segments and gaps which are repeated to define the rendering of the curve primitive. It is workstation dependent whether this sequence is restarted or continued at the start of each curve in a set and at the start of a scissored piece of a curve primitive. For curve primitives defined by vertices, the sequence shall be continuous across vertices.

The curvewidth is calculated as a nominal curvewidth multiplied by the curvewidth scale factor. This value is mapped by the workstation to the nearest available curvewidth in the realized picture.

8.3 Marker output primitives

There is a single output primitive in the class marker which is POLYMARKER. This generates a set of symbols of one type centred at given positions.

The marker logical attributes are MARKERTYPE, MARKERSIZE SCALE FACTOR, and MARKER COLOUR SPECIFIER.

Markertypes 1 to 5 are dot, plus sign, asterisk, circle and diagonal cross each centred on the positions they are identifying.

The marker size is calculated as a nominal size multiplied by the markersize scale factor. This size is mapped by the workstation to the nearest available marker size in the realized picture. Marker type 1 is always

displayed as the smallest displayable dot in the realized picture irrespective of the markersize scale factor.

The marker is visible if, and only if, the marker position is not scissored. The scissoring of partially visible markers is workstation dependent.

8.4 Area output primitives

8.4.1 Area primitives

The output primitives in the class area are:

a) SET OF FILL AREA: generates a set of areas each of which is defined by a closed sequence of connected points.

b) SET OF CLOSED NURB: generates a set of areas each defined by a nurb which is closed by the first point being connected to the last.

c) SET OF ELLIPTIC SECTOR: generates a set of areas each defined by an elliptic arc. Each area is closed by lines from the ends of the arc to the centre of the ellipse.

d) SET OF ELLIPTIC SEGMENT: generates a set of areas each defined by an elliptic arc. Each area is closed by a line joining the ends of the arc.

e) SET OF ELLIPTIC DISC: generates a set of areas each defined by a complete ellipse.

The elliptic arcs in the SET OF ELLIPTIC SECTOR and SET OF ELLIPTIC SEGMENT primitives are defined by a 3×3 matrix which defines the ellipse, in the same way as the SET OF CONIC SECTION curve output primitives (see 8.2), and two points which specify the start and end point of the elliptic arc. Finally, a sense flag defines whether it is the clockwise or anti-clockwise arc from the start point to the end point. The utility functions EVALUATE CIRCULAR ARC 3 POINT, EVALUATE CIRCULAR ARC CENTRE and EVALUATE ELLIPTIC ARC are provided to define the elliptic arc.

The SET OF ELLIPTIC DISC primitive is defined by a 3×3 matrix which defines the ellipse. The utility functions EVALUATE CIRCLE and EVALUATE ELLIPSE are provided to define the 3×3 matrix.

Area primitives allow the specification of areas with holes or disjoint regions that should be treated as a single entity. For each area primitive, the areas may be hollow or filled with a uniform colour, a pattern, or a hatch style, with or without edges (the boundary of each closed region).

8.4.2 Definition of interior

The set of interior points of area primitives is defined in the following way (see figure 8). For any point not on the boundary of the area primitive, if the number of intersections between the boundary of the area primitive and any ray from the point to infinity (not passing through a vertex) is odd, the point is interior, otherwise it is outside. If a point is interior, it is included in the area to be filled, subject to scissoring. When an area primitive is scissored, the resulting new boundaries become part of the area boundaries (see figure 9). These boundaries are not displayed as edges. Multiple subareas may be generated.

8.4.3 Area attributes

The area primitives have the NDC attributes PATTERN REFERENCE POINT and PATTERN SIZE. If the PATTERN SIZE is (SX,SY), this defines the pattern height vector as (0,SY) and pattern width vector as (SX,0). These values are defined in WC and are subject to the same transformations as the geometric data contained in the definition of the primitive. The usage of these area NDC attributes is described later in this subclause.

The logical attributes of area primitives are INTERIOR STYLE, INTERIOR STYLE INDEX, INTERIOR COLOUR SPECIFIER, EDGE FLAG, EDGETYPE, EDGEWIDTH SCALE FACTOR and EDGE COLOUR SPECIFIER.

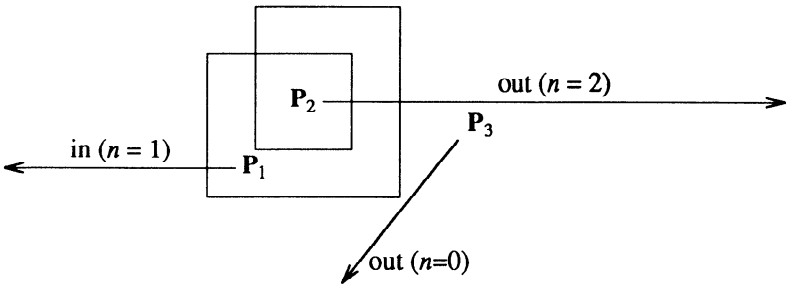

Figure 8 - The inside of an area

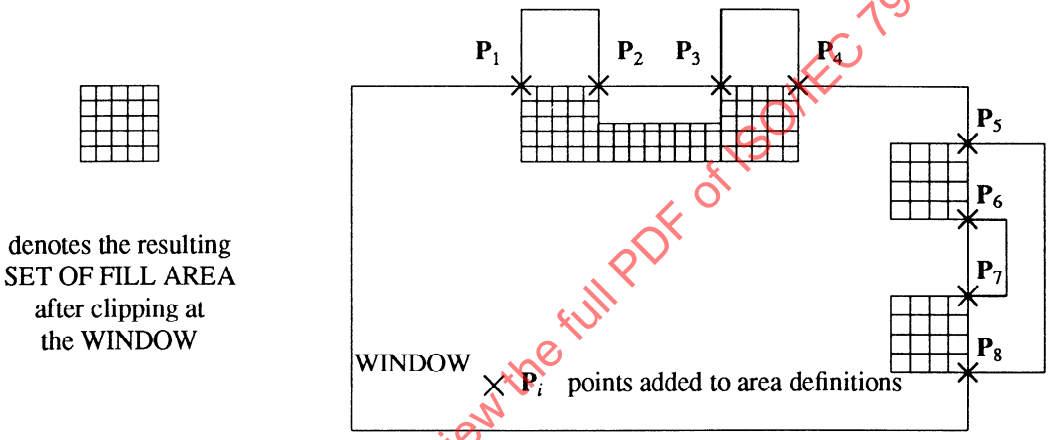

Figure 9 - Example of SET OF FILL AREA scissoring

The INTERIOR STYLE is used to determine in what style the areas should be filled. It has the following values and meanings:

a) HOLLOW: No filling, but draw the bounding curve of the area, using the INTERIOR COLOUR SPECIFIER currently selected. The curvetype and curvewidth are implementation dependent. The bounding curve is drawn along boundaries created by scissoring.

b) SOLID: Fill the interior of the area using the INTERIOR COLOUR SPECIFIER currently selected.

c) PATTERN: Fill the interior of the area using the INTERIOR STYLE INDEX currently selected as an index into the pattern table. In this context, the INTERIOR STYLE INDEX is sometimes referred to as the pattern index.

d) HATCH: Fill the interior of the area using the INTERIOR COLOUR SPECIFIER and the INTERIOR STYLE INDEX currently selected. The INTERIOR STYLE INDEX is used as a pointer into the list of hatch styles, in which case it is sometimes referred to as the hatch index.

e) EMPTY: No filling.

For interior style PATTERN, the pattern index selects among different patterns. The position in the pattern table defines a pattern representation, which specifies a colour array specifier of dimension (DX × DY). The size and position of the start of the pattern are determined by a pattern box. The pattern box, which is a parallelogram in NDC, is defined by the pattern width vector and the pattern height vector located relative to the PATTERN REFERENCE POINT. The pattern box is conceptually divided into a grid of DX × DY cells of

equal size. The colour array is associated with the cells as follows: the element (1,DY) is associated with the cell having the PATTERN REFERENCE POINT at one corner; elements with increasing first dimension are associated with successive cells in the direction of the pattern width vector; elements with decreasing second dimension are associated with successive cells in the direction of the pattern height vector. The attributes defining the pattern box are subject to all the transformations producing a transformed pattern box. The pattern is mapped onto the area by conceptually replicating the transformed pattern box in directions parallel to its sides until the interior of the complete area is covered.

Mapping the transformed pattern cells to the pixels of a raster display is performed by the following rules:

f) if the location of a pixel lies inside the parallelogram defined by the transformed cell, its colour is set;

g) the pixel is assigned the colour of the cell corresponding to the pixel's centre.

For a workstation which can implement patterns but not transformable patterns, a suitable action is to generate non-transformed patterns to fill an area.

For interior style HATCH, the hatch index selects among hatch styles. The defined hatch styles 1 to 6 are horizontal equally spaced parallel lines, vertical equally spaced parallel lines, positive slope equally spaced parallel lines, negative slope equally spaced parallel lines, horizontal/vertical cross hatch and positive slope/negative slope cross hatch. The ideal angle for positive slope hatch style is +45° and the ideal slope for negative slope hatch style is +135°.

Whether hatching is affected by transformations or not is workstation dependent.

Interior styles HOLLOW, SOLID and EMPTY are available on every workstation. It is workstation dependent which of the interior styles PATTERN and HATCH are available.

The other logical attributes permit independent control of the edges of area primitives. They are EDGE FLAG, EDGETYPE, EDGEWIDTH SCALE FACTOR and EDGE COLOUR SPECIFIER.

The EDGE FLAG may have values ON and OFF. When the value is OFF, the edge is not drawn. When the value is ON, the edge is rendered according to the other edge attributes. Edges are drawn on top of the interior. Conceptually edges and interior are disjoint; in practice, they can overlap. Parts of edges that are scissored are not visible. The defined edgetypes are the same as curvetypes and have the same meanings. The edgewidth is calculated as a nominal edgewidth multiplied by the edgewidth scale factor. This value is mapped by the workstation to the nearest available edgewidth in the realized picture. For example, if the value is less than or equal to zero, the thinnest available edgewidth is used.

For an area primitive with interior style HOLLOW, the rendering of the bounding curve may be overlaid by the rendering of the edge since the boundary is considered part of the interior. The bounding curve may be visible between gaps of non-solid edgetypes, or around the rendering of the edge.

New boundaries created by scissoring are not displayed as edges.

Area primitives with interior style EMPTY are visible only if edge flag is ON.

8.5 Character output primitives

8.5.1 Introduction

The only output primitive in the class character is TEXT which generates a sequence of glyphs specified as a sequence of character codes relative to a specified text position.

The NDC attributes associated with the TEXT primitive are TEXT HEIGHT, TEXT UP VECTOR, TEXT PATH, TEXT SKEW ANGLE, and TEXT ALIGNMENT. It also has the logical attributes of the character class associated with it, namely CHARACTER FONT AND PRECISION, CHARACTER EXPANSION FACTOR, CHARACTER SPACING and CHARACTER COLOUR SPECIFIER. The attribute CHARACTER FONT AND PRECISION defines font index and precision.

The definition of the graphical output proceeds as follows:

a) *Glyph specification*: The character codes in the parameter to the TEXT primitive together with the font index define the individual glyphs required.

b) *Glyph size*: The nominal sizes of the glyphs in the font are scaled using TEXT HEIGHT and CHARACTER EXPANSION FACTOR.

c) *Text extent*: The arrangement of the sequence of glyphs is established using TEXT PATH, and the extent of the sequence of glyphs is initially established from this using CHARACTER SPACING.

d) *Text skewing*: The text extent is modified by changing the posture of the individual glyphs defined by TEXT SKEW ANGLE.

e) *Text alignment*: The text extent is aligned to the text position specified in the TEXT primitive using the TEXT ALIGNMENT attribute.

f) *Text orientation*: The aligned text is orientated relative to the text position using the TEXT UP VECTOR attribute.

g) *Colour*: The colour of the graphical output is defined by the CHARACTER COLOUR SPECIFIER.

This gives the position and extent of the glyph sequence in world coordinates. Normalization transformations, transformations on picture parts, and view transformations may all cause asymmetric scaling of the glyphs in the sequence.

To allow workstations to support fonts which have more limited capabilities (for example, size change and orientation not available), three precisions are defined by the precision value (see 8.5.9). The explanation below is in terms of STROKE precision which can perform all the above functions.

8.5.2 Glyph specification

The font index value defines a particular font on the workstation. The sequence of character codes defined in the parameter to the TEXT primitive specify the glyphs to be used from the font selected.

The font designer specifies the shape of each glyph in a local 2D cartesian glyph coordinate system. For each glyph, the extent of the *glyph body* and the lines defined in figure 1 are specified by the font designer. For monospaced fonts, the glyph bodies of all glyphs have the same size. For proportionally spaced fonts, the width of the glyph bodies may differ from glyph to glyph. The height of a glyph in the glyph coordinate system is given by the height from the font baseline to the capline. The width of a glyph is given by the width of the glyph body (from the leftline to the rightline). The width of a glyph may include space on either side of the glyph and this space is generally evenly split between the left and right sides of the glyph. It is assumed that the glyphs lie within their bodies, except that kerned glyphs may exceed the side limits of the glyph body.

In general, the top limit of the glyph bodies for a font are identical with, or very close to, the typographical capline or ascender line, and the bottom limit to the descender line. The space, if any, between the topline and the capline may be used for an additional mark over the glyph, for example an accent. However, these and other details are purely for the use of the font designer. The intention is only that glyphs placed with their bodies touching in the horizontal direction should give an appearance of good normal spacing, and glyphs touching in the vertical direction will avoid clashes between ascenders and descenders (typographically 'set solid').

Every workstation supports at least two fonts, namely those associated with font indices 1 and 2. Each of these shall be able to generate graphical representations of the characters defined in ISO/IEC 646. The two representations shall be visually distinguishable from each other. The font associated with font index 1 is the monospaced font Courier. Font index values greater than 2 are reserved for registration. Font index values less than 0 may be supported but are implementation dependent. Each font supports the representation of all characters in all supported character sets. A request to represent a character not contained in the font will result in the generation of an implementation dependent representation which indicates that the requested representation is unavailable.

The GKS description table contains the default font index mapping which is operative when GKS is opened. The font associated with a font index value can be changed by the function SET FONT INDEX MAPPING.

This associates a specified font index value with one of the ISO/IEC 9541 font resources.

Individual character code representations may be replaced by different glyphs. The function GET GLYPH NAME returns the name of a glyph in a specified character code position of a font associated with a specified font index value. The function SET CHARACTER CODE assigns a specified glyph identified by its name to a specified character code position of a font defined by a font index value.

8.5.3 Glyph size

The TEXT HEIGHT attribute defines the height of the glyphs in the sequence of glyphs specified by the parameter in the TEXT output primitive. The glyph is expanded so that the distance from the baseline to the capline is TEXT HEIGHT. The width of the glyph from leftline to rightline is expanded by the same ratio.

The attribute CHARACTER EXPANSION FACTOR causes the width of the expanded glyph defined above to be increased or decreased in width by the factor specified. Examples of different character expansion factors are given in figure 10.

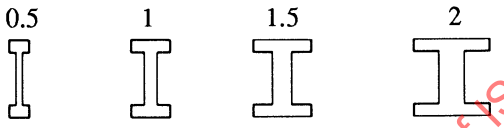

Figure 10 - Character expansion factor

The remaining attributes associated with the TEXT primitive effect the shape of the glyph by performing the same transformation on all points within the bounding box of the glyph. Consequently, the glyph can be represented by the bounding box as shown in figure 11.

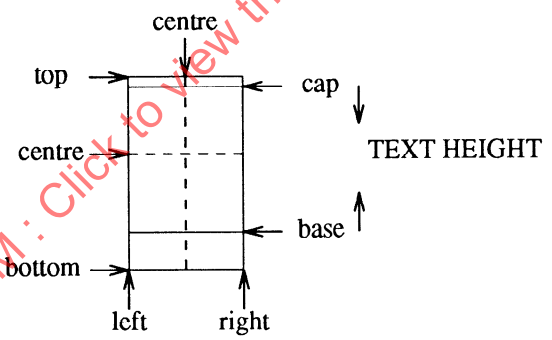

Figure 11 - Glyph bounding box

8.5.4 Text extent

The extent of the text in world coordinates is specified by the sequence of glyphs (which may have different widths), and the attributes TEXT PATH and CHARACTER SPACING. Before orientation, the up direction is defined as the direction from the baseline to the capline (perpendicular to the baseline) and the baseline direction is defined from the leftline to the rightline (perpendicular to the leftline).

TEXT PATH has the possible values RIGHT, LEFT, UP and DOWN. It specifies the escapement direction of the sequence of glyphs. For RIGHT, the sequence of glyphs is written in the baseline direction. The CHARACTER SPACING value specifies how much additional space is to be inserted between two adjacent glyph bodies. If the value of CHARACTER SPACING is zero, the glyph bodies are arranged one after the other along the TEXT PATH, without any additional space between. A positive value of CHARACTER SPACING will insert additional space between glyph bodies. A negative value of CHARACTER SPACING will cause adjacent glyph bodies to overlap. CHARACTER SPACING is specified as a fraction of TEXT HEIGHT.

Figure 12 gives examples of different glyph spacing in the RIGHT text path escapement direction.

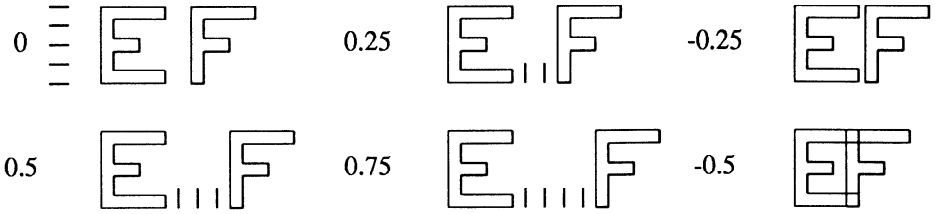

Figure 12 - Glyph spacing

Figure 13 shows the positions of three glyph bodies with CHARACTER SPACING of 0.25 for the TEXT PATH escapement directions RIGHT, LEFT, UP and DOWN. The first glyph is extra wide, the second narrow and the last is of normal size.

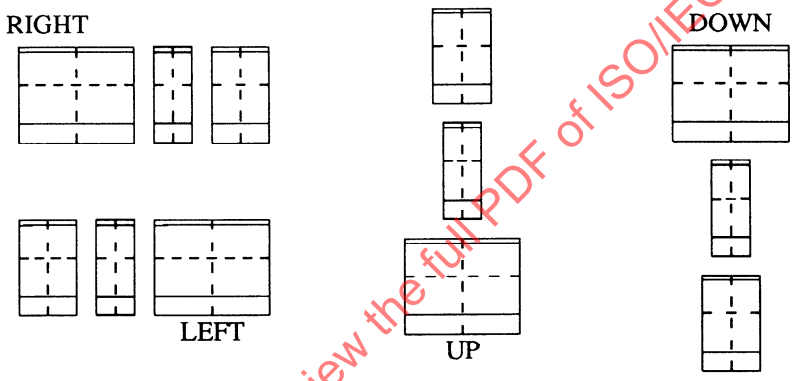

Figure 13 - Text paths

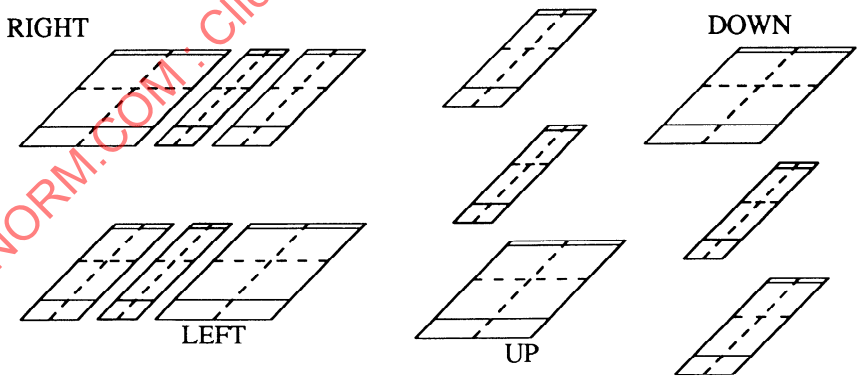

Figure 14 - Skewed text paths

8.5.5 Text skewing

The attribute TEXT SKEW ANGLE defines a skew to the posture of each glyph in radians relative to the up direction, thus a TEXT SKEW ANGLE of zero gives no skew and the glyph sequence will continue to appear as in figure 13. The effect of text skewing is shown in figure 14 for each of the TEXT PATH escapement directions using the glyph bodies defined in figure 13.

8.5.6 Text alignment

The TEXT ALIGNMENT attribute controls the positioning of the text extent in relation to the text position specified in the TEXT primitive.

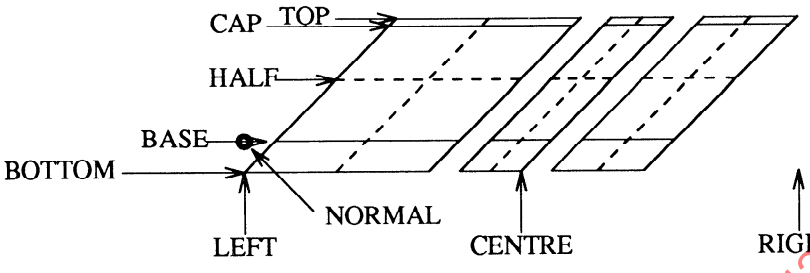

Figure 15 - RIGHT text alignment

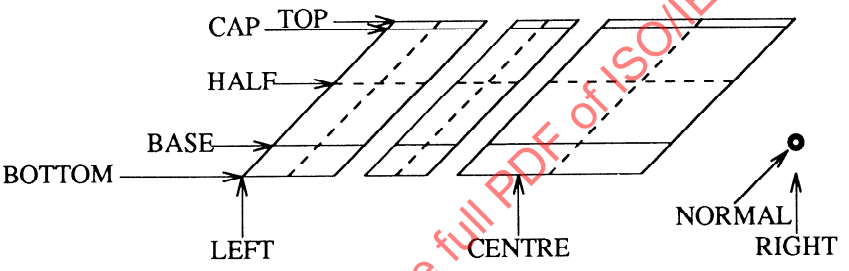

Figure 16 - LEFT text alignment

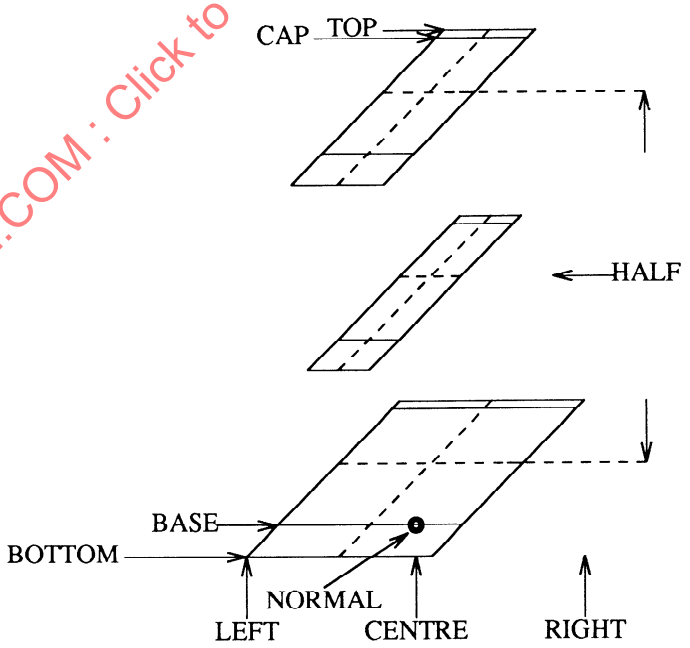

Figure 17 - UP text alignment

The horizontal component of TEXT ALIGNMENT has four values: LEFT, CENTRE, RIGHT and NORMAL.

The vertical component of TEXT ALIGNMENT has six values: TOP, CAP, HALF, BASE, BOTTOM and NORMAL.

The definitions of text alignment for the four TEXT PATH escapement directions are given in figures 15, 16, 17 and 18.

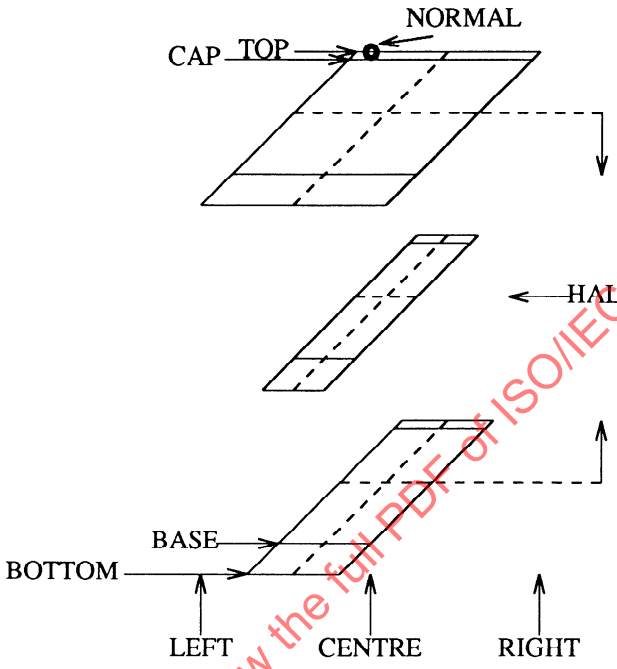

Figure 18 - DOWN text alignment

Examples of the horizontal and vertical text alignment for TEXT PATH RIGHT escapement direction are shown in figures 19 and 20.

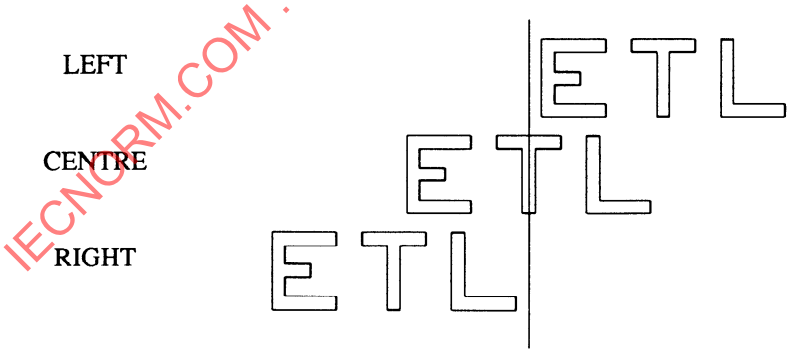

Figure 19 - Horizontal text alignment

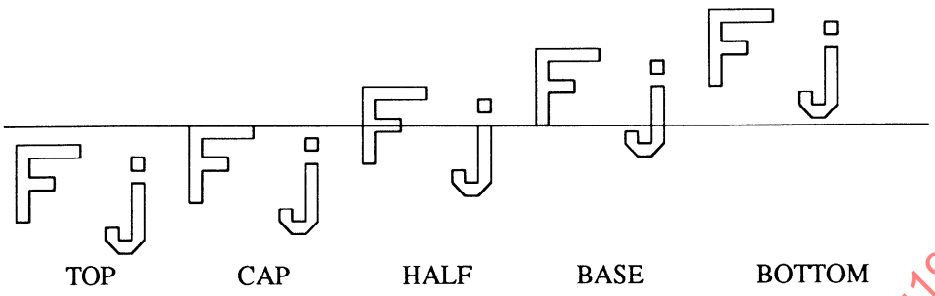

Figure 20 - Vertical text alignment

8.5.7 Text orientation

The complete text extent's position has been defined relative to the text position defined in the TEXT primitive assuming that the up direction is the vertical direction. The TEXT UP VECTOR parameter defines the up direction of the glyph sequence relative to the text position. Figure 21 shows how text in the four TEXT PATH escapement directions is reorientated if the TEXT UP VECTOR is not vertical but is instead at an angle of 45° anti-clockwise to the vertical.

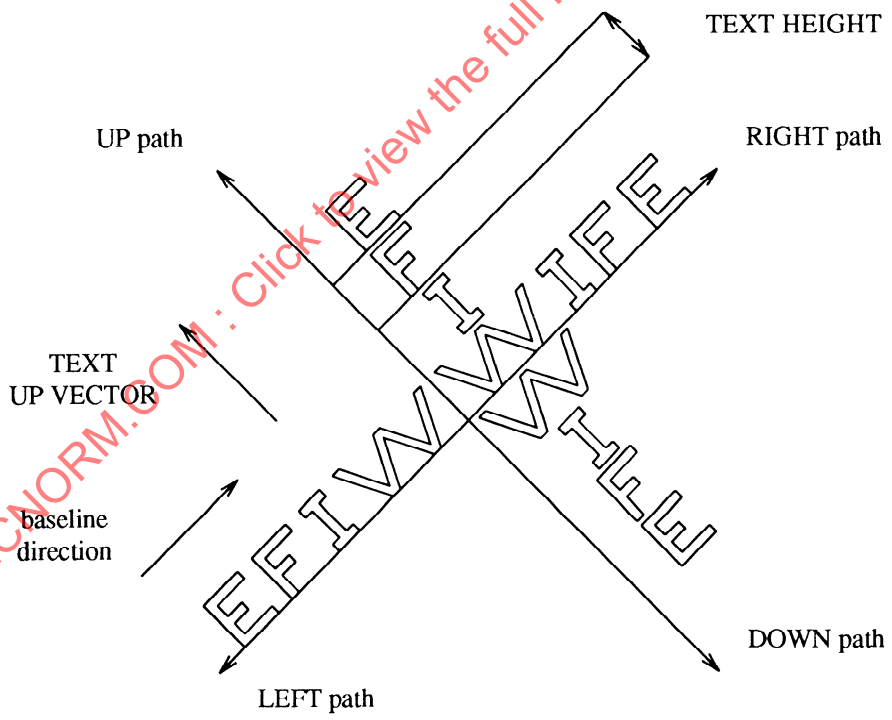

Figure 21 - Text geometric attributes

8.5.8 Transformed text

The values of TEXT HEIGHT and TEXT UP VECTOR are defined in world coordinates. Consequently, the final extent of the text defined so far is in world coordinates. All the points defining the locus of the individual glyphs will be transformed by the normalization transformation to produce the glyph outlines in NDC space. As this can be an asymmetric transformation, glyph deformation can take place. Figure 22 shows the effect of the normalization transformation.

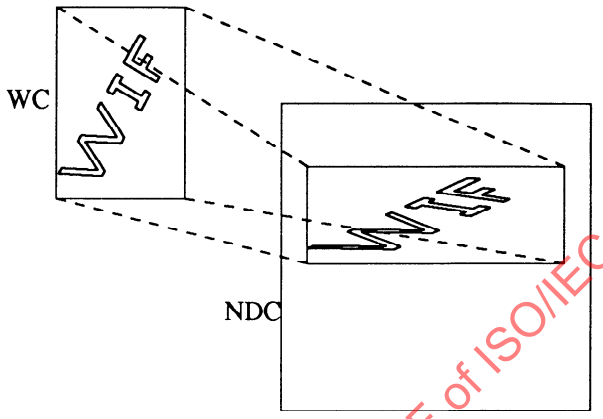

Figure 22 - Effect of normalization transformation on text

The glyph outlines will also be transformed by any transformation applied to picture parts, any viewing transformation and, finally, the workstation transformation before it is displayed. Consequently, it is likely that the glyph will be deformed in this process and the correct graphics on the workstation will only be possible if the font definition is one where the outlines of the glyph are defined.

8.5.9 Precision

The precision value is used to select the 'closeness' of the text representation at the workstation in relation to that defined by the NDC text attributes and the transformation and scissoring currently applicable. The precision value has the following possible values:

a) STRING: The TEXT glyph sequence is generated in the requested font and is positioned by aligning the TEXT output primitive at the given text position. TEXT HEIGHT and CHARACTER EXPANSION FACTOR are evaluated as closely as reasonable, given the capabilities of the workstation. TEXT UP VECTOR, TEXT PATH, TEXT ALIGNMENT, TEXT SKEW ANGLE and CHARACTER SPACING, need not be used. Scissoring is done in an implementation and workstation dependent way.

b) CHAR: The TEXT glyph sequence is generated in the requested font. For the representation of each individual glyph, the attributes TEXT HEIGHT, the implied width, the up direction of the TEXT UP VECTOR, the baseline direction, TEXT SKEW ANGLE and CHARACTER EXPANSION FACTOR are evaluated as closely as possible, in a workstation dependent way. The spacing used between glyph bodies is evaluated exactly; the glyph body, for this purpose, is an ideal glyph body, calculated precisely from the text attributes and the font dimensions. The position of the resulting text extent parallelogram is determined by the TEXT ALIGNMENT and the text position. Scissoring is performed at least on a glyph by glyph basis.

c) STROKE: The TEXT glyph sequence in the requested font is displayed at the text position by applying all text attributes. The sequence of glyphs is scissored exactly.

STROKE precision does not necessarily mean vector strokes; as long as the representation adheres to the rules governing STROKE precision, the font may be realized in any form, for example by raster fonts.

All character precisions are supported as follows. A workstation may use a higher precision than the one requested for this purpose, that is if STROKE precision is supported in a particular font, the implication is that both STRING and CHAR precision are available in that font. However it is not necessary for a workstation to support all precisions for a given font (i.e. for a given font, STROKE can be missing or both STROKE and CHAR can be missing). Every workstation of a particular installation should support at least two STROKE precision fonts. These are the fonts associated with font indices 1 and 2. Not all workstations need to support all fonts, but where the same font is supported on all workstations, the same font index value is used to select that font on all workstations of a particular installation.

8.5.10 Estimate of text extent

GKS provides a function GET TEXT EXTENT which returns an estimate of the area that a given glyph sequence, positioned relative to a specified text position will occupy when displayed with the current NDC, source and logical attributes on a specified workstation. A concatenation point which can be used as the text position of a subsequent TEXT output primitive for the concatenation of glyph sequences, is also returned where this is meaningful.

The function returns four points which clockwise from the bottom left define an area that encloses the set of glyphs in the specified string of text. The concatenation point returned can be used as the text position of a subsequent TEXT output primitive for the concatenation of glyph sequences, where meaningful. For example, if the extent of a string ABC is estimated with text position W and the concatenation point returned is Z, the graphical output produced on the workstation from outputting the string ABCDEF at text position W should be the same as outputting the string ABC at text position W and DEF at text position Z.

For certain combinations of TEXT PATH and TEXT ALIGNMENT, concatenation is not meaningful and the returned concatenation point is the same as the text position. For example, if the TEXT PATH is set to LEFT or RIGHT, horizontal text alignment of CENTRE is not meaningful for concatenation. For text paths UP and DOWN, horizontal text alignment of LEFT or RIGHT is not meaningful nor is vertical alignment of HALF.

At precisions STRING and CHAR, the text extent is an approximation of that defined above, being the minimum which completely encloses the glyph bodies of the displayed sequence.

8.6 Image output primitives

The only output primitive in the class image is *cell array*. This generates an array of cells with individual colours. It has no logical attributes and no additional NDC attributes.

A CELL ARRAY primitive is specified by a pair of points P, Q and a colour array specifier. The points P and Q define a rectangle aligned with the world coordinate axes which is divided into a grid of DX × DY cells, where DX and DY are the dimensions of the colour array. The colour array is oriented with respect to the rectangle as shown in figure 23. The grid is subject to all transformations, potentially transforming the rectangular cells into parallelograms.

The output primitive CELL ARRAY has all its attributes bound on creation. The realized picture is obtained by realizing all the entries in the colour array specifier. When the colour array specifier is an array of colour indices, the colour indices are replaced by the corresponding colours specified in the workstation's colour table.

Mapping the transformed cells onto the pixels of a raster display is shown in figure 24.

The following rules apply:

a) If the location of a pixel lies inside the parallelogram defined by the transformed rectangle, its colour is set.

b) The pixel is assigned the colour of the cell which contains the pixel's centrepoint. Thus, the pixel colour is selected by point sampling the transformed rectangle at the pixel centrepoint, not by area sampling or filtering.

The minimal rendering required is to draw the transformed boundaries of the cell rectangle, using implementation dependent colour, curve width and curve type.

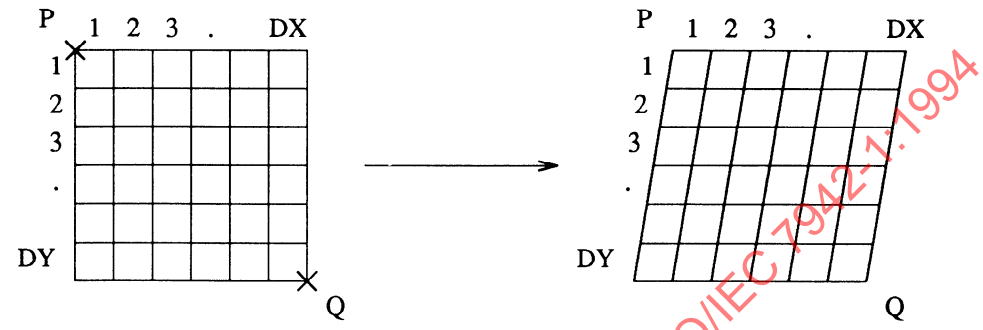

Figure 23 - Mapping of CELL ARRAYs

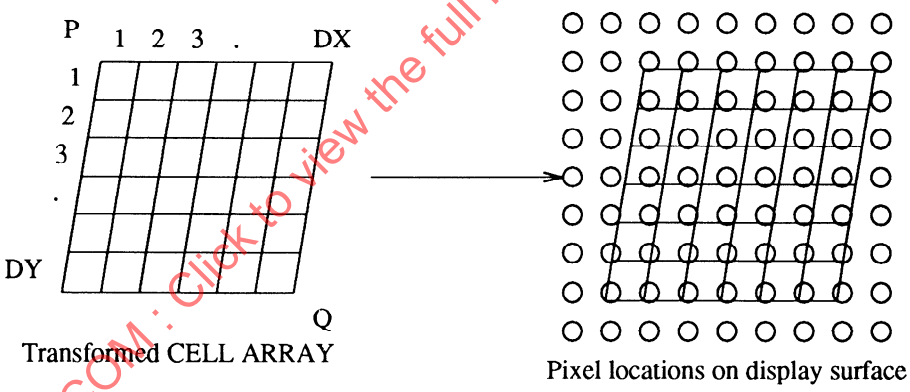

Figure 24 - Mapping CELL ARRAY cells onto pixels

8.7 Design output primitives

8.7.1 Introduction

Design primitives are defined by a stencil and an associated tiling. Stencils are stored in the *stencil store* and may be constructed from area boundaries, closed contours and other stencils. Tilings are stored in the *tiling store* and a number of predefined tilings and functions for the specification of tiles and their replication can be used in the construction of tilings. Stencils and tilings are all defined in their own coordinate systems. An instance of a design primitive is produced by extruding a tiling through a stencil. First, transformations are applied to the stencil and tiling to reorientate and scale them in their local coordinate system (LC). The origin of the stencil is colocated with a reference point in WC. Similarly, the origin of the tiling is colocated with a reference point in WC. The overall scheme is illustrated in figure 25. The tiling has not been extended to infinity in all directions. Only the part necessary for the extrusion is shown. The design primitive created is then transformed by the current normalization transformation.

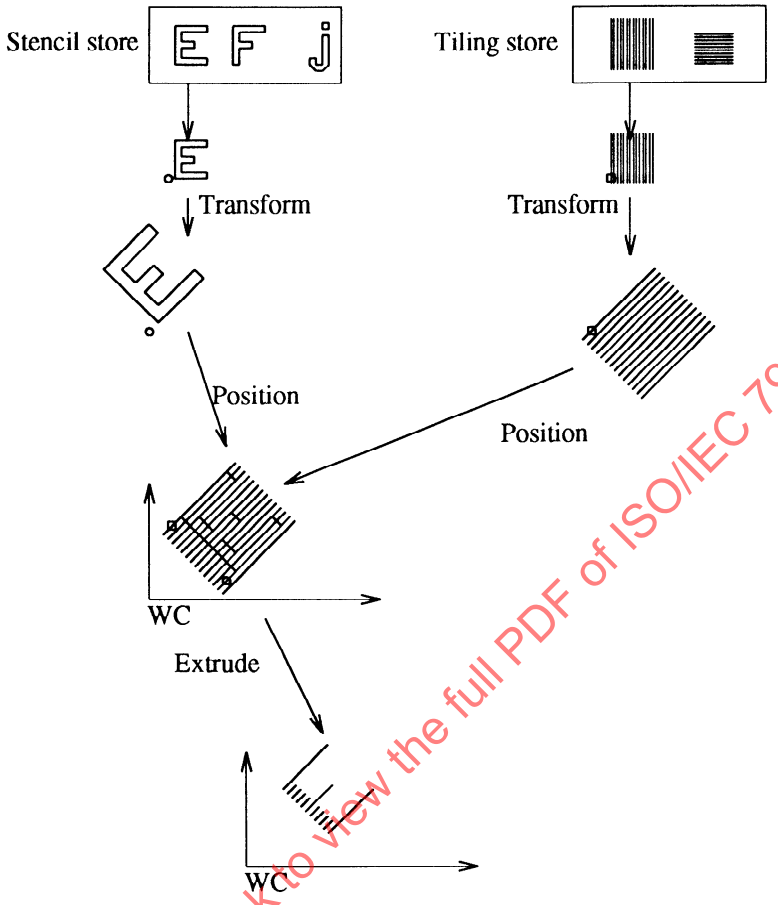

Figure 25 - Stencil store and tiling store

8.7.2 Stencils

A *stencil* is a set of shapes used to define a primitive in the design class. Stencils can be constructed in three ways:

a) *From a sequence of boundary definitions*: a stencil is defined which consists of a set of areas defined by the boundary definitions. Each boundary definition defines a *closed path*.

b) *From a path contour*: a stencil is defined corresponding to the closed path specified by the contour of a path such as a sequence of line segments.

c) *By concatenation*: a stencil can be created by concatenating instances of previously defined stencils. The relative positioning of the instances can be controlled.

Closed paths can be defined using a combination of polylines, nurbs and conic sections.

Stencils have an associated inside rule (EVENODD or WINDING) which is used to determine the interior of the stencil. The EVENODD rule is defined in 8.4.2. For the WINDING rule, a point is filled according to the following rule. Draw an imaginary line from the point to any vertex of the polygon. Rotate the line about the point so it intersects each vertex of the polygon, in order, until the line returns to the original position. If the line makes one or more complete rotations, the point is inside.

The function CREATE STENCIL FROM BOUNDARY creates a named stencil from a sequence of boundaries. A boundary is either an area shape or a sequence of paths. A sequence of *paths* is implicitly closed. If

the end point of one path is not equal to the start point of the next, a single line path is added to link the two points. If the start point of the first path is not equal to the end point of the last path, then a line is added to link the two points and hence complete the closure of the boundary definition.

An area shape, for example an ellipse, is intrinsically closed.

The function CREATE STENCIL FROM CONTOURS defines a stencil as the set of areas defined by the *contours* (see figure 26) surrounding a sequence of paths. The precise form of the stencil created depends upon the attributes associated with the sequence of paths. The contour attributes are:

d) *STYLE*: defines the style to be applied to a path consisting of a sequence of curves. The possible styles are SOLID, DASHED, DOTTED, DASHED-DOTTED, DASHED-DOTTED-DOTTED.

e) *WIDTH*: defines the width of the contours created. The width is defined in NDC space; a utility function is provided to transform a WC ordinate value into NDC (EVALUATE WC ORDINATE).

f) *CAP*: defines the style of the two ends of a path assuming it is not closed. Defined CAP types are BUTTED, SQUARE and ROUNDED.

g) *JOIN*: for a path consisting of a sequence of lines, this attribute specifies the join between line segments. The JOIN types defined are: ROUND, BEVEL and MITRED (mitre is defined to limit spikes between two lines with a small angle between them).

Contour attributes are defined by the SET CONTOUR ATTRIBUTE function.

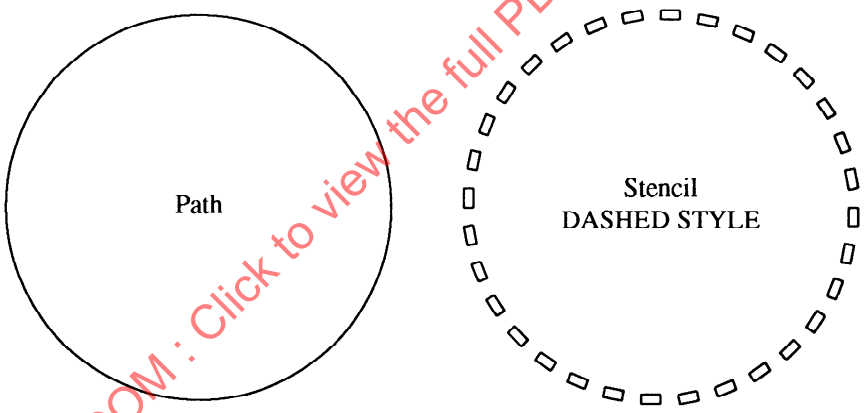

Figure 26 - Creating stencil from contours

To construct a stencil from other stencils, the function OPEN STENCIL is invoked. The name of the stencil to be created is supplied as a parameter and is recorded as the name of the open stencil in the GKS state list. Invocation of the function CLOSE STENCIL completes the definition of the stencil.

Between invocations of OPEN STENCIL and CLOSE STENCIL, the function INSTANCE STENCIL can be invoked, which creates an instance of a previously defined stencil. This function is described in 8.7.4. Names given to instances of stencils created in this context are treated as transient names. These transient names can be used by the function INSTANCE STENCIL to concatenate stencils with instances created by previous invocations of the INSTANCE STENCIL function. The names disappear when CLOSE STENCIL is invoked.

8.7.3 Stencil attributes

Each stencil has a number of associated attributes, which attach specific names to points or ordinate values within the stencil. The stencil attributes are defined implicitly when a stencil is created, but can be defined explicitly or modified by the function SET STENCIL ATTRIBUTE.

There are two categories of stencil attributes, those which define ordinate values and those which define coordinate positions. The stencil attributes defining ordinate values are:

a) TOPY, CAPY, HALFY, BASEY, BOTTOMY, CENTREY, LEFTX, RIGHTX, CENTREX

The stencil attributes defining coordinate positions are:

b) CENTRE, ORIGIN, CENTRETOP, CENTREBOTTOM, CENTRELEFT, CENTRERIGHT, TOPLEFT, TOPRIGHT, BOTTOMLEFT, BOTTOMRIGHT

Typical stencil attribute values are shown in figures 27 and 28.

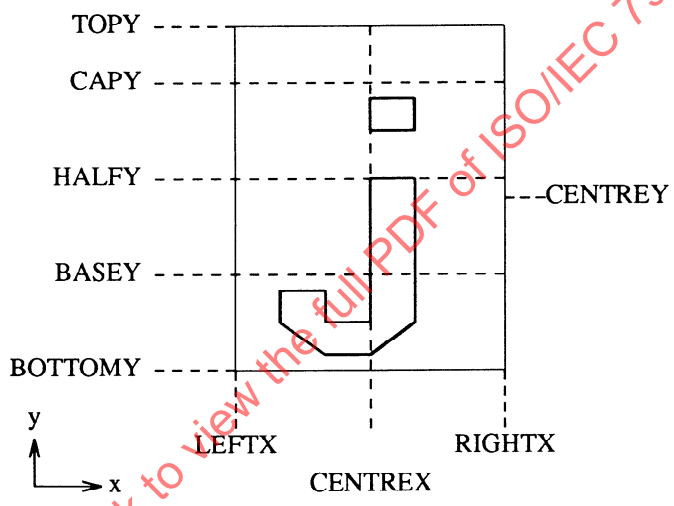

Figure 27 - Typical stencil attribute ordinate values

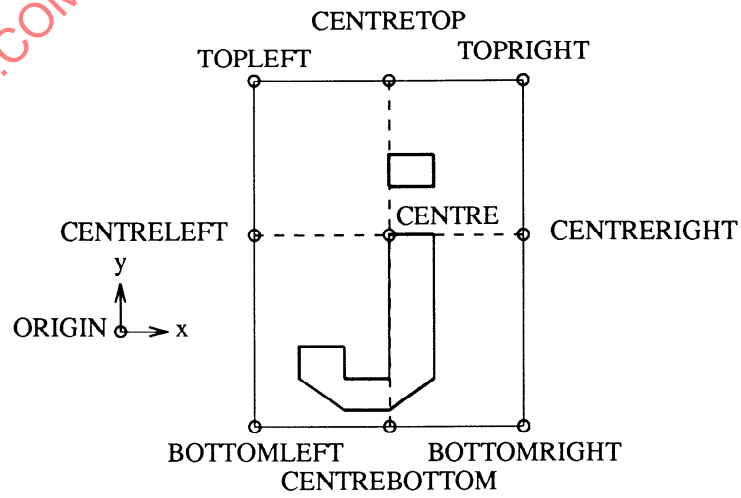

Figure 28 - Typical stencil attribute coordinate positions

Implicit values of stencil attributes are computed from the bounding box of the stencil. The bounding box is the smallest rectangle which completely encloses the stencil whilst maintaining its sides parallel to the x and y axes in the local coordinate system as illustrated in figure 29.

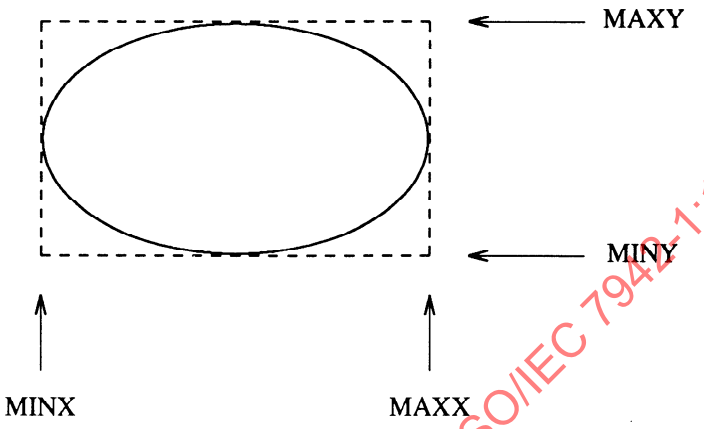

Figure 29 - Creating the bounding box

If the two points (MINX, MINY), (MAXX, MAXY) define the bounding box, the default values of the stencil attributes are given by:

TOPY	= MAXY
CAPY	= MAXY
HALFY	= 0.5 * (MAXY + MINY)
BASEY	= MINY
BOTTOMY	= MINY
CENTREY	= 0.5 * (MAXY + MINY)
LEFTX	= MINX
RIGHTX	= MAXX
CENTREX	= 0.5 * (MAXX + MINX)
CENTRE	= (CENTREX, CENTREY)
ORIGIN	= (0, 0)

When stencils are created by concatenation, invocation of the function CLOSE STENCIL causes values to be assigned to any undefined attributes of the open stencil. The missing values are computed using the equations above.

The values of stencil attributes can be inquired by the function GET STENCIL ATTRIBUTE which returns the value of a specified stencil attribute, if the value is defined and an error indicator otherwise.

8.7.4 Stencil composition

The function INSTANCE STENCIL is used to create an instance of a previously defined stencil, within the open stencil. The position and alignment of the instance with respect to the open stencil may be defined in one of three ways:

a) *PUT*: the instanced stencil is translated such that a reference point on the instanced stencil is coincident with a reference point on the open stencil;

b) *ALIGN*: the instanced stencil is rotated and translated such that a reference point on the instanced stencil is coincident with a reference point on the open stencil and a direction vector on the instanced stencil coincides with a direction vector on the open stencil.

c) *MAP*: the instanced stencil is transformed such that three positions on the instanced stencil are coincident with three positions on the open stencil.

In each case, a stencil transformation specified by a 2×3 matrix is applied to the stencil to be instanced before instantiation.

Figure 30 gives an example of PUT. The circle denotes the reference point on the instanced stencil while the square denotes the reference point on the open stencil. The stencil transformation applied is a 90° anti-clockwise rotation.

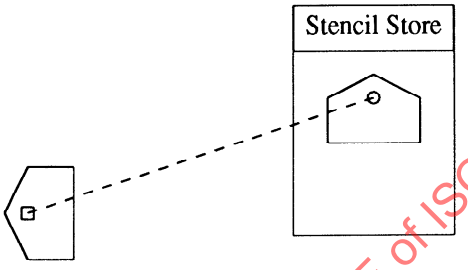

Figure 30 - An example of PUT

Figure 31 gives an example of ALIGN. The reference point on the instanced stencil is denoted by a circle and the dashed arrow from the circle denotes the direction vector associated with it. The square denotes the reference point in the open stencil and the dashed arrow from the square denotes the direction vector. In this case, the stencil transformation matrix is identity. The 90° anti-clockwise rotation is caused by the different directions of the direction vectors.

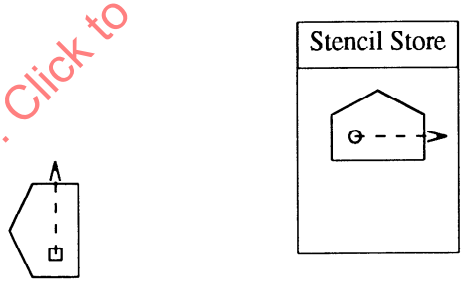

Figure 31 - An example of ALIGN

Figure 32 gives an example of MAP. The three positions on the instanced stencil are denoted by a circle, a square and a triangle. The 3 points in the open stencil that they are mapped to are denoted in the same way. Here, the rotation and expansion are caused by the relative positioning of the three points. The stencil transformation matrix is identity.

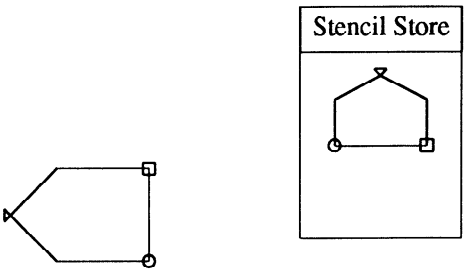

Figure 32 - An example of MAP

The function INSTANCE STENCIL ALONG PATH enables a number of instances of a single stencil to be placed precisely along a supplied path.

The placement of the stencil is specified by a placement line which is defined by two points, start_map and end_map. The placement line is the straight line joining the start_map and end_map points. This line defines both the portion of the stencil to be mapped and its orientation when it is instanced along the supplied path.

The instance number specifies the number of instances of the stencil which should be placed along the supplied path. The placement procedure will proceed until the end of the supplied path is reached or the specified number of instances has been placed. If the instance number specified is zero then the stencil will be instanced until the end of the supplied path is reached.

The stencil is instanced along the supplied path so that the placement line, specified by the start_map and end_map points, lies along the supplied path. All portions of the stencil which lie on a line orthogonal to the placement line are mapped so they still lie the same distance along a line orthogonal from the mapped placement line. It is permissible to approximate the supplied path by a sequence of chords and to instantiate the stencil along the chord sequence.

In figure 33 the stencil 'dashed line' is instanced along the nurb path specified.

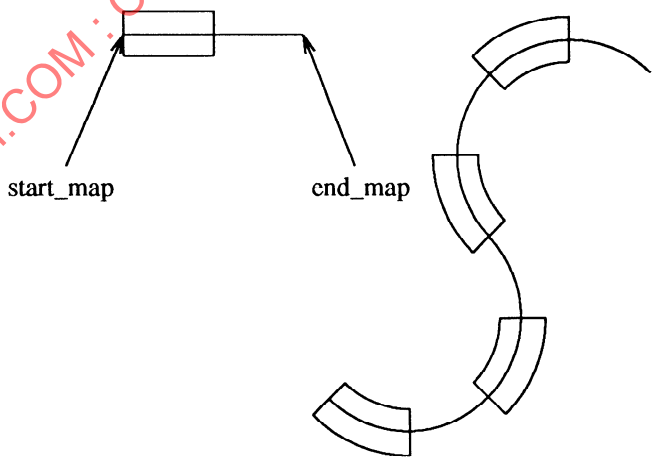

Figure 33 - Instancing the 'dashed line' stencil

Figure 34 illustrates how a point on the stencil which lies on the line orthogonal to the placement line is mapped so it remains the same distance along the line orthogonal to the mapped placement line.

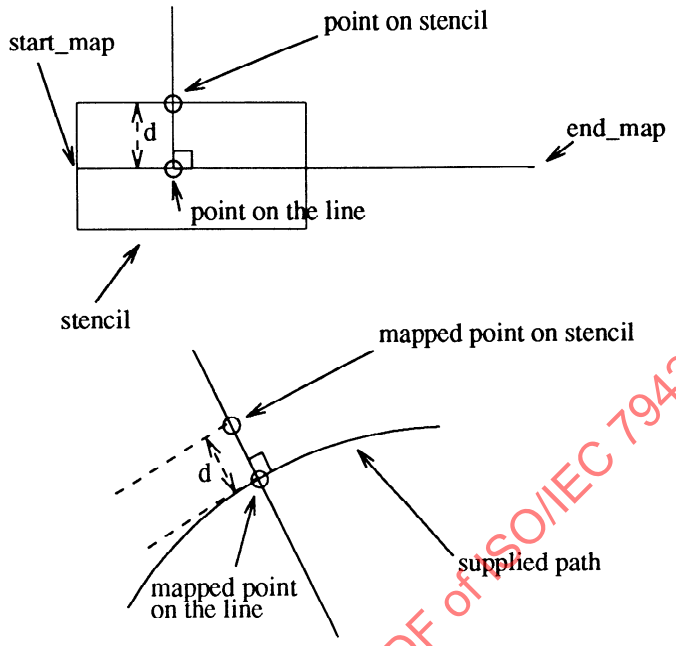

Figure 34 - Mapping a point on the stencil

The function INSTANCE STENCIL SEQUENCE ALONG PATH enables a sequence of stencils to be instanced within the open stencil at positions along a path defined in the local coordinate system of the open stencil. Each stencil in the sequence has an associated reference point, stencil transformation matrix and stencil name to be given to the instance created. Instanced stencils are placed along the path as follows:

(1) the first instanced stencil is placed such that its reference point is coincident with a reference point specified on the path;

(2) the intersection between the path and the edge of the instanced stencil defined by the abutting specifier is computed;

(3) the next instanced stencil is placed such that its reference point is coincident with the intersection point just computed;

(4) steps (2) and (3) are repeated until all the instanced stencils have been placed on the path. An error is generated if the end of the path is reached before all the instanced stencils have been placed.

There are eight abutting specifiers available. These are listed below along with the edge of the stencil they define.

d) RIGHT: defines the RIGHTX edge;

e) LEFT: defines the LEFTX edge;

f) TOP: defines the TOPY edge;

g) BOTTOM: defines the BOTTOMY edge;

h) CENTREVERT: defines the CENTREX edge;

i) CENTREHORIZ: defines the CENTREY edge;

j) CAPLINE: defines the CAPX edge;

k) BASELINE: defines the BASEY edge.

Figure 35 gives an example of INSTANCE STENCIL SEQUENCE ALONG PATH. The stencils to be instanced are the letters E, F and T. The forms are outlines. Each stencil has an associated reference point denoted by a circle. The abut specifier has been defined as the right hand edge. In this example, the path is shown as a dashed straight line, it could be a more general curve. The positions of the stencils on the path are shown.

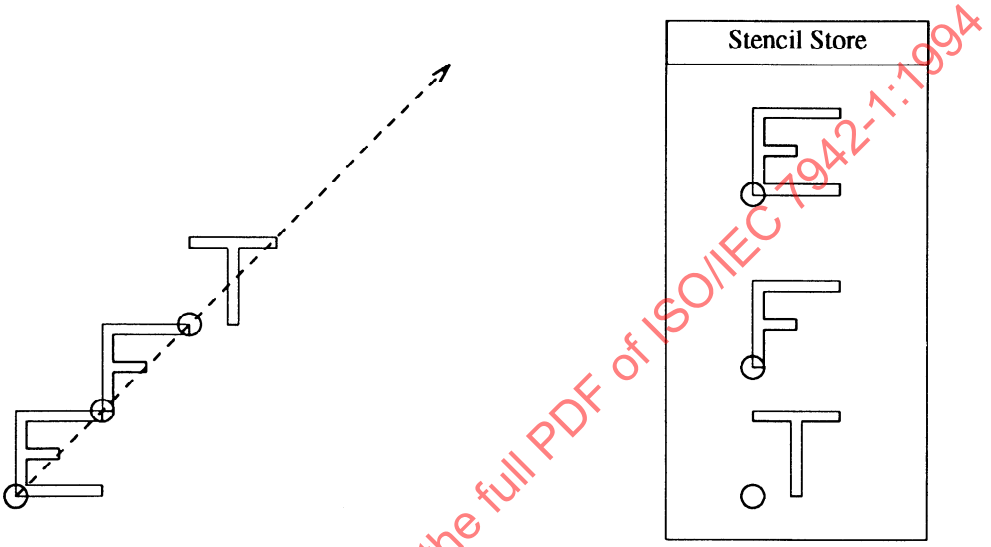

Figure 35 - INSTANCE STENCIL SEQUENCE ALONG PATH

8.7.5 Tiling store

All *tilings* are stored in the *tiling store* and are defined in their own local coordinate system. Tilings conceptually have an unlimited extent i.e., they extend to infinity in all dimensions. When a design primitive is created, the extent of a tiling is fixed by extruding it through a stencil into the NDC picture with any parts of the tiling outside the stencil not passing into the NDC picture. The inside of a stencil is defined by the inside rule specified when the stencil is created.

A tiling is created as a sequence of *tiling components* by invoking the function OPEN TILING and supplying the name of the tiling. A number of CREATE TILING COMPONENT functions can then be invoked to specify *tiles* and their replications across infinite space. The tile to be replicated may be specified similarly to output primitives of the following classes:

a) CURVE

b) AREA

c) DESIGN

The origin parameter to the CREATE TILING COMPONENT function defines the origin for the replication technique. The origin and the replication technique define a lattice of points across infinite space. The CURVE, AREA or DESIGN tile define a prototile in a local coordinate system. The prototile is laid down at each of the positions in the lattice such that one of them has its origin coinciding with the origin of this application technique. The following replication techniques may be used:

d) DX: This technique offsets the tile by a fixed amount in the X direction across infinite space. Starting from minus infinity, replication occurs in order to plus infinity in the X direction.

e) DY: This technique offsets the tile by a fixed amount in the Y direction across infinite space. Starting from minus infinity, replication occurs in order to plus infinity in the Y direction.

f) DXDY: This technique offsets the tile by a fixed amount in the X and Y directions across infinite space. Starting from minus infinity in the Y direction, the tile is incremented by a fixed amount in the X direction from minus infinity to plus infinity. This process is repeated with Y being offset by a fixed amount until it also goes from minus infinity to plus infinity.

g) DYDX: This is similar to DXDY but tiles are drawn initially in the Y direction rather than the X direction.

Consider the following design in figure 36, 'brick-part'.

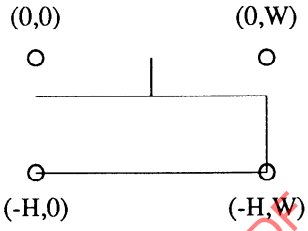

Figure 36 - The 'brick-part'

This is turned into a tiling by specifying the replication technique DXDY and the following parameters to the CREATE TILING COMPONENT function:

origin = (LX, LY)
reptech = dxdy <<(W, H)>>

The resulting tiling is shown in figure 37.

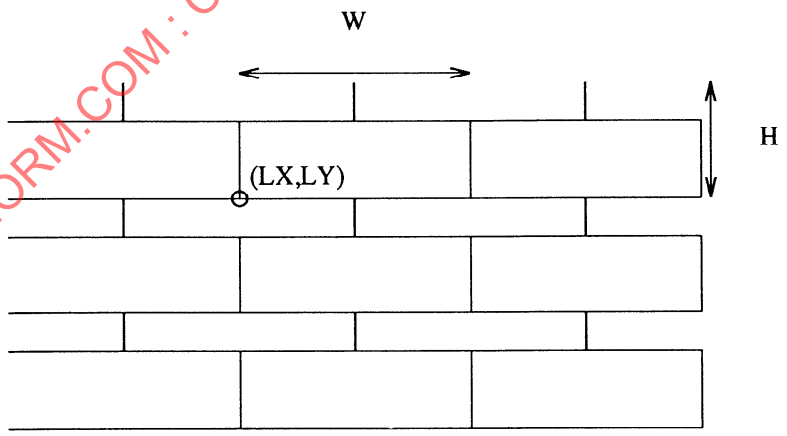

Figure 37 - The resulting tiling

Invoking the function CLOSE TILING completes the definition of the tiling.

8.7.6 Stencil and tiling functions

Functions exist to delete and rename stencils and tilings. The function DELETE STENCIL deletes the stencil from the stencil store and makes the name of the stencil available for reuse. The function RENAME STENCIL changes the name of the stencil to a new name and makes the old name of the stencil available for reuse.

The functions DELETE TILING and RENAME TILING provide similar operations to tilings that exist in the tiling store.

Deleting or renaming stencils and tilings has no effect on previous uses of the stencil or tiling. Functions such as INSTANCE STENCIL, INSTANCE STENCIL ALONG PATH, INSTANCE STENCIL SEQUENCE ALONG PATH and CREATE OUTPUT PRIMITIVE are defined as having their parameters 'called by value'. Any stencil or tiling used in such functions accesses the current version of the stencil or tiling incorporating it within the stencil or output primitive being created. Subsequent changes to either a stencil or tiling will not effect the stencil or output primitive previously created.

8.8 Generalized drawing primitive

The GENERALIZED DRAWING PRIMITIVE (GDP) primitive does not belong to any particular class. Instead it provides a means of defining composite output primitives that may use the attributes of any of the primitive classes. If used by a GDP, the attributes of the other primitives behave in the same way as for the primitive class itself. It is provided to address special capabilities of a workstation inaccessible from the standard set of output primitives. It is characterized by an identifier, a set of points and additional data. GKS applies all transformations to the points but leaves the interpretation to the workstation.

9 Logical input device classes

9.1 Introduction

Clause 6 contains a description of the overall capabilities of the logical input devices. This clause gives details of the logical input device classes.

9.2 Measures of each logical input device class

Details of the measures of logical input devices of different classes are as follows.

A LOCATOR measure consists of a position in world coordinates (P) and a normalization transformation number (N). Then if N≠0, P transformed to NDC (P^{NDC}) by the normalization transformation number N lies within the viewport specified by N and outside all viewports of higher priority than N. If N=0, P is transformed to NDC (P^{NDC}) by the identity transformation and (P^{NDC}) lies outside all viewports of higher priority than the viewport of normalization transformation 0. P^{NDC} is then transformed to LDC (P^{LDC}) by view transformation V. P^{LDC} lies within the scissor region associated with V and outside the scissor regions of all view transformations of higher priority than V. P^{LDC} also lies within the workstation window.

A STROKE measure consists of a sequence of points in world coordinates and a normalization transformation number (N). Let $P_1 \ldots P_m$ be the points. Then if N≠0, P_i ($1 \le i \le m$) transformed to NDC by the normalization transformation number N (P_i^{NDC}, $1 \le i \le m$) lie within the viewport specified by N and there is no viewport, N_j, of higher priority than N containing all the points P_i transformed to NDC by N_j. If N=0, the P_i ($1 \le i \le m$) are transformed to NDC by the identity transformation and there is no viewport, N_j, of higher priority than the viewport of normalization transformation 0 containing all the points P_i transformed to NDC by N_j. N may change as points are added to the stroke. (P_i^{NDC}, $1 \le i \le m$) are then transformed to LDC by view transformation V. (P_i^{LDC}, $1 \le i \le m$) all lie within the scissor region associated with V and outside the scissor regions of all view transformations of higher priority than V. The viewing transformation applied may change as points are added to the stroke. After viewing, the transformed points P_i^{LDC} lie within the workstation window.

Any invocation of SET WINDOW AND VIEWPORT, SET VIEWPORT INPUT PRIORITY, SET VIEW or SET VIEW PRIORITY may cause a change in P (any P_i for STROKE) or N or both, but the above conditions hold for the new values.

The rules imply that no normalization transformation having priority less than that of transformation 0 can appear in the state of a LOCATOR or STROKE measure process (with the default settings of the viewport input priorities, normalization transformation 0 has the highest).

A VALUATOR measure provides logical input values that are real numbers. Each value lies between (possibly including) minimum and maximum values, which are stored in the data record in the workstation state list.

A CHOICE measure provides logical input values whose components are OK or NOCHOICE and an integer in the range 1 to a device dependent maximum. If the first component is OK, then the integer is valid. CHOICE input typically occurs when an operator presses a button (the numeric identifier of the button determines the measure) or combination of buttons (the measure is derived from the combination of buttons pressed).

A PICK measure provides logical input values whose components are OK or NOPICK, NAMESET and a PICK IDENTIFIER. If the first component is OK, then the NAMESET and PICK IDENTIFIER obey the following rules:

a) The output primitive picked has been selected for display and is detectable on the workstation.

b) The NAMESET and PICK IDENTIFIER are the NAMESET and PICK IDENTIFIER attributes of the output primitive picked. Part of the primitive lies within the workstation window and, if clipping was set to CLIP, part also lies within the primitive's set of clipping rectangles and, if shielding was set to SHIELD, part also lies outside the primitive's set of shielding rectangles.

The PICK initial value is tested against the above rules whenever the PICK measure process is initiated. If the rules are not satisfied, the process state is set to NOPICK.

A STRING measure provides logical input values which are character strings up to a device dependent maximum length specified by the buffer size value in the data record (see 7.13.2).

A COMPOSITE measure provides logical input values which are compound values built up from the above basic values. The construction of logical input devices of class COMPOSITE is described in 7.13.3.

9.3 Transformation of LOCATOR and STROKE input

9.3.1 Transformation of LOCATOR input

The application programmer requires LOCATOR input to define a position in the most appropriate world coordinate system currently defined by the set of normalization transformations. This is achieved by first transforming the input data from DC to LDC by the inverse workstation transformation which is in effect when LOCATOR input is generated. LOCATOR input can only be obtained from positions within the part of the current workstation viewport into which the current workstation window is mapped (note that this is a subset of the workstation viewport whenever the aspect ratio of the workstation viewport and workstation window differ). Thus, LOCATOR input always defines a position in the LDC range $[0,1] \times [0,1]$.

The position in LDC space needs to have an inverse view transformation applied to it to generate the NDC position relevant to the NDC picture. Each viewing transformation has associated with it a view priority which is only relevant to LOCATOR, STROKE and possibly COMPOSITE input. When a workstation is opened, the view input priorities are set relative to view 0. View 0 is given the highest priority, view 1 the next highest priority and so on. These can be changed at any time by SET VIEW PRIORITY. The inverse viewing transformation applied is the one associated with the view of highest priority whose associated view scissor region contains the points.

To return to the application program a position in WC, the position in NDC space needs to be transformed from NDC to WC by the inverse of one of the normalization transformations. Each normalization transformation has associated with it a viewport input priority which is only relevant to LOCATOR and STROKE input and possibly COMPOSITE input. When GKS is opened, the viewport input priorities are set relative to normalization transformation number 0. Normalization transformation number 0 is given the highest priority, normalization transformation number 1 the next highest priority and so on. These can be changed at any time by SET VIEWPORT INPUT PRIORITY.

The LOCATOR input position in NDC space is compared with the viewports of the normalization transformations, to find the normalization transformation with the viewport which has the highest viewport input priority and contains the LOCATOR position. The LOCATOR position is transformed by the inverse of this normalization transformation to the associated WC position. This LOCATOR position is returned to the application program in WC together with the number of the normalization transformation used.

An example of LOCATOR input is given in figure 38. The output on the display has been constructed using four normalization transformations 0, 1, 2 and 3. Normalization transformation 0 is the identity transformation. The normalization transformations 2 and 3 have the aspect ratio changed in the transformation from WC to NDC coordinates.

Three views (0,1 and 2) of the NDC picture are displayed on the workstation. The selection criteria for view 0 has been set to REJECTALL so the display consists of the appearance of the NDC picture defined by view 1 and view 2. In consequence, view 0 has not been shown. View 1 is a straightforward view of the top half of the NDC picture. View 2 is rotated by 90° and halved in size. The graphical output in views 1 and 2 is displayed on the workstation. The view scissor in each case is set to the clipping rectangle shown.

The view priorities are set so that view 2 has higher view priority than view 1 which has higher view priority than view 0. The normalization transformations are set so that the viewport input priority of normalization transformation 1 is higher than 2 which is higher than 3 which is higher than 0.

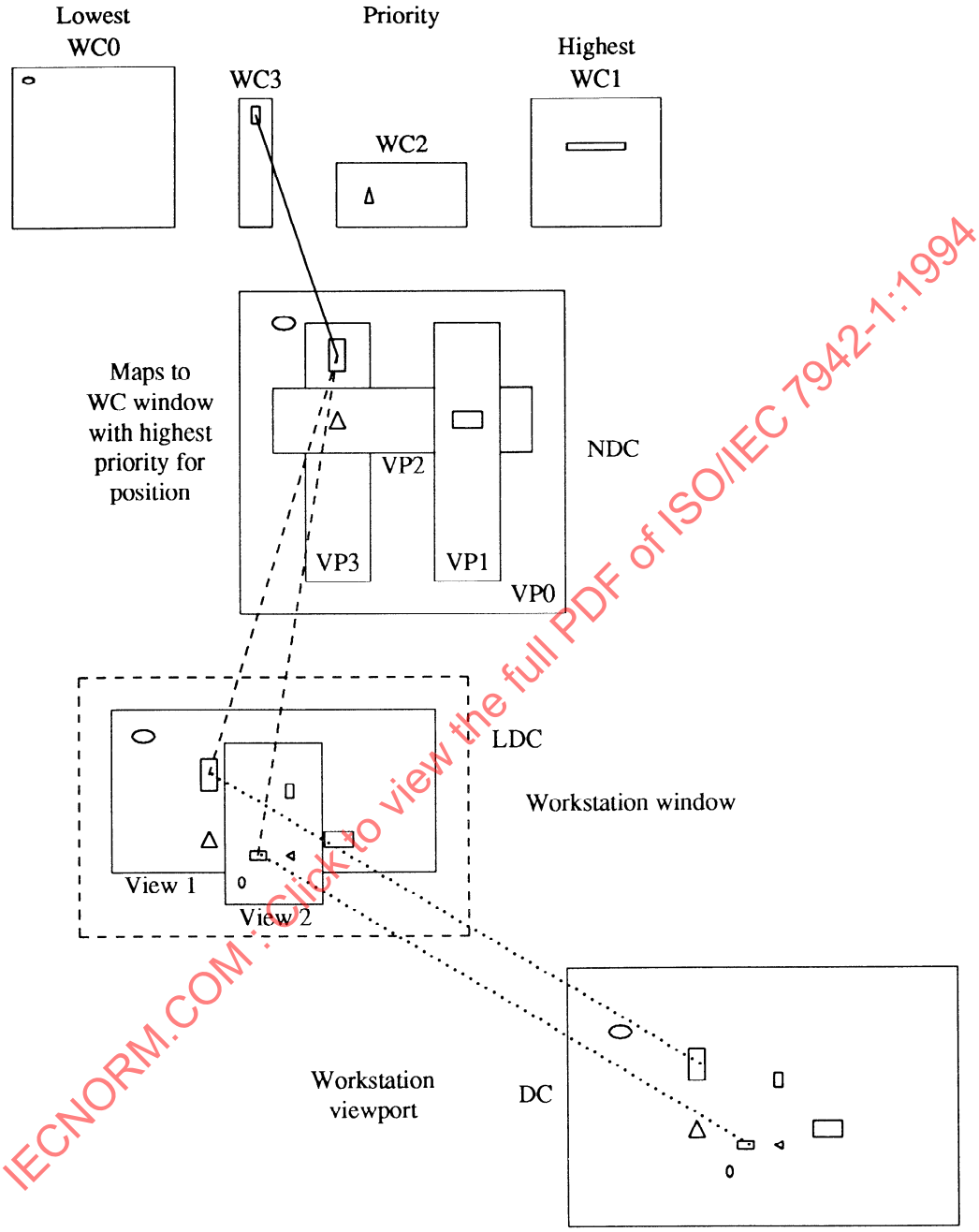

Figure 38 - LOCATOR input

The figure shows the way input in the two rectangular areas of DC space are transformed back to deliver the LOCATOR measure. First, the points are mapped from the workstation viewport in DC to the workstation window in LDC. The smaller rectangle is within the clipping boundary of view 2 so this is the view used to perform the inverse view mapping back to a point in NDC space. Note there is a 90° rotation. The larger rectangle is not within the clipping boundary of view 2. It is within the clipping boundary of view 1 so view 1 is used to perform the inverse view mapping back to a point in NDC space. Note the points within this larger rectangle map to the same set of points as the ones mapped from the smaller rectangle by view 2. Finally, the

transformed points in the NDC space are within viewport 3 but not within viewports 2 and 1, the normalization transformations with higher viewport input priority. Consequently, the inverse transformation performed is the one associated with normalization transformation number 3.

Points in the other areas indicated in the NDC picture are transformed back in a similar way. The figure indicates the areas in LCD, NDC and WC space that they will be mapped into in order to generate the LOCATOR measure.

As normalization transformation number 0 is given the highest viewport input priority initially, LOCATOR input is effectively returned in WC equivalent to NDC until a normalization transformation is defined with a viewport input priority greater than that of normalization transformation number 0. If a normalization transformation is no longer required for mapping LOCATOR input back to WC, it can effectively be hidden by reassigning it a viewport input priority lower than normalization transformation number 0.

Changing the viewport input priority of normalization transformation number 0 is allowed.

In an event report, generated by a LOCATOR device in EVENT mode, the DC position is transformed to the appropriate WC position before the event report is placed on the input queue. These transformations may be performed while the normalization transformation, views and workstation transformations are being changed; thus, there is a race condition. The implementation has therefore to treat the transformations as resources to be allocated and deallocated between the competing processes.

9.3.2 Transformation of STROKE input

Similar considerations apply to transformation of STROKE input as apply to LOCATOR input, with the complication that more than one point is involved.

When each point of a stroke is generated, the coordinates of the point are transformed from DC to LDC by the inverse workstation transformation then in effect (see 7.6). STROKE input always consists of points in the LDC range $[0,1] \times [0,1]$. An inverse view transformation is then selected to transform the points of the stroke from LDC to NDC. The transformation is selected in an analogous manner to LOCATOR input with the additional restriction that all the points of the stroke lie in the scissor region of the view selected.

The STROKE points in NDC space are compared with the viewports of the normalization transformations, to find the normalization transformation with the viewport which has the highest viewport input priority and contains all of the points. The STROKE points are then transformed by the inverse of this normalization transformation and returned to the application program in WC together with the number of the normalization transformation used. If the STROKE device is in SAMPLE mode, the normalization transformation used can vary between successive samples.

In EVENT mode, there is a similar race condition to that applying to LOCATOR input. Between placing an event report on the input queue and the execution of AWAIT INPUT which removes the STROKE event from the queue, it is possible for the normalization transformation, views and the workstation transformation to be changed by the application program. To ensure that the DC points input by the operator are equivalent to the WC points retrieved from the input queue, it is advisable for the application program not to change transformations while a STROKE device is in EVENT mode.

9.4 Prompt and echo types

9.4.1 LOCATOR prompt and echo types

Prompt and echo types for LOCATOR logical input devices are:

<0 prompting and echoing is LOCATOR device dependent.

1 designate the current position of the LOCATOR device using an implementation-defined technique.

2 crosshair, i.e. designate the current position of the LOCATOR device using a vertical line and a horizontal line spanning over the display surface or the workstation viewport intersecting at the current LOCATOR position.

3 designate the current position of the LOCATOR device using a tracking cross.

4 designate the current position of the LOCATOR device using a rubber band line connecting the initial locator position defined in the workstation state list and the current LOCATOR position.

5 designate the current position of the LOCATOR device using a rectangle. The diagonal of the rectangle is the line connecting the initial locator position defined in the workstation state list and the current LOCATOR position.

6 display a digital representation of the current position of the LOCATOR device in LOCATOR device dependent coordinates within the echo area.

≥7 reserved for registration.

9.4.2 STROKE prompt and echo types

Prompt and echo types for STROKE logical input devices are:

<0 prompting and echoing is STROKE device dependent.

1 display the current stroke using an implementation defined technique.

2 display a digital representation of the current position of the STROKE device within the echo area.

3 display a marker at each point of the current stroke.

4 display a line joining successive points of the current stroke.

≥5 reserved for registration.

If the operator enters more points than the current input buffer size, the additional points are lost.

STROKE data record entries for variables such as intervals in X, Y and time may be provided to constrain the number of points delivered.

For all prompt and echo types, the first entry in the STROKE data record is the input buffer size which is an integer in the range (1..n). This is compared against an implementation defined maximum input buffer size for this device. If the requested buffer size is greater, the maximum input buffer size is substituted in the stored data record. If the initial stroke specified in the initial value is longer than the buffer size, an error is issued.

When a STROKE measure process comes into existence, it obtains a buffer of the current input buffer size. The initial stroke is copied into the buffer, and the editing position is placed at the initial buffer editing position within it. Replacement of points begins at this initial position. If the initial buffer editing position cannot be specified in the STROKE data record, the value 1 is used.

9.4.3 VALUATOR prompt and echo types

Prompt and echo types for VALUATOR logical input devices are:

<0 prompting and echoing is VALUATOR device dependent.

1 designate the current VALUATOR value using an implementation defined technique.

2 display a graphical representation of the current VALUATOR value within the echo area (for example, a dial or a pointer).

3 display a digital representation of the current VALUATOR value within the echo area.

≥4 reserved for registration.

For all VALUATOR prompt and echo types, the VALUATOR data record includes, in the first two positions, a low value and a high value, in that order, specifying the range. The values from the device will be scaled linearly to the specified range.

9.4.4 CHOICE prompt and echo types

Prompt and echo types for CHOICE logical input devices are:

<0 prompting and echoing is CHOICE device dependent.

1 designate the current CHOICE number using an implementation defined technique.

2 the physical input devices that are most commonly used to implement a CHOICE logical input device normally have a built-in prompting capability. This prompt and echo type allows the application program to invoke this prompting capability. If the value of the i-th element of 'prompt array' in the CHOICE data record is OFF, prompting of the i-th alternative of the specified choice input device is turned off. An ON value indicates that prompting for that alternative is turned on. The first entry in the CHOICE data record is the number of CHOICE alternatives. This is compared against an implementation defined maximum number of CHOICE alternatives for this device. The second entry in the CHOICE data record is the 'prompt array'.

3 allow the operator to indicate a CHOICE number by selecting, using an appropriate technique, one of a set of CHOICE strings. The CHOICE strings are contained in the CHOICE data record and are displayed within the echo area. The logical input value is the number of the string selected. The first entry in the CHOICE data record is the number of CHOICE strings. This is compared against an implementation defined maximum number of CHOICE alternatives for this device. The second entry in the CHOICE data record is the 'array of choice strings'.

4 allow the operator to indicate a CHOICE number by selecting, via an alphanumeric keyboard, one of a set of CHOICE strings. The CHOICE strings are contained in the CHOICE data record and may be displayed in the echo area as a prompt. The string typed in by the operator is echoed in the echo area. The logical input value is the number of the string that has been typed in by the operator. The first entry in the CHOICE data record is the number of CHOICE strings. This is compared against an implementation defined maximum number of CHOICE alternatives for this device. The second entry in the CHOICE data record is the 'array of choice strings'.

5 the picture part named by the CHOICE data record is interpreted during execution of the INITIALIZE LOGICAL INPUT DEVICE function for later use as a prompt of the specified CHOICE device. It will be displayed within the echo area by mapping the unit square $[0,1] \times [0,1]$ of NDC space onto the echo area. The PICK IDENTIFIERs in the picture part are mapped to CHOICE numbers in a CHOICE device dependent fashion. Picking these primitives selects the corresponding CHOICE value. After the interpretation, no logical connection between the specified picture part and the specified CHOICE device exists. The first entry in the CHOICE data record is the picture part name.

≥6 reserved for registration.

9.4.5 PICK prompt and echo types

Prompt and echo types for PICK logical input devices are:

<0 prompting and echoing is PICK device dependent.

1 use an implementation-defined technique that at least highlights the 'picked' primitive for a short period of time.

2 echo the contiguous group of primitives within the NDC picture with the same PICK IDENTIFIER and NAMESET as the 'picked' primitive, or all primitives of the NDC picture with the same PICK IDENTIFIER and NAMESET as the 'picked' primitive.

3 echo all the primitives in the NDC picture with the same NAMESET as the 'picked' primitive.

≥4 reserved for registration.

9.4.6 STRING prompt and echo types

Prompt and echo types for STRING logical input devices:

<0 prompting and echoing is STRING device dependent.

1 display the current STRING value within the echo area.

≥2 reserved for registration.

For all prompt and echo types, the first entry of the STRING data record is the input buffer size, which is an integer in the range (1..n). This is compared against an implementation defined maximum input buffer size for this device. If the requested buffer size is greater, the maximum input buffer size is substituted in the stored record.

For all prompt and echo types, the second entry of the STRING data record is an initial cursor position, which may range from 1 to the length of the initial string plus 1.

When a STRING measure process comes into existence, it obtains a buffer of the current input buffer size. The initial string is copied into the buffer, and the cursor is placed at the initial cursor position within it. Replacement of characters begins at this cursor position.

10 Segments and workstation activation

10.1 Introduction

The combination of picture part store, namesets and selection criteria gives a very powerful means of structuring graphical data and controlling its appearance on workstations. The functions for segments and workstation activation provide a particular example of how these facilities can be used. A segment allows a group of primitives to be manipulated as a whole. Each segment has a unique name, assigned by the application. The *mapped segment name* is a name derived from the segment name that can be used either as a name in a primitive's nameset or the name of a picture part. A utility function GET MAPPED SEGMENT NAME given a segment name returns the mapped segment name. The GLOBAL TRANSFORMATION attribute of the primitives in the segment are all identical and are defined as an attribute of the segment. A segment is stored as a picture part without reference to its associated workstations. The primitives within a segment are also stored in the NDC picture, with namesets that include both the mapped segment name and the mapped workstation identifiers of associated workstations.

Segments and workstation activation provide a mechanism for controlling visibility, highlighting and detectability on a global basis by defining them as segment aspects. The *mapped workstation identifier* is a name derived from the workstation identifier that can be used together with the mapped segment name in a primitive's nameset to control display, highlighting and detectability. The mapped segment names and mapped workstation identifiers are distinct from names in namesets or picture part names defined in the standard way. A utility function GET MAPPED WORKSTATION IDENTIFIER given a workstation identifier returns the mapped workstation identifier. Finally, segments have an attribute *segment priority* which determines the order in which the corresponding picture parts appear in the NDC picture. It provides a convenient mechanism for reordering the primitives in the NDC picture.

10.2 Selection criteria

Three attributes control the appearance and detectability of segments: display, highlighting and detectability. In each case, this is achieved by appropriate setting of selection criteria for each associated workstation. As described in 7.4, one portion of the GKS selection criteria can be supplied by the application and the remainder is manipulated by GKS. The two portions are joined by an OR operator and the ordering of manipulations on the system part is as described below. The default selection criteria for display, highlighting and detectability are REJECTALL. On the first invocation of ACTIVATE WORKSTATION, the display criterion is set to:

s OR CONTAINS (wsid)

where s denotes the selection criterion previously defined and wsid is the mapped workstation identifier. When a segment is created or its visibility attribute is changed to VISIBLE, the display selection criterion, t, of each of the workstations with which the segment is associated is revised to:

t OR CONTAINS (sn, wsid)

where sn is the mapped segment name of the segment. The effect in the case above would be to produce the selection criterion:

(s OR CONTAINS (wsid)) OR CONTAINS (sn, wsid)

Any term in the GKS part of the selection criterion which refers to the segment whose attribute is being changed, is deleted. When a segment is made invisible, the display selection criterion, u, of each of the workstations with which the segment is associated is revised to:

u AND (NOT CONTAINS (sn))

and any existing term which refers to this segment is deleted. Thus for the first case above, the resulting

selection criterion would be:

(s OR CONTAINS (wsid)) AND (NOT CONTAINS (sn))

Changes to the segment attributes for highlighting and detectability are handled in the same manner. On the first invocation of ACTIVATE WORKSTATION, the criteria are not changed in this case so that output primitives outside segments are not highlighted or detectable but are visible.

10.3 Segment state list

Each segment has an associated state list, which holds the current values of its segment attributes:

a) *visibility*: a segment may be visible or invisible;

b) *highlighting*: a segment may be highlighted or normal;

c) *detectability*: a segment may be made detectable or undetectable;

d) *segment transformation*: a transformation matrix which is the GLOBAL TRANSFORMATION attribute of all the primitives in the segment;

e) *segment priority*: an ordering of segments is prescribed which determines the order in which the corresponding picture parts appear in the NDC picture.

The segment state list holds the set of associated workstations, which is the set of workstations active when the segment is created. Workstations cannot be activated or deactivated while a segment is open.

10.4 Workstation activation

The function ACTIVATE WORKSTATION activates a workstation by adding the name of the mapped workstation identifier to the current nameset. On the first invocation of ACTIVATE WORKSTATION, the display selection criterion is set to:

s OR CONTAINS (wsid)

Subsequent invocations add the term OR CONTAINS (wsid) to the display selection criterion. The function DEACTIVATE WORKSTATION removes the name from the current nameset and resets the selection criterion to remove the mapped workstation identifier. Note that under the default selection criterion for display, set when a workstation is opened (see 7.4), primitives outside segments are visible on a workstation if and only if generated while the workstation is active.

10.5 Segment creation

A segment is opened by the function CREATE SEGMENT. While a segment is open, all output primitives dispatched to the NDC picture will contain the mapped segment name in their namesets, will have the GLOBAL TRANSFORMATION attribute set to the segment transformation attribute and will have the workstation selection criteria consistent with the segment attributes for the set of active workstations. Changes to any of the segment attributes will cause the NDC picture and the workstation selection criteria to be updated to reflect these changes (see 10.7).

The function CLOSE SEGMENT closes the open segment. When the segment is closed, it is stored as a picture part removing any segment and workstation names. The current value of the GLOBAL TRANSFORMATION attribute in the GKS state list is reset to identity when the segment is closed.

10.6 Segment manipulation

An existing segment may be renamed by the function RENAME SEGMENT. The corresponding picture part is renamed, and all primitives in the NDC picture related to the segment have their nameset revised. Selection criteria are similarly revised to reflect the name change.

The function DELETE SEGMENT deletes a segment. The corresponding picture part is deleted, and all primitives in the NDC picture related to the segment are deleted. Selection criteria are revised to delete the terms

originating from this segment. A segment may be deleted only from a specific workstation by the function DELETE SEGMENT FROM WORKSTATION. In this case, all primitives in the NDC picture related to the segment have their namesets revised to delete references to the specified workstation. Selection criteria on the specific workstation are revised to delete the terms originating from this segment. The workstation identifier is also removed from the set of associated workstations in the segment state list; if this set becomes empty, then the action of DELETE SEGMENT is carried out.

10.7 Segment attributes

The function SET SEGMENT ATTRIBUTE can be used to set any of the segment attributes. Setting the visibility, highlighting or detectability attributes causes the selection criteria for the set of associated workstations to be revised as described in 10.2.

Segment transformation is specified as a 2×3 matrix. Utility functions (EVALUATE TRANSFORMATION MATRIX and ACCUMULATE TRANSFORMATION MATRIX) are available to assist the application in setting up the transformation matrix. When a segment transformation attribute is changed, the GLOBAL TRANSFORMATION attributes of the related primitives in the NDC picture are changed to the specified segment transformation matrix. The primitives stored in the picture part retain the global transformation matrix defined when the segment is closed (that is the value of segment transformation when the segment is closed).

Segment priority defines the order in which corresponding picture parts appear in the NDC picture. Segments with equal priority appear in the order in which they were created. Segment priority is specified as a real number in the interval [0,1]. The default is zero and hence if segment priorities are not explicitly set, picture parts appear in the order the segments are created.

10.8 Segment storage

A segment is stored as a picture part without reference to its associated workstations. The primitives within the segment are also stored in the NDC picture, with namesets that include both the mapped segment name and the mapped workstation identifiers of associated workstations. Three operations make use of this dual storage.

ASSOCIATE SEGMENT WITH WORKSTATION affects only the primitives stored in the NDC picture. It updates their namesets to include the specified workstation and defines the selection criteria for the workstation appropriately. COPY SEGMENT TO WORKSTATION dispatches the corresponding picture part to the router, with the GLOBAL TRANSFORMATION attribute of primitives equal to the segment transformation attribute, and with the nameset attribute of primitives augmented by the mapped workstation identifier. Note that the primitives are not tagged with the mapped segment name in this case.

INSERT SEGMENT dispatches the picture part to the router. The LOCAL TRANSFORMATION attributes of each of the primitives are replaced by:

$$I \times S \times L$$

where I is the transformation specified as a parameter to the INSERT SEGMENT function, S is the current segment transformation of the segment being inserted and L is the LOCAL TRANSFORMATION attribute associated with each primitive. The GLOBAL TRANSFORMATION attribute of each primitive is replaced by the current value of the GLOBAL TRANSFORMATION attribute in the GKS state list. The SCISSOR SET attribute of each primitive is replaced by the current value of the SCISSOR SET attribute in the GKS state list. The NAMESET attribute of each primitive is augmented by the current nameset.

10.9 Clear workstation

The function CLEAR WORKSTATION removes the mapped workstation identifier from the namesets of all primitives in the NDC picture and deletes the backdrop from the specified workstation.

The effect of the function CLEAR WORKSTATION can be achieved by first changing SET WORKSTATION VISUAL EFFECTS to ALLOW if set to SUPPRESS, invoking the functions REMOVE SET OF NAMES FROM NDC PICTURE and REMOVE BACKDROP, and resetting SET WORKSTATION VISUAL EFFECTS to SUPPRESS if set to this initially.

11 Data types

11.1 Data type definitions

11.1.1 Notational conventions

The heading of each function in clauses 12, 13 and 14 specifies

a) the function's name;

b) references to the function (aligned to the right).

The parameter lists indicate for each entry

c) whether the entry is an input (In) or output (Out) parameter (aligned to the left);

d) the name of the parameter;

e) the data type (aligned to the right).

The definition of the GKS functions in clauses 12, 13 and 14 rely heavily on the data type definitions in this subclause. Abstract data types are constructed from basic types with a number of type constructors. Values of these types are determined by the GKS language binding standard.

The simplest data type definition is exemplified by:

RouteDir == NDC | BACKDROP

This lists the values the type has, in this case NDC and BACKDROP. Values are separated by the symbol '|'.

Data types which consist of sets of values of another type are defined using the powerset constructor, $\boldsymbol{P}$, for example:

NameSet == $\boldsymbol{P}$ GenName

means that values of type NameSet consist of sets of values, each of type GenName.

Ordered tuples are defined using the Cartesian product constructor '×'. For example:

FontPrec == FontInd × Prec

defines values of the type FontPrec to consist of pairs of values, the first of type FontInd, the second of type Prec.

Types which consist of subranges of other types are defined using a set comprehension construct. The general form of this construct is:

TypeName == { *declarations* | *predicate* • *expression* }

The values of TypeName are just those values of *expression* which satisfy the condition expressed by *predicate*. The *declarations* part declares the types of the variables used in the predicate and expression parts. For example, to define a tuple type whose components are restricted ranges of other types:

WCOrd == **R**
PatSize == {x,y:WCOrd | $x>0 \wedge y>0$ • (x,y)}

This states that values of the type PatSize consist of pairs of values each of type WCOrd×WCOrd which satisfy the predicate $x>0 \wedge y>0$. This approach is used in preference to the equivalent approach of defining a subtype of WCOrd for strictly positive ordinate values and then defining PatSize as a Cartesian product on this type.

Another example is:

WCOrdPos == $\{r: \text{WCOrd} \mid r>0 \bullet r\}$

This declares WCOrdPos to be a subtype of WCOrd. WCOrdPos consists of those value of the type WCOrd which are positive.

For some simple cases, shorthand forms of the declarations are used. If the predicate is omitted, the default predicate *true* is assumed. If the expression is omitted, the default is the characteristic tuple of the declaration part (the tuple of declared variables in the order in which they are declared). For example, the type WCOrdPos could be written as:

WCOrdPos == $\{r: \text{WCOrd} \mid r>0\}$

Here the expression is omitted and the characteristic tuple of the declaration part (r) is assumed.

Functions are defined using the notation:

Selects == SelectCritType $\rightarrow$ SelectCrit

This defines Selects as a function from values of type SelectCritType to values of type SelectCrit. The arrow '$\rightarrow$' denotes a total function, meaning that there is a value of SelectCrit for every value of type SelectCritType.

The arrow '$\nrightarrow$' denotes a partial function, for example:

CurveBunTab == CurveInd $\nrightarrow$ CurveBun

A partial function is a function which may be undefined for some values of the source type. Thus CurveBunTab defines a function from values of type CurveInd to values of type CurveBun, but for some values of type CurveInd, it is possible that there is no corresponding value of type CurveBun.

Where it is necessary to define a function whose source type is a subtype of some other type, a further extension of the predicate notation is used, for example:

Matrix23 == $\{i,j:\mathbb{N}_1, m:\mathbf{R} \mid 1\leq i\leq 2,\ 1\leq j\leq 3 \bullet (i,j) \rightarrow m\}$

This type is a subtype of:

Matrix == $\mathbb{N}_1 \times \mathbb{N}_1 \rightarrow \mathbf{R}$

which describes a matrix of dimension $p \times q$. For Matrix23 the first index of the source is restricted to the values {1,2} and the second to the values {1,2,3}. The notation $(i,j)\rightarrow m$ means that the value m is associated with the ordered pair (i,j) in the source type.

In order to describe some of the GKS functions in a concise manner, it is convenient to define types which may take any of the values of a number of other types. An abbreviated example is:

PrimAttrValue == pickidentifier <<PickId>> |
nameset <<NameSet>> |
textheight <<TextHt>>

This definition states that values of the type PrimAttrValue are either values of type PickId, values of type NameSet, or values of type TextHt. The constructor names 'pickidentifier', 'nameset' and 'textheight' are chosen to describe usage of the type following. This is done because in some cases, the type name alone either does not provide an adequate explanation, or provides an ambiguous explanation. Types defined in this way are disjoint unions of the component types. A given value of the type corresponds to a value in only one of the component types. Thus in the example here, a value of the type PrimAttrValue arises from only one of the component types PickId, NameSet and TextHt. Thus the value alone is sufficient to determine whether it is the value of a PICK IDENTIFIER, NAMESET or TEXT HEIGHT attribute.

The main usage of such union types is the following:

PrimAttrName == PICKIDENTIFIER | NAMESET | TEXTHEIGHT
PrimAttr == PrimAttrName $\rightarrow$ PrimAttrValue

The type PrimAttr is a function from the source type PrimAttrName to range type PrimAttrValue. The source type denotes the names of primitive attributes and the type PrimAttr associates a value with each of the primitive attributes. There is a notational convention in such types that the name of the constructor function in the union type, for example, pickidentifier, is the lower case equivalent of the name in the source type with which the values can be associated. Thus in this case, the name PICKIDENTIFIER will always have an associated value of type PickId. The one exception is that some type names have a constructor name 'undefined'. This denotes the undefined value of that type and in this case any source type value can map to undefined.

The constructor seq defines a type whose values consist of a sequence of values of some other type, for example,

CharString == seq Char

defines values of type CharString to consist of sequences of values each of type Char. The constructor seq allows sequences of length 0. The first three data type constructor definitions following define sequences of an arbitrary type (X) of length greater than or equal to 1, of length greater than or equal to 2 and of length greater than or equal to 3, respectively. The ordinary sequence type, seq X, allows sequences of length 0. The fourth type definition defines a sequence of an arbitrary type (X) with an even number (greater than 0) of elements.

seq_1 X == $\{f : \text{seq } X \mid \#f \geq 1\}$
seq_2 X == $\{f : \text{seq } X \mid \#f \geq 2\}$
seq_3 X == $\{f : \text{seq } X \mid \#f \geq 3\}$
seq_e X == $\{f : \text{seq } X \mid \#f \bmod 2 = 0 \wedge \#f \neq 0\}$

where the operator # delivers the length of the sequence.

GKS implementations are required to support a minimum set of capabilities, for example to provide at least a certain number of predefined bundle table entries. These constraints are specified in the data type definitions.

11.1.2 Basic types

The basic types used are:

Type	Description
ArName	Archive name (of picture part)
ArSpec	Archive specifier
AuditId	Audit identifier
AuditSpec	Audit specifier
AuditUserData	Audit user data record
Char	Character code
ColrCharacteristics	Colour characteristics
ErrFile	Error file
EscapeFuncId	Escape function identifier
EscapeInData	Escape input data record
EscapeOutData	Escape output data record
GDPData	GENERALIZED DRAWING PRIMITIVE data record
GDPId	GENERALIZED DRAWING PRIMITIVE identifier
GlyphName	Glyph name
InData	Input data record
ISO9541FontName	Name of ISO9541 font resource
MeasureId	Measure process identifier
$\mathbb{N}$	Non-negative integer
$\mathbb{N}_1$	Integer greater than 0
$\mathbb{N}_{0,2}$	Integer equal to 0 or greater than 1
Name	Name

NDCMfSpec	NDC metafile specifier
PicId	Picture identifier
PickId	Pick identifier
PicPartName	Picture part name
R	Real number
RealizMfSpec	Realized metafile specifier
ScissorId	Scissor identifier
SegName	Segment name
StencilName	Stencil name
TilingName	Tiling name
TriggerId	Trigger process identifier
WsGenType	Workstation generic type
WsId	Workstation identifier
WsSpec	Workstation specifier
WsSpecInfo	Workstation specific information
$\mathbb{Z}$	Integer

11.1.3 Data types for GKS control

The type ErrFile is used in OPEN GKS for the type of the error file.

ErrFile

The following types are used by the ESCAPE function:

EscapeFuncId
EscapeInData
EscapeOutData

The following types are used by the error handling functions:

ErrNum ==$\{i: \mathbb{Z} \mid i<0 \vee i>1000\}$
ErrSt == OFF | ON
FuncName == OPEN GKS | CLOSE GKS | ...

The values of error numbers are listed in Annex B. The type FuncName is an enumeration of all the GKS function names (see Annex A for a complete list of function names).

The type RouteDir is used in ROUTE to define the current routing of output primitives.

RouteDir == NDC | BACKDROP

11.1.4 Data types for output primitives

Points in world coordinates are of type WCPoint.

WCPoint == $\{x,y: \text{WCOrd} \bullet (x,y)\}$

Ordinate values in WC are of type WCOrd.

WCOrd == **R**
WCOrdPos == $\{r: \text{WCOrd} \mid r>0 \bullet r\}$

Points in homogeneous world coordinates are of type HWCPoint.

HWCPoint == $\{x,y,w:\text{WCOrd} \bullet (x,y,w)\}$

Character strings are sequences of characters. Char is a basic type.

Char
CharString == seq Char

Colour indices are of type ColrInd.

ColrInd == $\mathbb{I}\mathbf{N}$

Colour array specifiers are of type ColrArraySpec.

ColrArraySpec == indirectarray <<$\mathbb{I}\mathbf{N}_1 \times \mathbb{I}\mathbf{N}_1 \rightarrow$ ColrInd>> |
rgbarray <<$\mathbb{I}\mathbf{N}_1 \times \mathbb{I}\mathbf{N}_1 \rightarrow$ ColrRGB>> |
cieluvarray <<$\mathbb{I}\mathbf{N}_1 \times \mathbb{I}\mathbf{N}_1 \rightarrow$ ColrCIELUV>> |
hsvarray <<$\mathbb{I}\mathbf{N}_1 \times \mathbb{I}\mathbf{N}_1 \rightarrow$ ColrHSV>> |
hlsarray <<$\mathbb{I}\mathbf{N}_1 \times \mathbb{I}\mathbf{N}_1 \rightarrow$ ColrHLS>>

Types for output primitives are now defined.

ClosedNURB == nurb where the curve is closed by connecting the first point to the last
ConicSection == Matrix33 × WCPoint × WCPoint × SenseFlag
The two points define start and end points respectively of
the arc of the conic defined by Matrix33
ControlPoints == hcp <<seq_1 HWCPoint>> | cp <<seq_1 WCPoint>>
EllipticDisc == Matrix33
Knots == seq_1 **R**
Matrix33 == $\{i,j:\mathbb{I}\mathbf{N}_1, m:\mathbf{R} \mid 1 \le i \le 3,\ 1 \le j \le 3 \bullet (i,j) \rightarrow m\}$
Matrix23 == $\{i,j:\mathbb{I}\mathbf{N}_1, m:\mathbf{R} \mid 1 \le i \le 2,\ 1 \le j \le 3 \bullet (i,j) \rightarrow m\}$
The type Matrix33 defines a 3 × 3 matrix
The type Matrix23 denotes a 2 × 3 matrix.
NURB == { k: SplineOrder, P: ControlPoints, W: SeqWeight, T: Knots,
(t_{min}, t_{max}): ParamRan | $\#P > k \ge 1 \wedge \#W = \#P \wedge$
$\#T = k + \#P \wedge (\forall\, i,j \in 1..\#T \mid j > i \Rightarrow T(j) \ge T(i)) \wedge$
$T(k) \le t_{min} < t_{max} \le T(\#P+1)$
$\bullet (k, P, W, T, (t_{min}, t_{max}))\}$
ParamRan == **R**×**R**
PrimParam == setofpolyline <<seq_1 (seq_2 WCPoint)>> |
setofnurb <<seq_1 NURB>> |
setofconicsection <<seq_1 ConicSection>> |
polymarker <<seq_1 WCPoint>> |
setoffillarea <<seq_1 (seq_3 WCPoint) >> |
setofellipticsector <<seq_1 ConicSection>> |
setofellipticsegment <<seq_1 ConicSection>> |
setofellipticdisc <<seq_1 EllipticDisc>> |
setofclosednurb <<seq_1 ClosedNURB>> |
text <<WCPoint × CharString>> |
cellarray <<WCPoint × WCPoint × ColrArraySpec>> |
design <<StencilName × StencilOrigin × StencilTran × TilingName ×
TilingOrigin × TilingTran>> |
gdp <<seq WCPoint × GDPId × GDPData>>
SenseFlag == CLOCKWISE | ANTICLOCKWISE
SeqWeight == seq_1 Weight
SplineOrder == $\mathbb{I}\mathbf{N}_1$
StencilName
StencilOrigin == WCPoint
StencilTran == Matrix23

TilingName
TilingOrigin == WCPoint
TilingTran == Matrix23
Weight == $\{r:\mathbf{R} \mid r>0 \bullet r\}$

Types for GDP data records and identifiers, are basic types.

GDPData
GDPId

11.1.5 Data types for output attributes

The values of the type PrimAttrName are the names of the primitive attributes. These are listed in the table below together with the types of the values of each attribute. The table also lists the attribute class to which each attribute belongs and the primitives to which it applies. ALL indicates that an attribute applies to all classes of primitives.

Applies to	Class	Attribute name	Type
ALL	Identification	PICK IDENTIFIER	PickId
ALL	Identification	NAMESET	NameSet
ALL	NDC	SCISSOR SET	ScissorSet
ALL	NDC	GLOBAL TRANSFORMATION	Matrix23
ALL	NDC	LOCAL TRANSFORMATION	Matrix23
AREA	NDC	PATTERN SIZE	PatSize
AREA	NDC	PATTERN REFERENCE POINT	WCPoint
TEXT	NDC	TEXT HEIGHT	TextHt
TEXT	NDC	TEXT UP VECTOR	TextUpVec
TEXT	NDC	TEXT SKEW ANGLE	TextSkew
TEXT	NDC	TEXT PATH	TextPath
TEXT	NDC	TEXT ALIGNMENT	TextAlign
CURVE	Source	CURVE INDEX	CurveInd
CURVE	Source	CURVE ASFS	CurveAsfs
CURVE	Logical	CURVETYPE	CurveType
CURVE	Logical	CURVEWIDTH SCALE FACTOR	CurveWidth
CURVE	Logical	CURVE COLOUR SPECIFIER	ColrSpec
MARKER	Source	MARKER INDEX	MarkerInd
MARKER	Source	MARKER ASFS	MarkerAsfs
MARKER	Logical	MARKERTYPE	MarkerType
MARKER	Logical	MARKERSIZE SCALE FACTOR	MarkerSize
MARKER	Logical	MARKER COLOUR SPECIFIER	ColrSpec
AREA	Source	AREA INDEX	AreaInd
AREA	Source	AREA ASFS	AreaAsfs
AREA	Logical	INTERIOR STYLE	IntStyle
AREA	Logical	INTERIOR STYLE INDEX	IntStyleInd
AREA	Logical	INTERIOR COLOUR SPECIFIER	ColrSpec
AREA	Logical	EDGE FLAG	EdgeFlag
AREA	Logical	EDGETYPE	EdgeType
AREA	Logical	EDGEWIDTH SCALE FACTOR	EdgeWidth
AREA	Logical	EDGE COLOUR SPECIFIER	ColrSpec

Applies to	Class	Attribute name	Type
CHARACTER	Source	CHARACTER INDEX	CharInd
CHARACTER	Source	CHARACTER ASFS	CharAsfs
CHARACTER	Logical	CHARACTER FONT AND PRECISION	FontPrec
CHARACTER	Logical	CHARACTER EXPANSION FACTOR	CharExpan
CHARACTER	Logical	CHARACTER SPACING	CharSpace
CHARACTER	Logical	CHARACTER COLOUR SPECIFIER	ColrSpec

PrimAttrName == PICK IDENTIFIER | NAMESET | ... |
CHARACTER COLOUR SPECIFIER

The types listed in the table above are defined below.

AreaAsfs == AsfValue × AsfValue × AsfValue × AsfValue ×
AsfValue × AsfValue × AsfValue
The order is INTERIOR STYLE, INTERIOR STYLE INDEX, INTERIOR COLOUR SPECIFIER, EDGE FLAG, EDGETYPE, EDGEWIDTH SCALE FACTOR, EDGE COLOUR SPECIFIER

AreaInd == $\mathbb{N}_1$

AsfValue == BUNDLED | INDIVIDUAL

CharAsfs == AsfValue × AsfValue × AsfValue × AsfValue
The order is CHARACTER FONT AND PRECISION, CHARACTER EXPANSION FACTOR, CHARACTER SPACING, CHARACTER COLOUR SPECIFIER

CharExpan == $\{x:\mathbb{R} \mid x \geq 0\}$

CharInd == $\mathbb{N}_1$

CharSpace == $\mathbb{R}$

ClipInd == CLIP | NOCLIP

ColrSpec == indirect <<ColrInd>> |
colrrgb <<ColrRGB>> |
colrcieluv <<ColrCIELUV>> |
colrhsv <<ColrHSV>> |
colrhls <<ColrHLS>>

CurveAsfs == AsfValue × AsfValue × AsfValue
The order is CURVETYPE, CURVEWIDTH SCALE FACTOR, CURVE COLOUR SPECIFIER

CurveInd == $\mathbb{N}_1$

CurveType == $\{x:\mathbb{Z} \mid x \neq 0\}$

CurveWidth == $\{x:\mathbb{R} \mid x \geq 0\}$

EdgeFlag == OFF | ON

EdgeType == $\{x:\mathbb{Z} \mid x \neq 0\}$

EdgeWidth == $\{x:\mathbb{R} \mid x \geq 0\}$

FontInd == $\{x:\mathbb{Z} \mid x \neq 0\}$

FontPrec == FontInd × Prec

GenName == workstationidentifier<<WsId>> | segmentname<<SegName>>
| name<<Name>>

HorAlign == NORMAL | LEFT | CENTRE | RIGHT

IntStyle == HOLLOW | SOLID | PATTERN | HATCH | EMPTY

IntStyleInd == $\{x:\mathbb{Z} \mid x \neq 0\}$

MarkerAsfs == AsfValue × AsfValue × AsfValue
The order is MARKERTYPE, MARKERSIZE SCALE FACTOR, MARKER COLOUR SPECIFIER

MarkerInd == $\mathbb{N}_1$

MarkerSize == $\{x:\mathbf{R} \mid x \geq 0\}$

MarkerType == $\{x:\mathbb{Z} \mid x \neq 0\}$

Name

NameSet == $\mathbb{P}$ GenName

NDCOrd == $\{r:\mathbf{R} \mid \exists r_{max}:\mathbf{R} \bullet -r_{max} \leq r \leq r_{max} \wedge r_{max} \geq 7\}$

NDCRect == $\{x_{min}, x_{max}, y_{min}, y_{max}:\text{NDCOrd} \mid x_{min} < x_{max} \wedge y_{min} < y_{max} \bullet (x_{min}, x_{max}, y_{min}, y_{max})\}$

NDCRectSet == $\mathbb{P}$ NDCRect

PatSize == $\{x,y:\text{WCOrd} \mid x>0 \wedge y>0 \bullet (x,y)\}$

PickId

Prec == STRING | CHAR | STROKE

Scissor == ClipInd × NDCRectSet × ShieldInd × NDCRectSet

ScissorId

ScissorIdSet == $\mathbb{P}$ ScissorId

ScissorSet == ScissorId ⇸ Scissor

SegName

ShieldInd == SHIELD | NOSHIELD

TextAlign == HorAlign × VertAlign

TextHt == $\{x:\text{WCOrd} \mid x>0\}$

TextPath == RIGHT | LEFT | UP | DOWN

TextSkew == **R**
The type TextSkew is defined in radians (positive is anticlockwise).

TextUpVec == $\{x,y:\text{WCOrd} \mid x^2+y^2>0 \bullet (x,y)\}$

VertAlign == NORMAL | TOP | CAP | HALF | BASE | BOTTOM

The type PrimAttrValue is the supertype of all the types of the primitive attributes.

PrimAttrValue == pickidentifier <<PickId>> |
nameset <<NameSet>> |
scissorset <<ScissorSet>> |
globaltransformation <<Matrix23>> |
localtransformation <<Matrix23>> |
patternsize <<PatSize>> |
patternreferencepoint <<WCPoint>> |
textheight <<TextHt>> |
textupvector <<TextUpVec>> |
textskewangle <<TextSkew>> |
textpath <<TextPath>> |
textalignment <<TextAlign>> |
curveindex <<CurveInd>> |
curveasfs <<CurveAsfs>> |
curvetype <<CurveType>> |
curvewidthscalefactor <<CurveWidth>> |
curvecolourspecifier <<ColrSpec>> |
markerindex <<MarkerInd>> |
markerasfs <<MarkerAsfs>> |

markertype <<MarkerType>> |
markersizescalefactor <<MarkerSize>> |
markercolourspecifier <<ColrSpec>> |
areaindex <<AreaInd>> |
areaasfs <<AreaAsfs>> |
interiorstyle <<IntStyle>> |
interiorstyleindex <<IntStyleInd>> |
interiorcolourspecifier <<ColrSpec>> |
edgeflag <<EdgeFlag>> |
edgetype <<EdgeType>> |
edgewidthscalefactor <<EdgeWidth>> |
edgecolourspecifier <<ColrSpec>> |
characterindex <<CharInd>> |
characterasfs <<CharAsfs>> |
characterfontandprecision <<FontPrec>> |
characterexpansionfactor <<CharExpan>> |
characterspacing <<CharSpace>> |
charactercolourspecifier <<ColrSpec>>

PrimAttr == PrimAttrName → PrimAttrValue

11.1.6 Data types for transformations

The following types are used to describe transformations.

DCOrd == **R**

DCVp == $\{x_{min}, x_{max}, y_{min}, y_{max}$:DCOrd | $x_{min} < x_{max} \wedge y_{min} < y_{max} \bullet (x_{min}, x_{max}, y_{min}, y_{max})\}$

LDCWin == $\{x_{min}, x_{max}, y_{min}, y_{max}$:LDCOrd | $x_{min} < x_{max} \wedge y_{min} < y_{max} \wedge 0 \le x_{min} < 1 \wedge 0 < x_{max} \le 1 \wedge 0 \le y_{min} < 1 \wedge 0 < y_{max} \le 1 \bullet (x_{min}, x_{max}, y_{min}, y_{max})\}$

LDCOrd == **R**

LDCRect == $\{x_{min}, x_{max}, y_{min}, y_{max}$:LDCOrd | $x_{min} < x_{max} \wedge y_{min} < y_{max} \wedge 0 \le x_{min} < 1 \wedge 0 < x_{max} \le 1 \wedge 0 \le y_{min} < 1 \wedge 0 < y_{max} \le 1 \bullet (x_{min}, x_{max}, y_{min}, y_{max})\}$

LDCRectSet == ***P*** LDCRect

LDCVp == $\{x_{min}, x_{max}, y_{min}, y_{max}$:LDCOrd | $x_{min} < x_{max} \wedge y_{min} < y_{max} \wedge 0 \le x_{min} < 1 \wedge 0 < x_{max} \le 1 \wedge 0 \le y_{min} < 1 \wedge 0 < y_{max} \le 1 \bullet (x_{min}, x_{max}, y_{min}, y_{max})\}$

NDCWin == $\{x_{min}, x_{max}, y_{min}, y_{max}$:NDCOrd | $x_{min} < x_{max} \wedge y_{min} < y_{max} \bullet (x_{min}, x_{max}, y_{min}, y_{max})\}$

NDCVp == $\{x_{min}, x_{max}, y_{min}, y_{max}$:NDCOrd | $x_{min} < x_{max} \wedge y_{min} < y_{max} \bullet (x_{min}, x_{max}, y_{min}, y_{max})\}$

NormTran == WCWin × NDCVp
NormTranNum == $\{n:\mathbb{I}\mathbb{N} \mid 0 \le n \le 63\}$
NormTranNum1 == $\{n:\mathbb{I}\mathbb{N} \mid 1 \le n \le 63\}$
Pri == HIGHER | LOWER
ScissorMode == LOCUS | SHAPE

ViewInd == $\{n:\mathbb{I}\mathbb{N} \mid 0 \le n \le 15\}$
ViewInd1 == $\{n:\mathbb{I}\mathbb{N} \mid 1 \le n \le 15\}$
ViewPri == $\{f:\mathbb{I}\mathbb{N} \rightarrow\!\!\!\!\!\!+\ \text{ViewInd} \mid \text{dom } f = 0 \,..\, 15 \wedge \text{ran } f = \text{ViewInd} \wedge \forall\, n_1, n_2 : \text{dom } f \bullet f(n_1) = f(n_2) \Rightarrow n_1 = n_2\}$
ViewScissor == ClipInd × LDCRectSet × ShieldInd × LDCRectSet
ViewUpVec == $\{x,y:\text{WCOrd} \mid x^2 + y^2 > 0 \bullet (x,y)\}$
VpInPri == $\{f:\mathbb{I}\mathbb{N} \rightarrow\!\!\!\!\!\!+\ \text{NormTranNum} \mid \text{dom } f = 0 \,..\, 63 \wedge \text{ran } f = \text{NormTranNum} \wedge \forall\, n_1, n_2 : \text{dom } f \bullet f(n_1) = f(n_2) \Rightarrow n_1 = n_2\}$
WCWin == $\{x_{min}, x_{max}, y_{min}, y_{max} : \text{WCOrd} \mid x_{min} < x_{max} \wedge y_{min} < y_{max} \bullet (x_{min}, x_{max}, y_{min}, y_{max})\}$

The type NormTranNum1 is the subtype of NormTranNum which excludes normalization transformation number 0. The type ViewPri is a bijection from the integers 0 .. 15 to the view indices. Each view index has a unique position in the sequence. Position 0 is the highest priority. The type VpInPri is a bijection from the integers 0 .. 63 to the normalization transformation numbers. Each normalization transformation number has a unique position in the sequence. Position 0 is the highest priority.

11.1.7 Data types for NDC picture

The type Position is used by REORDER NDC PICTURE.

Position == FRONT | BACK

11.1.8 Data types for metafiles

Values of the identifiers associated with pictures in metafiles are of type PicId. This is a basic type.

PicId

NDC metafile specifiers are type NDCMfSpec and realized metafile specifiers are of type RealizMfSpec

NDCMfSpec
RealizMfSpec

11.1.9 Data types for picture parts and archives

The type PicPartName is a basic type for the names of picture parts.

PicPartName

The following flags are used when copying primitives from the picture part store.

ScissorSelect == CHANGE | LEAVE
TransMode == REPLACE | PRE | POST

Values of the names associated with picture parts in archives are of type ArName. NDC archive specifiers are of type ArSpec. Both are basic types.

ArName
ArSpec

11.1.10 Data types for utility functions

The utility functions for building transformation matrices (see 12.13) use the following types. WCNDCPoint is a point in world or NDC coordinates.

CircularArcSpec == $\{s_x,s_y,i_x,i_y,e_x,e_y$: WCOrd | $(i_x-s_x)(i_y-e_y)-(i_x-e_x)(i_y-s_y)\neq 0$ • $((s_x,s_y),(i_x,i_y),(e_x,e_y))\}$

The points $(s_x,s_y),(i_x,i_y),(e_x,e_y)$ define the start, intermediate and end points of a circular arc

EllipseSpec == $\{p_x,p_y,c_x,c_y,q_x,q_y$: WCOrd | $(c_x-p_x)(c_y-q_y)-(c_x-q_x)(c_y-p_y)\neq 0$ • $((p_x,p_y),(c_x,c_y),(q_x,q_y))\}$

The point (c_x,c_y) defines the centre of the ellipse, (p_x,p_y) defines the first conjugate radius end point and (q_x,q_y) defines the second conjugate radius end point

HyperbolaSpec == $(p_x,p_y,c_x,c_y,q_x,q_y$: WCOrd | $(c_x-p_x)(c_y-q_y)-(c_x-q_x)(c_y-p_y)\neq 0$ • $((p_x,p_y),(c_x,c_y),(q_x,q_y))\}$

The point (c_x,cy) defines the centre of the hyperbola, (p_x,p_y) is the transverse radius end point and (q_x,q_y) is the conjugate radius end point

ParabolaSpec == $(p_x,p_y,c_x,c_y,q_x,q_y$: WCOrd | $(c_x-p_x)(c_y-q_y)-(c_x-q_x)(c_y-p_y)\neq 0$ • $((p_x,p_y),(c_x,c_y),(q_x,q_y))\}$

The point (p_x,p_y) is the start point of the arc.
The point (q_x,q_y) is the end point of the arc.
The point (c_x,c_y) is the tangential intersection point.

NDCPoint == $\{x,y$:NDCOrd • $(x,y)\}$

OrdSelector == X | Y

Rad == **R**

The type Rad is defined in radians (positive is anticlockwise).

ScaleFactors == $\mathbf{R}\times\mathbf{R}$

Vec == $\{x,y$:WCOrd | $x^2+y^2>0$ • $(x,y)\}$

WCNDCPoint == wcpoint <<WCPoint>> | ndcpoint <<NDCPoint>>

11.1.11 Data types for segments

The values of the type SegAttrName are the names of the segment attributes. These are listed in the table below with the types of the values of each attribute.

Attribute name	Type
VISIBILITY	Vis
HIGHLIGHTING	Highl
DETECTABILITY	Det
SEGMENT TRANSFORMATION	Matrix23
SEGMENT PRIORITY	SegPri

SegAttrName == VISIBILITY | HIGHLIGHTING | ...

Definitions of the types in the table above are given below.

Det == UNDETECTABLE | DETECTABLE

Highl == NORMAL | HIGHLIGHTED

SegPri == $\{x$:**R** | $0\leq x\leq 1\}$

Vis == VISIBLE | INVISIBLE

The type SegAttrValue is the supertype of all the types of the segment attributes.

SegAttrValue == visibility<<Vis>> |
highlighting<<Highl>> |
detectability<<Det>> |
segmenttransformation<<Matrix23>> |
segmentpriority<<SegPri>>

11.1.12 Data types for input

Types for input are described below.

ChoiceNum == $\mathbb{N}_1$

ChoiceStatus == OK | NOCHOICE

DevCompose == ($\mathbb{P}$ MeasureId) × ($\mathbb{P}$ TriggerId)

DevId == WsId × $\mathbb{N}_1$ × MeasureClass
The value $\mathbb{N}_1$ defines the device number within the measure class on the specified workstation

DevIdValueSet == devidvalue <<$\mathbb{P}$ (DevId × InValue)>>
| NONE

EchoArea == $\{x_{min}, x_{max}, y_{min}, y_{max}$:DCOrd | $x_{min} < x_{max} \wedge y_{min} < y_{max}$ • $(x_{min}, x_{max}, y_{min}, y_{max})\}$

EchoSwitch == ECHO | NOECHO

InitInValue == InValue × PET × EchoSwitch × EchoArea × InData

InitValue == initialvalue <<InValue>>
| pet <<PET>>
| echoflag <<EchoSwitch>>
| echoarea <<EchoArea>>
| datarecord <<InData>>
| all <<InitInValue>>

InData

InStatus == OK | NONE

InValue == locatorvalue <<NormTranNum × WCPoint>>
| strokevalue <<NormTranNum × seq WCPoint>>
| valuatorvalue <<$\mathbb{R}$>>
| choicevalue <<ChoiceStatus × ChoiceNum>>
| pickvalue <<PickStatus × NameSet × PickId>>
| stringvalue <<CharString>>
| compvalue <<seq InValue>>

MeasureClass == LOCATOR | STROKE | VALUATOR | CHOICE | PICK |
STRING | COMPOSITE

MeasureId

OpMode == REQUEST | SAMPLE | EVENT

PET == $\{n: \mathbb{Z} \mid n \neq 0\}$
The type PET defines Prompt and Echo Type

PickStatus == OK | NOPICK

SeqMeasureId == seq_1 MeasureId

Timeout == $\{x: \mathbb{R} \mid x \geq 0\}$

TriggerId

TriggerSet == $\mathbb{P}$ TriggerId

The type InData is a basic type describing input data records. Timeout is defined in seconds.

11.1.13 Data types for workstation control

Types associated with workstation control are listed below. All are basic types.

Entity	Type
Workstation generic type	WsGenType
Workstation identifier	WsId
Workstation specifier	WsSpec
Workstation specific information	WsSpecInfo

WsGenType
WsId
WsSpec
WsSpecInfo
WsType ==WsGenType × WsSpecInfo
ChangeFlag == CHANGED | NOTCHANGED
ColrAvail == COLOUR | MONOCHROME
DCUnits == METRES | OTHER
DispKind == VECTOR | RASTER | OTHER
WsStatus == UPTODATE | NOTUPTODATE

The type ChangeFlag is used to indicate whether or not the contents of a specific workstation description table has changed.

The types associated with bundles are listed below.

Bundle	Type
Curve	CurveBun
Marker	MarkerBun
Character	CharBun
Area	AreaBun

Type definitions are given below.

AreaBun == IntStyle × IntStyleInd × ColrSpec
× EdgeFlag × EdgeType × EdgeWidth × ColrSpec
First ColSpec is interior colour specifier, second is edge colour specifier
CharBun == FontPrec × CharExpan × CharSpace × ColrSpec
CurveBun == CurveType × CurveWidth × ColrSpec
MarkerBun == MarkerType × MarkerSize × ColrSpec

Colour models are specified by values of the type ColrModel:

ColrModel == RGB | CIELUV | HSV | HLS

The type ColrCoords specifies colour coordinates. The number of coordinates necessary is dependent on the colour model. Colours in the RGB and CIELUV colour models, which are required to be available, are specified by three coordinates. Colours in the HSV and HLS colour models, which may be provided, are also specified by three coordinates. Additional colour models may be included in the ISO International Register of Graphical Items which is maintained by the Registration Authority (see 4.2). Implementation dependent colour models may also be provided. The type definitions are:

ColrCIELUV == $\{u',v',Y:\mathbf{R} \bullet (u',v',Y)\}$
ColrHLS == $\{h,l,s:\mathbf{R} \mid 0\leq h\leq 1 \wedge 0\leq l\leq 1 \wedge 0\leq s\leq 1 \bullet (h,l,s)\}$
ColrHSV == $\{h,s,v:\mathbf{R} \mid 0\leq h\leq 1 \wedge 0\leq s\leq 1 \wedge 0\leq v\leq 1 \bullet (h,s,v)\}$
ColrRGB == $\{r,g,b:\mathbf{R} \mid 0\leq r\leq 1 \wedge 0\leq g\leq 1 \wedge 0\leq b\leq 1 \bullet (r,g,b)\}$
ColrCoords == colrrgb <<ColrRGB>> | colrcieluv <<ColrCIELUV>>
| colrhsv <<ColrHSV>> | colrhls <<ColrHLS>> | ···

The type ColrCharacteristics describes the colour characteristics of a display. It is a basic type.

ColrCharacteristics

The type ConversionFlag is used by the colour utility function, CONVERT COLOUR.

ConversionFlag == DONE | APPROXIMATED | NOTCONVERTED

The visual effect state is described by the type:

VisEffSt == SUPPRESS | ALLOW

The bundle, pattern and colour tables are described by the types:

AreaBunTab == AreaInd ⇸ AreaBun
CharBunTab == CharInd ⇸ CharBun
ColrTab == ColrInd ⇸ ColrCoords
CurveBunTab == CurveInd ⇸ CurveBun
MarkerBunTab == MarkerInd ⇸ MarkerBun
PatTab == IntStyleInd ⇸ ColrArraySpec
Rep == curve <<CurveInd × CurveBun>> |
marker <<MarkerInd × MarkerBun>> |
area <<AreaInd × AreaBun>> |
pattern <<IntStyleInd × ColrArraySpec>> |
character <<CharInd × CharBun>> |
colour <<ColrInd × ColrCoords>> |
undefined

For the types AreaBunTab to MarkerBunTab, an entry for index value 1 is always defined. At least 20 entries are supported for each of the curve, marker and character bundle tables and 10 entries for each of the area bundle table and pattern table.

Workstation selection criteria are defined by the types:

SelectCritType == DISPLAY | HIGHLIGHTING | DETECTABILITY
SelectCrits == SelectCritType → SelectCrit

Selection criteria are defined by the type SelectCrit. This type is described further in 11.1.18.

11.1.14 Data types for inquiry functions

The following types are used by inquiry functions.

ErrInd == $\{i:\mathbb{Z} \mid i<0 \vee i>1000\}$
SetReal == SET | REALIZED

11.1.15 Data types for operating state

The following types describe the operating state of GKS. The first distinguishes between GKS being open or not. The others distinguish the state in which the specified store is (open or closed).

```
GKSOpSt        == GKCL | GKOP
PicPartOpSt    == PPCL | PPOP
SegOpSt        == SGCL | SGOP
TilingOpSt     == TLCL | TLOP
StencilOpSt    == STCL | STOP
```

11.1.16 Data types for font and glyph functions

The following basic types are used by the font and glyph functions.

```
GlyphName
ISO9541FontName
```

11.1.17 Data types for audit trails

Audit trail identifiers are of type AuditId and specifiers are of type AuditSpec. The names of operations allowed on audit trails are of type PlaybackOps and ProcessOps. The type RecAuditFlag denotes whether recording is on or off. Audit user data records are of type AuditUserData.

```
AuditId
AuditOps        == OPEN <<AuditSpec>> | CLOSE | BEGIN | END
AuditSpec
AuditUserData
ProcessOps      == SKIP | DO
PlaybackOps     == OPEN <<AuditSpec>> | CLOSE
RecAuditFlag    == OFF | ON
```

The type AuditFuncName consists of the names of the GKS functions as listed in Annex A except for ERROR HANDLING and ERROR LOGGING.

The type FuncParams is the supertype of the types of the GKS function parameters for the functions named in AuditFuncName. The full definition of the type is derivable from the function headings in clauses 12, 13 and 14. The first two entries in the type are:

```
FuncParams      == opengks <<ErrFile>>
                 | closegks
                 | ...
```

11.1.18 Data types for selection criteria

Selection criteria are defined by the type SelectCrit.

```
SelectCrit      == contains <<NameSet>> |
                   isin <<NameSet>> |
                   equals <<NameSet>> |
                   SELECTALL |
                   REJECTALL |
                   and <<SelectCrit × SelectCrit>> |
                   or <<SelectCrit × SelectCrit>> |
                   not <<SelectCrit>>
```

If the nameset associated with a primitive is *pns* and *ns* is a nameset, the operators for constructing selection criteria are:

CONTAINS(*ns*) $ns \subseteq pns$

ISIN(*ns*)	*pns*⊆*ns*
EQUALS(*ns*)	*ns*≡*pns*
SELECTALL	True (for any nameset)
REJECTALL	False (for any nameset)
AND	
OR	
NOT	

The algebraic definitions of selection criteria are given below. Selection criteria are constructed from the operators:

CONTAINS: NameSet → SelectCrit
ISIN: NameSet → SelectCrit
EQUALS: NameSet → SelectCrit
AND: SelectCrit SelectCrit → SelectCrit
OR: SelectCrit SelectCrit → SelectCrit
NOT: SelectCrit → SelectCrit
SELECTALL: → SelectCrit
REJECTALL: → SelectCrit

A function:

satisfy: NameSet SelectCrit → Bool

which delivers T (true) if the nameset satisfies the selection criterion and F (false) if it does not, is defined by the equations:

satisfy(ns, AND(s1, s2)) = satisfy(ns, s1) and satisfy(ns, s2)
satisfy(ns, OR(s1, s2)) = satisfy(ns, s1) or satisfy(ns, s2)
satisfy(ns, NOT(s)) = not (satisfy(ns, s))
satisfy(ns, CONTAINS(emptyNS)) = T
satisfy(ns, CONTAINS(mkSet(n)) = member(n, ns)
satisfy(ns, CONTAINS(ns1 ∪ ns2)) =
  satisfy(ns, CONTAINS(ns1)) and satisfy(ns, CONTAINS(ns2))
satisfy(emptyNS, ISIN(ns)) = T
satisfy(mkSet(n), ISIN(ns)) = member(n, ns)
satisfy(ns1 ∪ ns2, ISIN(ns)) = satisfy(ns1, ISIN(ns)) and satisfy(ns2, ISIN(ns))
satisfy(ns, EQUALS(ns1)) = (ns == ns1)
satisfy(ns, SELECTALL) = T
satisfy(ns, REJECTALL) = F

where the function member(n,ns) delivers T if n is an element of ns and F otherwise, and the operator '=' determines equality between terms of the same sort. 'mkSet' and '∪' are set construction operators. 'and', 'or' and 'not' are the logical operators.

11.1.19 Data types for paths, tilings and stencils

The following types are used to describe stencils and tilings.

AbutSpec	== RIGHT | LEFT | TOP | BOTTOM | CENTREVERT | CENTREHORIZ | CAPLINE | BASELINE
Boundary	== closedpathseq << seq_1 Path>> | area <<EllipticDisc>>
Cap	== BUTTED | ROUNDED | SQUARE

ContourAttrName == STYLE | WIDTH | CAP | JOIN
ContourAttrValue == style <<Style>> | width <<NDCOrd>> | cap <<Cap>> | join <<Join>>
HLCPoint == $\{x,y,w$:LCOrd • $(x,y,w)\}$
InsideRule == EVENODD | WINDING
InstanceNum == $\{r$: **R** | $r \geq 0\}$
InstanceSeqSpec == seq_1 SeqSpec
InstanceSpec == put <<LCPoint × LCPoint>> |
order is reference point on instance, reference point on open stencil
align <<LCPoint × LCPoint × LCPoint × LCPoint>>
order is reference point on instance, direction point on instance,
reference point on open stencil, direction point on open stencil
map <<LCPoint × LCPoint × LCPoint × LCPoint × LCPoint × LCPoint>>
order is 3 reference points on instance followed by
3 corresponding points on open stencil
Join == ROUND | BEVEL | mitred <<NDCOrd>>
LCClosedNURB == LCNURB where the curve is closed by connecting the first point to the last.
LCConicSection == Matrix33 × LCPoint × LCPoint × SenseFlag
LCControlPoints == hlccp <<seq_1 HLCPoint>> | lccp <<seq_1 LCPoint>>
LCNURB == { k: SplineOrder, P: LCControlPoints, W: SeqWeight, T: Knots,
(t_{min},t_{max}): ParamRan | $\#P > k \geq 1 \wedge \#W = \#P \wedge$
$\#T = k+\#P \wedge (\forall\, i,j \in 1..\#T \mid j>i \Rightarrow T(j) \geq T(i)) \wedge$
$T(k) \leq t_{min} < t_{max} \leq T(\#P+1)$
• $(k,P,W,T,(t_{min},t_{max}))\}$
LCOrd == **R**
stencil local coordinate system
LCPoint == $\{x,y$:LCOrd • $(x,y)\}$
LCStencilOrigin == LCPoint
LCTilingOrigin == LCPoint
Path == polyline <<seq_2 LCPoint>> | nurb <<LCNURB>> |
conicsection<<LCConicSection>>
PathSpec == AbutSpec × Path × LCPoint
RepTech == dx <<LCOrd>> | dy <<LCOrd>> | dxdy <<LCOrd × LCOrd>> |
dydx <<LCOrd × LCOrd>>
SeqBoundary == seq_1 Boundary
SeqPaths == seq_1 Path
SeqSpec == StencilName × StencilName × LCPoint
order is instance name, existing stencil name, reference point
StencilAttrName == TOPY | CAPY | HALFY | BASEY | BOTTOMY | CENTREY |
LEFTX | RIGHTX | CENTREX |
CENTRE | ORIGIN | CENTRETOP | CENTREBOTTOM |
CENTRELEFT | CENTRERIGHT |
TOPLEFT | TOPRIGHT | BOTTOMLEFT | BOTTOMRIGHT
StencilAttrValue == topy <<LCOrd>> | capy <<LCOrd>> | halfy <<LCOrd>> |
basey <<LCOrd>> | bottomy <<LCOrd>> |
centrey <<LCOrd>> | leftx <<LCOrd>> | rightx <<LCOrd>> |
centrex <<LCOrd>> |
centre <<LCPoint>> | origin <<LCPoint>> | centretop <<LCPoint>> |
centrebottom <<LCPoint>> |
centreleft <<LCPoint>> | centreright <<LCPoint>> |
topleft <<LCPpoint>> | topright <<LCPoint>> |
bottomleft <<LCPoint>> | bottomright <<LCPoint>>
Style == SOLID | DASHED | DOTTED | DASHED-DOTTED |

	DASHED-DOTTED-DOTTED
TilingParam	== setofpolyline <<(seq_1 (seq_2 LCPoint)) × ColrSpec>> |
	setofnurb <<(seq_1 LCNURB) × ColrSpec>> |
	setofconicsection <<seq_1 LCConicSection × ColrSpec>> |
	setoffillarea <<(seq_1 (seq_3 LCPoint)) × ColrSpec >> |
	setofellipticsector <<(seq_1 LCConicSection) × ColrSpec>> |
	setofellipticsegment <<(seq_1 LCConicSection) × ColrSpec>> |
	setofellipticdisc <<(seq_1 EllipticDisc) × ColrSpec>> |
	setofclosednurb <<(seq_1 LCClosedNURB) × ColrSpec>> |
	design <<StencilName × LCStencilOrigin × StencilTran × TilingName × LCTilingOrigin × TilingTran>>

11.2 Data type definitions for state lists and description tables

11.2.1 Introduction

In this subclause, the types of GKS data structures are defined. Each data structure is defined by a type, say Xx. This is a function from the source type XxEntName to XxEntValue. The left-hand column of the tables below contain the description of the entry. The centre column is the corresponding value of XxEntName and the right-hand column is the corresponding type of the entry. The type XxEntValue is a union of the types of the individual entries given in the right-hand column. The workstation description table entries may not all apply to all workstation types and so this type is a partial, rather than total function as some entries may not be defined for a particular workstation type.

11.2.2 Operating state list (OSL)

GKS operating state	GKSOPST	GKSOpSt
picture part operating state	PICPARTOPST	PicPartOpSt
segment operating state	SEGOPST	SegOpSt
stencil operating state	STENCILOPST	StencilOpSt
tiling operating state	TILINGOPST	TilingOpSt

11.2.3 GKS description table (GDT)

set of available workstation types	AVAILWSGENTYPES	***P*** WsGenType
set of available fonts	AVAILFONTS	***P*** ISO9541FontName
default font index mapping	DEFFONTINDMAP	FontInd →ISO9541FontName

11.2.4 GKS state list (GSL)

route direction	ROUTEDIR	RouteDir
scissor mode	SCISSORMODE	ScissorMode
set of open workstations	OPENWSIDS	***P*** WsId
set of active workstations	ACTIVEWSIDS	***P*** WsId
set of open audits	OPENAUDITS	***P*** (AuditId × RecAuditFlag)
set of open playbacks	OPENPLAYBACKS	***P*** AuditId
input queue	INPUTQUEUE	seq (DevId × InValue)
font index mapping	FONTINDMAP	FontInd ⇸ ISO9541FontName
current primitive attributes	CURPRIMATTRS	PrimAttr
current normalization transformation number	CURNORMTRANNUM	NormTranNum
normalization transformations	NORMTRANS	NormTranNum → NormTran
viewport input priorities	VPINPRIS	VpInPri
name of the open picture part	OPENPICPARTNAME	picturepartname<<PicPartName>> | undefined
set of picture part names in use	PICPARTNAMES	***P*** PicPartName

name of the open segment	OPENSEGNAME	segmentname<<SegName>> | undefined
set of segment names in use	SEGNAMES	***P*** SegName
set of segment state lists	SEGSTLISTS	***P*** SSL
name of the open stencil	OPENSTENCILNAME	stencilname<<StencilName>> | undefined
set of stencil names in use	STENCILNAMES	***P*** StencilName
set of stencil state lists	STENCILSTLISTS	***P*** STSL
current contour attributes	CURCONTOURATTRS	ContourAttrName → ContourAttrValue
name of the open tiling	OPENTILINGNAME	tilingname<<TilingName>> | undefined
set of tiling names in use	TILINGNAMES	***P*** TilingName

11.2.5 Workstation state list (WSL)

curve bundle table	CURVEBUNTAB	CurveBunTab
marker bundle table	MARKERBUNTAB	MarkerBunTab
area bundle table	AREABUNTAB	AreaBunTab
pattern table	PATTAB	PatTab
character bundle table	CHARBUNTAB	CharBunTab
colour table	COLRTAB	ColrTab
current workstation window	CURWSWIN	LDCWin
current workstation viewport	CURWSVP	DCVp
current visual effects state	CURVISEFFST	VisEffSt
logical input device operating modes	OPMODES	DevId ⇸ OpMode
logical input device initial values	INITVALUES	DevId ⇸ InitInValue
selection criteria	SELECTCRITS	Selects
set of stored segments for this workstation	SEGNAMES	***P*** SegName
view representations	VIEWREPS	ViewInd → SelectCrit × Matrix23 × Matrix23 × ViewScissor
view priorities	VIEWPRIS	ViewPri

11.2.6 Workstation description table (WDT)

The type WDTSt describes the state of each specific workstation description table entry for an open workstation.

WDTSt == WDTEntName ⇸ ChangeFlag

Constraints on the number of entities of a particular type that have to be provided, for example at least 5 predefined curve bundle table entries have to be provided, are indicated beneath the datatype of the state list or description table entry. For example:

predefined curve bundles	PREDCURVEBUNS	CurveBunTab
		#(dom WDT(PREDCURVEBUNS)) ≥ 5

This states that the size ('#') of the domain ('dom') of the PREDCURVEBUNS entry in the workstation description table (WDT(PREDCURVEBUNS)) is (the number of curve indices for which bundles are defined) at least 5.

workstation type	WSTYPE	WsType
workstation status	WSSTATUS	WsStatus
device coordinate units	DCUNITS	DCUnits
display space size	DISPSIZE	
in device units (DC)		DCOrd × DCOrd
in raster units		$\mathbb{N} \times \mathbb{N}$

GKS functions	Data type definitions for state lists and description tables	
vector or raster display	DISPKIND	DispKind
set of available curvetypes	AVAILCURVETYPES	***P*** CurveType
number of available curvewidths	NUMAVAILCURVEWIDTHS	$\mathbb{N}$
(a value of 0 indicates that a continuous range is supported)		
nominal curvewidth (DC)	NOMCURVEWIDTH	DCOrd
minimum curvewidth (DC)	MINCURVEWIDTH	DCOrd
maximum curvewidth (DC)	MAXCURVEWIDTH	DCOrd
predefined curve bundles	PREDCURVEBUNS	CurveBunTab
	#(dom WDT(PREDCURVEBUNS)) ≥ 5	
set of available marker types	AVAILMARKERTYPES	***P*** MarkerType
number of available marker sizes	NUMAVAILMARKERSIZES	$\mathbb{N}$
(a value of 0 indicates that a continuous range is supported)		
nominal marker size (DC)	NOMMARKERSIZE	DCOrd
minimum marker size (DC)	MINMARKERSIZE	DCOrd
maximum marker size (DC)	MAXMARKERSIZE	DCOrd
predefined marker bundles	PREDMARKERBUNS	MarkerBunTab
	#(dom WDT(PREDMARKERBUNS)) ≥ 5	
set of available interior styles	AVAILINTSTYLES	***P*** IntStyle
	#(WDT(AVAILINTSTYLES)) ≥3	
set of available hatch styles	AVAILHATCHSTYLES	***P*** IntStyleInd
predefined area bundles	PREDAREABUNS	AreaBunTab
	#(dom WDT(PREDAREABUNS)) ≥ 5	
set of available edgetypes	AVAILEDGETYPES	***P*** EdgeType
number of available edgewidths	NUMAVAILEDGEWIDTHS	$\mathbb{N}$
(a value of 0 indicates that a continuous range is supported)		
nominal edgewidth (DC)	NOMEDGEWIDTH	DCOrd
minimum edgewidth (DC)	MINEDGEWIDTH	DCOrd
maximum edgewidth (DC)	MAXEDGEWIDTH	DCOrd
predefined pattern representations	PREDPATS	PatTab
	#(dom WDT(PREDPATS)) ≥ 1	
set of available font and precision pairs	AVAILFONTPRECS	***P*** FontPrec
number of available character expansion factors	NUMAVAILCHAREXPANS	$\mathbb{N}$
(a value of 0 indicates that a continuous range is supported)		
minimum character expansion factor	MINCHAREXPAN	CharExpan
maximum character expansion factor	MAXCHAREXPAN	CharExpan
number of available text heights	NUMAVAILTEXTHT	$\mathbb{N}$
(a value of 0 indicates that a continuous range is supported)		
minimum text height (DC)	MINTEXTHT	DCOrd
maximum text height (DC)	MAXTEXTHT	DCOrd
predefined character bundles	PREDCHARBUNS	CharBunTab
	#(dom WDT(PREDCHARBUNS)) ≥ 6	
set of available colour models	AVAILCOLRMODELS	***P*** ColrModel
colour characteristics of display	COLRCHARACTERISTICS	ColrCharacteristics
number of available colours or intensities	NUMAVAILCOLRS	$\mathbb{N}_{0,2}$
colour available	COLRAVAIL	ColrAvail
predefined colour representations	PREDCOLRREPS	ColrTab
	#(dom WDT(PREDCOLRREPS)) ≥ 2	
set of available GDPs	AVAILGDPS	***P*** GDPId
set of available input devices	AVAILINDEVS	***P*** DevId
logical input device initial values	INITVALUES	DevId → InitInValue
set of available measures	AVAILMEASUREIDS	***P*** MeasureId
set of available triggers	AVAILTRIGGERIDS	***P*** TriggerId

device compositions	DEVCOMPOSE	DevId ⇸ DevCompose

11.2.7 Error state list (ESL)

error state	ERRST	ErrSt
error file	ERRFILE	crrorfile<<ErrFile>> | undefined
input device causing queue overflow	DEVID	deviceidentifier<<DevId>> | undefined

11.2.8 Segment state list (SSL)

segment name	SEGNAME	SegName
set of associated workstations	ASSOCWSIDS	***P*** WsId
current segment attributes	CURSEGATTRS	SegAttrName ⇸ SegAttrValue

11.2.9 Stencil state list (STSL)

stencil name	STENCILNAME	StencilName
current stencil attributes	CURSTENCILATTRS	StencilAttrName ⇸ StencilAttrValue

11.3 Initial values of state lists and description tables

11.3.1 Operating state list

In this subclause, the initial values of GKS data structures are defined. The left-hand column of the tables below contain the description of the entry. The right-hand column is the corresponding initial value.

GKS operating state	GKCL
picture part store operating state	PPCL
segment operating state	SGCL
stencil operating state	STCL
tiling operating state	TLCL

11.3.2 GKS description table

All initial values are implementation dependent.

11.3.3 GKS state list

route direction	NDC
scissor mode	LOCUS
set of open workstations	empty
set of active workstations	empty
set of open audits	empty
set of open playbacks	empty
input queue	empty
font index mapping	default from GDT
current primitive attributes	
PICK IDENTIFIER	language binding dependent
NAMESET	empty
SCISSOR SET	(*scissorid*, (CLIP,{(0,1,0,1)},NOSHIELD,empty)) The value of *scissorid* is language binding dependent
GLOBAL TRANSFORMATION	Identity
LOCAL TRANSFORMATION	Identity
PATTERN SIZE	(1, 1)
PATTERN REFERENCE POINT	(0, 0)
TEXT HEIGHT	0.01

TEXT UP VECTOR	(0, 1)
TEXT SKEW ANGLE	0
TEXT PATH	RIGHT
TEXT ALIGNMENT	(NORMAL, NORMAL)
CURVE INDEX	1
CURVE ASFS	All INDIVIDUAL
CURVETYPE	1
CURVEWIDTH SCALE FACTOR	1.0
CURVE COLOUR SPECIFIER	indirect <<1>>
MARKER INDEX	1
MARKER ASFS	All INDIVIDUAL
MARKERTYPE	3
MARKERSIZE SCALE FACTOR	1.0
MARKER COLOUR SPECIFIER	indirect <<1>>
AREA INDEX	1
AREA ASFS	All INDIVIDUAL
INTERIOR STYLE	HOLLOW
INTERIOR STYLE INDEX	1
INTERIOR COLOUR SPECIFIER	indirect <<1>>
EDGE FLAG	OFF
EDGETYPE	1
EDGEWIDTH SCALE FACTOR	1.0
EDGE COLOUR SPECIFIER	indirect <<1>>
CHARACTER INDEX	1
CHARACTER ASFS	All INDIVIDUAL
CHARACTER FONT AND PRECISION	(1, STRING)
CHARACTER EXPANSION FACTOR	1.0
CHARACTER SPACING	1.0
CHARACTER COLOUR SPECIFIER	indirect <<1>>
current normalization transformation number	0
for each normalization transformation:	
window	(0,1,0,1)
viewport	(0,1,0,1)
viewport input priorities	<0..63>
0 highest priority	
63 lowest (maximum normalization transformation number)	
name of the open picture part	undefined
set of picture part names in use	empty
name of the open segment	undefined
set of segment names in use	empty
set of segment state lists	empty
name of the open stencil	undefined
set of stencil names in use	empty
set of stencil state lists	empty
current contour attributes	
STYLE	SOLID
WIDTH	0.01
CAP	ROUNDED
JOIN	ROUND
name of the open tiling	undefined
set of tiling names in use	empty

11.3.4 Workstation state list

Initial values are taken from the workstation description table except for the following entries.

current workstation window	(0,1,0,1)
current workstation viewport	(0,xd,0,yd)
where (xd,yd) is the display space size in device coordinates from the workstation description table	
current visual effects state	ALLOW
selection criteria	
display	REJECTALL
highlighting	REJECTALL
detectability	REJECTALL
set of stored segments for this workstation	empty
logical input device operating modes all devices	REQUEST
view representations	
selection criterion view 0	SELECTALL
selection criterion other views	REJECTALL
view orientation	Identity
view mapping	Identity
view scissor	NOCLIP,empty,NOSHIELD,empty
view priorities	<0..15>
0 highest	
15 lowest	

11.3.5 Generic workstation description table

All initial values of a generic workstation description table are implementation dependent. When a workstation is opened, the state of all specific workstation description table entries for that workstation is UNCHANGED.

11.3.6 Error state list

error state	OFF
error file	implementation dependent
input device causing queue overflow	undefined

11.3.7 Segment state list

set of associated workstations	workstations active at CREATE SEGMENT
current segment attributes	
VISIBILITY	VISIBLE
HIGHLIGHTING	NORMAL
DETECTABILITY	UNDETECTABLE
SEGMENT TRANSFORMATION	1,0,0:0,1,0
SEGMENT PRIORITY	0.0

11.3.8 Stencil state list

current stencil attributes	undefined

12 Workstation independent functions

12.1 Control functions

OPEN GKS 6.1

In	error file	ErrFile

The operating state list entry 'GKS operating state' is set to GKOP to indicate GKS is open. The GKS state list is allocated and initialized as indicated in clause 11.2. The GKS description table and the generic workstation description tables are made available. The entry 'error file' in the GKS error state list is set to the value specified by the parameter.

CLOSE GKS 6.1

none

The operating state list entry 'GKS operating state' is set to GKCL to indicate GKS is closed. The GKS description table, GKS state list, error state list and the workstation description tables become unavailable. GKS can be reopened by invoking the function OPEN GKS.

SAVE GKS STATE LIST 6.1

In	set of entry names	$\mathbb{P}$ GSLEntName
Out	entry values	GSLEntName $\pfun$ GSLEntValue

The set of entry names in the GKS state list are returned to the application as a set of values for storage.

RESTORE GKS STATE LIST 6.1

In	entry values	GSLEntName $\pfun$ GSLEntValue

The entry values previously stored by the application are restored in the GKS state list.

ESCAPE 6.10, 7.2

In	specific escape function identification	EscapeFuncId
In	escape input data record	EscapeInData
Out	escape output data record	EscapeOutData

The specified non-standard specific escape function is invoked. The form of the escape data records and which of them are used may vary for different functions. The following rules govern the initial definition of a specific escape function:

a) the GKS concepts (see clause 5) are not violated;

b) the GKS state list is not altered;

c) the function does not generate graphical output;

d) any side effects are well documented.

Specific escape functions may apply to more than one workstation, for example all open workstations. The escape input data record may include a workstation identifier where this is required. Examples of specific escape functions are:

e) local control of a frame buffer;

f) use of specialist hardware.

Specific escape function identifications are registered in the ISO International Register of Graphical Items, which is maintained by the Registration Authority (see 4.2). When a specific escape function has been approved by ISO/IEC, the specific escape function identification will be assigned by the Registration Authority.

EMERGENCY CLOSE GKS 6.9

none

The operating state list entry 'GKS operating state' is set to GKCL. GKS is closed due to an emergency. All open workstations are closed and GKS is closed. This function can be called even if there has already been an error. If GKS is already closed, no action is taken.

ERROR HANDLING 6.9

In	error number as listed in annex B	ErrNum
In	identification of the GKS function which caused the error detection	FuncName
In	error file	ErrFile

The ERROR HANDLING function is invoked by GKS in any of the error situations listed in annex B. The standard function just invokes the ERROR LOGGING function with the same parameters. The ERROR HANDLING function can be replaced by an application program supplied function to allow specific reaction to some error situations.

ERROR LOGGING 6.9

In	error number as listed in annex B	ErrNum
In	identification of the GKS function which caused the error detection	FuncName
In	error file	ErrFile

The ERROR LOGGING function appends an error message and identification of the GKS function that caused the error to the error file and then returns to the invoking function.

ROUTE 6.11

In	direction	RouteDir

The GKS state list entry 'route direction' is set to the value specified by the parameter.

12.2 Output functions

CREATE OUTPUT PRIMITIVE 6.2.2, 8.7.6

In	primitive parameters	PrimParam

A primitive of the specified type is created. The current values of the primitive's attributes as given by the 'current primitive attributes' entry in the GKS state list (see 11.2.4) are bound to the primitive.

12.3 Design output functions

OPEN STENCIL 8.7.2

In	stencil name	StencilName
In	inside rule	InsideRule

The operating state list entry 'stencil operating state' is set to STOP to indicate a stencil is open. The stencil name is recorded as the 'name of the open stencil' in the GKS state list and entered into the 'set of stencil names in use' entry in the GKS state list (see 11.2.4). The stencil state list is allocated and initialized as indicated in 11.3.8. All subsequent stencil instantiations until the next CLOSE STENCIL will become part of the stencil definition.

CLOSE STENCIL 8.7.2, 8.7.3

none

Any undefined attributes of the open stencil are set to default values. The operating state list entry 'stencil operating state' is set to STCL to indicate no stencil is open. The 'name of the open stencil' entry in the GKS state list (see 11.2.4) becomes unavailable for inquiry. Instance names created whilst the stencil was open are removed from the 'set of stencil names in use' entry in the GKS state list.

RENAME STENCIL 8.7.6

In	old stencil name	StencilName
In	new stencil name	StencilName

The specified stencil is renamed. If old stencil name is the name of the open stencil, the 'name of the open stencil' entry in the GKS state list is set to new stencil name. The occurrrence of old stencil name in the 'set of stencil names in use' entry in the GKS state list (see 11.2.4) is replaced by new stencil name. The old stencil name may be reused by the application program.

DELETE STENCIL 8.7.6

In	stencil name	StencilName

The specified stencil is deleted from stencil store. The stencil name is removed from the 'set of stencil names in use' entry in the GKS state list (see 11.2.4). The stencil state list is deallocated. The stencil name may be reused by the application program.

CREATE STENCIL FROM BOUNDARY 8.7.2

In	stencil name	StencilName
In	inside rule	InsideRule
In	boundary	SeqBoundary

A stencil with the specified name is created from the specified boundary. Points inside and outside the stencil are determined by the inside rule. The stencil state list is allocated and initialized as indicated in 11.3.8. The stencil name is entered into the 'set of stencil names in use' entry in the GKS state list (see 11.2.4).

CREATE STENCIL FROM CONTOURS 8.7.2, 8.7.6

In	stencil name	StencilName
In	inside rule	InsideRule
In	sequence of paths	SeqPaths

A stencil with the specified name is created from the specified sequence of paths using the 'current contour attributes' entry in the GKS state list. The stencil name is entered into the 'set of stencil names in use' entry in the GKS state list (see 11.2.4). The stencil state list is allocated and initialized as indicated in 11.3.8.

SET CONTOUR ATTRIBUTE 8.7.2

In	contour attribute value	ContourAttrValue

The 'current contour attributes' entry in the GKS state list corresponding to the specified contour attribute value is set to the value specified by the parameter.

SET STENCIL ATTRIBUTE 8.7.3

In	stencil name	StencilName
In	stencil attribute value	StencilAttrValue

The value of the 'current stencil attributes' entry in the stencil state list of the specified stencil corresponding to the specified attribute name is set to the specified attribute value.

GET STENCIL ATTRIBUTE 8.7.3

In	stencil name	StencilName
In	stencil attribute name	StencilAttrName
Out	error indicator	ErrInd
Out	attribute value	StencilAttrValue

The value attribute of the specified stencil attribute in the 'current stencil attributes' entry of the stencil state list of the specified stencil is returned and the error indicator is returned as 0, if the value of the attribute is defined.

If the information is not available, the error indicator is set to one of the following error numbers to indicate the reason for non-availability:

1001 GKS not open
1028 Specified stencil does not exist
1064 Value of stencil attribute is undefined

INSTANCE STENCIL 8.7.2, 8.7.4, 8.7.6

In	stencil name	StencilName
In	instance name	StencilName
In	stencil transformation matrix	StencilTran
In	instance specifier	InstanceSpec

An instance of the specified stencil is created with the specified instance name in the open stencil. The instance name is entered into the 'set of stencil names in use' entry in the GKS state list (see 11.2.4).

INSTANCE STENCIL ALONG PATH 8.7.4, 8.7.6

In	stencil name	StencilName
In	instance name	StencilName
In	start map	LCPoint
In	end map	LCPoint
In	instance number	InstanceNum
In	path definition	Path

An instance of the specified stencil is created along the specified path. The instance name is entered into the 'set of stencil names in use' entry in the GKS state list (see 11.2.4).

INSTANCE STENCIL SEQUENCE ALONG PATH 8.7.4, 8.7.6

In	instance sequence specifier	InstanceSeqSpec
In	path specifier	PathSpec
In	stencil transformation matrix	StencilTran

The instances of the specified stencil sequence are created along the specified path within the open stencil. The instance name is entered into the 'set of stencil names in use' entry in the GKS state list (see 11.2.4).

OPEN TILING 8.7.5

In	tiling name	TilingName

The operating state list entry 'tiling operating state' is set to TLOP indicating that a tiling is open. The tiling name is recorded as the 'name of the open tiling' in the GKS state list and entered into the 'set of tiling names in use' in the GKS state list (see 11.2.4). All subsequent tiling component definitions until the next CLOSE TILING will become part of the tiling.

CLOSE TILING 8.7.5

none

The operating state list entry 'tiling operating state' is set to TLCL indicating that no tiling is open. The 'name of the open tiling' entry in the GKS state list becomes unavailable for inquiry.

RENAME TILING 8.7.6

In	old tiling name	TilingName
In	new tiling name	TilingName

The specified tiling is renamed. If old tiling name is the name of the open tiling, the 'name of the open tiling' in the GKS state list is set to new tiling name. The occurrrence of old tiling name in the 'set of tiling names in use' entry in the GKS state list (see 11.2.4) is replaced by new tiling name. The old tiling name may be reused by the application program.

DELETE TILING 8.7.6

In	tiling name	TilingName

The specified tiling is deleted from tiling store. The tiling name is removed from the 'set of tiling names in use' entry in the GKS state list (see 11.2.4). The tiling name may be reused by the application program.

CREATE TILING COMPONENT 8.7.5

In	primitive parameters	TilingParam
In	origin	LCPoint
In	replication technique	RepTech

The tiling component derived from the specified primitive parameters is defined relative to the origin specified with replication defined by the replication technique. The world coordinate values used in the tiling definition form the local coordinate system of the defined tiling component.

12.4 Output attributes

SET PRIMITIVE ATTRIBUTE 6.2.2

In	primitive attribute value	PrimAttrValue

The value of the 'current primitive attributes' entry in the GKS state list corresponding to the specified primitive attribute value is set to the value specified by the parameter. Attribute names and their types are listed in 11.1.

Output attributes **GKS functions**

ADD SET OF NAMES TO NAMESET 6.2.2

In set of names NameSet

The value of the current nameset in the 'current primitive attributes' entry in the GKS state list has the specified set of names added to it.

REMOVE SET OF NAMES FROM NAMESET 6.2.2

In set of names NameSet

The value of the nameset in the 'current primitive attributes' entry in the GKS state list has specified set of names removed from it.

ADD SET OF SCISSORS TO SCISSOR SET 6.2.2

In set of scissors ScissorSet

The specified set of scissors is added to the value of the current scissor set in the 'current primitive attributes' entry in the GKS state list. If the specified set of scissors contains any scissor identifiers which are also contained in the current scissor set attribute, then the new definitions of the scissor associated with each such name replaces the definition in the current scissor set attribute.

REMOVE SET OF SCISSORS FROM SCISSOR SET 6.2.2

In set of scissor identifiers ScissorIdSet

The scissors and scissor identifiers associated with the specified set of scissor identifiers are removed from the current scissor set in the 'current primitive attributes' entry in the GKS state list.

12.5 Normalization transformation functions

SET WINDOW AND VIEWPORT 6.3, 9.2

In normalization transformation number NormTranNum1
In window limits WCWin
In viewport limits NDCVp

The window and viewport limits entries of the specified normalization transformation in the 'normalization transformations' entry in the GKS state list are set to the values specified by the parameters.

SET VIEWPORT INPUT PRIORITY 9.2, 9.3.1

In normalization transformation number NormTranNum
In reference transformation number NormTranNum
In relative priority Pri

The viewport input priority of the specified normalization transformation in the 'viewport input priorities' entry in the GKS state list is set to the next higher or next lower priority relative to the reference transformation according to the specified relative priority. If the specified transformation number is the same as the reference transformation number, the function has no effect.

SET SCISSOR MODE 6.2.2

In scissor mode ScissorMode

The 'scissor mode' entry in the GKS state list is set to the value specified by the parameter.

SET NORMALIZATION TRANSFORMATION NUMBER 6.3

In normalization transformation number NormTranNum

The 'current normalization transformation number' entry in the GKS state list is set to the value specified by the parameter.

12.6 NDC picture functions

DELETE PRIMITIVES FROM NDC PICTURE 6.5.2

In selection criterion SelectCrit

Primitives in the NDC picture whose NAMESET attributes satisfy the specified selection criterion are deleted from the NDC picture.

ADD SET OF NAMES TO NDC PICTURE 6.2.2, 6.5.2, 6.6

In set of names NameSet
In selection criterion SelectCrit

The specified set of names is added to the NAMESET attributes of all primitives in the NDC picture which satisfy the specified selection criterion.

REMOVE SET OF NAMES FROM NDC PICTURE 6.2.2, 6.5.2, 10.9

In set of names NameSet
In selection criterion SelectCrit

The specified set of names is removed from the NAMESET attributess of all primitives in the NDC picture which satisfy the selection criterion.

SET NDC PICTURE PRIMITIVE ATTRIBUTE 6.2.2, 6.5.2

In selection criterion SelectCrit
In attribute value PrimAttrValue

Primitives in the NDC picture whose NAMESET attributes satisfy the selection criterion and which have the attribute, specified by the attribute value, have the attribute value changed to the one specified.

ADD SET OF SCISSORS TO NDC PICTURE 6.2.2, 6.5.2

In set of scissors ScissorSet
In selection criterion SelectCrit

The specified set of scissors is added to the SCISSOR SET attribute of all primitives in the NDC picture which satisfy the specified selection criterion. If the SCISSOR SET attribute of a selected primitive already contains scissors corresponding to any of the scissor identifiers in the specified scissor set, then the new definition of the scissor associated with each such name replaces the definition in the SCISSOR SET attribute.

REMOVE SET OF SCISSORS FROM NDC PICTURE 6.2.2, 6.5.2

In set of scissor identifiers ScissorIdSet
In selection criterion SelectCrit

The scissors and scissor identifiers corresponding to the specified set of scissor identifiers are removed from the SCISSOR SET attribute of all primitives in the NDC picture which satisfy the specified selection criterion.

REORDER NDC PICTURE 6.5.2

In	source selection criterion	SelectCrit
In	reference selection criterion	SelectCrit
In	relative position	Position

The subsequence of primitives in the NDC picture which satisfy the source selection criterion is moved to the specified relative position in the NDC picture with respect to the remaining subsequence of primitives which satisfy the reference selection criterion.

If the relative position is defined as BACK, the subsequence of primitives satisfying the source selection criterion is moved to be in front of the first position in the remaining subsequence of primitives satisfying the reference selection criterion.

If the relative position is FRONT, the subsequence of primitives satisfying the source selection criterion is moved to be behind the last primitive in the subsequence of primitives satisfying the reference selection criterion.

12.7 Metafile functions

COPY NDC PICTURE TO NDC METAFILE 6.5.3

In	metafile specifier	NDCMfSpec
In	picture identifier	PicId
In	selection criterion	SelectCrit
In	set of names	NameSet

Primitives in the NDC picture which satisfy the specified selection criterion are stored in the specified metafile. The picture is given the specified picture identifier in the metafile. The specified set of names is added to the NAMESET attribute of each primitive in the specified metafile.

COPY NDC METAFILE PICTURE TO NDC PICTURE 6.5.3

In	metafile specifier	NDCMfSpec
In	picture identifier	PicId
In	set of names	NameSet

The specified picture in the specified metafile is added to the NDC picture. The specified set of names is added to the NAMESET attribute of each primitive added to the NDC picture.

12.8 Picture part store functions

OPEN PICTURE PART 6.4.1

In	picture part name	PicPartName

The operating state list entry 'picture part operating state' is set to PPOP to indicate a picture part is open. The picture part name is recorded as the 'name of the open picture part' in the GKS state list (see 11.2.4). The picture part name is entered into the 'set of picture part names in use' entry in the GKS state list (see 11.2.4). All subsequent output primitives until the next CLOSE PICTURE PART will be collected in sequence into this picture part. The current attributes are bound to the primitives. The primitives are not added to the NDC picture.

CLOSE PICTURE PART 6.4.1, 6.4.2

none

The operating state list entry 'picture part operating state' is set to PPCL to indicate a picture part is not open. Primitives are no longer added to the previously open picture part. The 'name of the open picture part' in the GKS state list (see 11.2.4) becomes unavailable for inquiry. The picture part is added to the picture part store.

ARCHIVE PICTURE PART 6.4.4

In	picture part name	PicPartName
In	archive specifier	ArSpec
In	archive name of picture part	ArName

The specified picture part in the picture part store is stored in the specified archive. The archive name of picture part is given to the picture part.

RETRIEVE PICTURE PART FROM ARCHIVE 6.4.4

In	archive specifier	ArSpec
In	archive name of picture part	ArName
In	picture part name	PicPartName

The picture part with archive name of picture part is retrieved from the specified archive and added to picture part store with the specified picture part name. The picture part name is entered into the 'set of picture part names in use' entry in the GKS state list (see 11.2.4).

REOPEN PICTURE PART 6.4.2

In	picture part name	PicPartName

The specified picture part is reopened. The operating state list entry 'picture part operating state' is set to PPOP to indicate a picture part is open. The picture part name is recorded as the 'name of the open picture part' in the GKS state list (see 11.2.4). All subsequent output primitives until the next CLOSE PICTURE PART will be collected in sequence into this picture part after the initial contents. The current attributes are bound to the primitives. The primitives are not added to the NDC picture.

APPEND PICTURE PART 6.4.2

In	source picture part name	PicPartName
In	sink picture part name	PicPartName
In	global transformation matrix	Matrix23
In	global transformation mode	TransMode
In	local transformation matrix	Matrix23
In	local transformation mode	TransMode
In	set of names	NameSet

Primitives in the picture part specified by source picture part name are appended to the picture part specified by sink picture part name. The specified global and local transformation matrices replace, or pre or post concatenate the transformation attributes of the primitives in the source picture part, depending on the transformation modes specified. If the transformation attribute of the primitive is T and the specified transformation matrix is P, the primitive added to the sink picture part will have GLOBAL TRANSFORMATION or LOCAL TRANSFORMATION attribute:

REPLACE:	P
PRE:	$T \times P$
POST:	$P \times T$

The specified set of names is added to the NAMESET attributes bound to each of the source primitives as they

are appended to the sink picture part.

RENAME PICTURE PART 6.4.2

In	old picture part name	PicPartName
In	new picture part name	PicPartName

The specified picture part is renamed. If old picture part name is the value of the 'name of the open picture part' entry in the GKS state list, the 'name of the open picture part' entry in the GKS state list is set to new picture part name. The occurrence of old picture part name in the 'set of picture part names in use' entry in the GKS state list (see 11.2.4) is replaced by new picture part name. The old picture part name may be reused by the application program.

DELETE PICTURE PART 6.4.2

In	picture part name	PicPartName

The specified picture part is deleted from picture part store. The picture part name may be reused by the application program. The picture part name is removed from the 'set of picture part names in use' entry in the GKS state list (see 11.2.4).

COPY PICTURE PART FROM PICTURE PART STORE 6.4.2, 6.4.3

In	picture part name	PicPartName
In	selection criterion	SelectCrit
In	global transformation matrix	Matrix23
In	global transformation mode	TransMode
In	local transformation matrix	Matrix23
In	local transformation mode	TransMode
In	set of names	NameSet
In	scissor select	ScissorSelect

Primitives in the specified picture part in picture part store which satisfy the specified selection criterion are added to the NDC picture or backdrop. The specified global and local transformation matrices replace, or pre or post concatenate the transformation attributes of the primitive depending on the transformation modes specified. If the transformation attribute of the primitive is T and the specified transformation matrix is P, the primitive added to the NDC picture will have GLOBAL TRANSFORMATION or LOCAL TRANSFORMATION attribute:

REPLACE:	P
PRE:	$T \times P$
POST:	$P \times T$

The specified set of names is added to the NAMESET attributes bound to each of the primitives in the picture part. If scissor select is LEAVE, the scissor set attribute bound to the primitives is unchanged. If scissor select is CHANGE, the current value of the SCISSOR SET attribute replaces the SCISSOR SET attribute bound to each of the primitives added.

COPY NDC PICTURE TO PICTURE PART STORE 6.5.2

In	selection criterion	SelectCrit
In	picture part name	PicPartName
In	set of names	NameSet

Primitives in the NDC picture which satisfy the specified selection criterion are copied to picture part store as the specified picture part. The specified set of names is deleted from the NAMESET attributes bound to each of the primitives in the picture part. The picture part name is entered into the 'set of picture part names in use'

entry in the GKS state list (see 11.2.4).

12.9 Input functions

SET LOGICAL INPUT DEVICE MODE 6.7.1, 6.7.3, 7.13.2

In	logical input device identifier	DevId
In	operating mode	OpMode

The specified logical input device is set to the specified operating mode. Depending on the specified operating mode, an interaction with the given device may begin or end. The input device state defined by operating mode is stored in the workstation state list entry 'logical input device operating modes' for the given device.

REQUEST INPUT 6.7.1, 6.7.2, 6.7.3, 7.13, 9.2, 9.3

In	logical input device identifier	DevId
Out	status	InStatus
Out	logical input value	InValue

GKS performs a REQUEST on the specified logical input device. If the break facility is invoked by the operator, the status NONE is returned; otherwise OK is returned together with the logical input value.

SAMPLE INPUT 6.7.1, 6.7.2, 6.7.3, 7.13, 9.2, 9.3

In	logical input device identifier	DevId
Out	logical input value	InValue

The current logical input value of the specified logical input device is returned.

AWAIT INPUT 6.7.1, 6.7.2, 6.7.3, 6.7.4, 7.13, 9.2, 9.3

In	timeout	Timeout
Out	set of logical input device identifications and logical input values	DevIdValueSet

If the 'input queue' entry in the GKS state list (see 11.2.4) is empty, GKS is set into a wait state until an input event is written into the queue or the time specified in the timeout parameter has elapsed. If a timeout occurs and there is still no entry in the queue, a NONE value is returned for device identification. If there is at least one entry in the queue, the oldest event is returned. The logical input device identifier is returned together with the corresponding logical input value.
The operation is performed even if input queue overflow (error 1037) has occurred.

A timeout of zero causes an immediate inspection of the queue, and a NONE value for logical input device identification is returned if the queue is empty.

FLUSH DEVICE EVENTS 6.7.4

In	logical input device identifier	DevId

All entries in the 'input queue' entry in the GKS state list (see 11.2.4) from the specified logical input device are removed. The operation is performed even if input queue overflow (error 1037) has occurred.

12.10 Font and glyph functions

SET FONT INDEX MAPPING 8.5.2

In	font index	FontInd
In	font resource name	ISO9541FontName

The specified font index is associated with the specified font resource name in the 'font index mapping' entry in the GKS state list.

GET GLYPH NAME 8.5.2

In	font index	FontInd
In	character code	Char
Out	glyph name	GlyphName

The glyph name for the specified character code in the specified font is returned.

SET CHARACTER CODE 8.5.2

In	font index	FontInd
In	character code	Char
In	glyph name	GlyphName

The specified glyph is associated with the specified character code in the specified font.

12.11 Audit and playback functions

AUDIT 6.12

In	audit identifier	AuditId
In	operation	AuditOps

The operation OPEN requests the operating system to establish a connection defined by the associated audit specifier to the specified audit trail and adds the specified audit identifier to the 'set of open audits' entry in the GKS state list with the recording audit flag set to OFF. The operations BEGIN and END reset the appropriate recording audit flag in the 'set of open audits' entry in the GKS state list to ON and OFF respectively. Corresponding items are added to the specified audit.

The operation CLOSE removes the specified audit identifier from the 'set of open audits' entry in the GKS state list.

WRITE USER RECORD TO AUDIT 6.12

In	audit identifier	AuditId
In	audit user data record	AuditUserData

The audit user data record specified is added to the specified audit.

PLAYBACK 6.12

In	audit identifier	AuditId
In	playback operation	PlaybackOps

The playback operation OPEN requests the operating system to establish a connection defined by the associated audit specifier to the specified audit trail. The specified audit identifier is added to the 'set of open playbacks' entry in the GKS state list.

The playback operation CLOSE removes the specified audit identifier from the 'set of open playbacks' entry in the GKS state list. The connection to the audit trail is released.

READ ITEM FUNCTION NAME FROM AUDIT 6.12

In	audit identifier	AuditId
Out	function name	AuditFuncName

The value of the function name of the current item on the specified playback audit is returned.

READ ITEM FROM AUDIT 6.12

In	audit identifier	AuditId
Out	function name	AuditFuncName
Out	function parameters	FuncParams

The values of the function name and function parameters of the current item on the specified playback audit are returned.

PROCESS AUDIT ITEM 6.12

In	audit identifier	AuditId
In	process operation	ProcessOps

For process operation SKIP, the item pointed at in the specified playback audit is changed to the next on the audit trail.

For process operation DO, the item pointed at is interpreted. For items that are concerned with output, the effect is the same as if the function was executed. For other functions, the interpretation has no effect. In both cases, the item pointed at in the specified playback audit is changed to the next on the specified playback audit.

12.12 Inquiry functions

Inquiry functions return values from the various state lists. The data types of the values and the default values of the state list entries are summarized in clause 11. Errors detected by inquiry functions are reported through an error indicator parameter, see 6.8. The error handling procedure is not called.

INQUIRE OPERATING STATE ENTRY 6.8

In	entry name	OpStEntName
Out	error indicator	ErrInd
Out	operating state entry value	OpStEntValue

The specified entry in the operating state list is returned. If the inquired information is available, the error indicator is returned as 0 and values are returned in the output parameter.

If the inquired information is not available, the values returned in the output parameter are implementation dependent and the error indicator is set to the following error number to indicate the reason for non-availability:

1001 GKS not open

INQUIRE GKS DESCRIPTION TABLE ENTRY 6.8

In	entry name	GDTEntName
Out	error indicator	ErrInd
Out	GKS description table entry value	GDTEntValue

The specified entry in the GKS description table is returned. If the inquired information is available, the error indicator is returned as 0 and values are returned in the output parameter.

If the inquired information is not available, the values returned in the output parameter are implementation dependent and the error indicator is set to the following error number to indicate the reason for non-availability:

1001 GKS not open

INQUIRE GKS STATE LIST ENTRY 6.8

In	entry name	GSLEntName
Out	error indicator	ErrInd
Out	GKS state list entry value	GSLEntValue

The specified entry in the GKS state list is returned. If the inquired information is available, the error indicator is returned as 0 and values are returned in the output parameter.

If the inquired information is not available, the values returned in the output parameter are implementation dependent and the error indicator is set to the following error number to indicate the reason for non-availability:

1001 GKS not open

INQUIRE INPUT QUEUE OVERFLOW 6.7.4, 6.8

Out	error indicator	ErrInd
Out	logical input device identifier	DevId

If the inquired information is available, the error indicator is returned as 0 and a value is returned in the output parameter.

If the input queue has overflowed since OPEN GKS or the last invocation of INQUIRE INPUT QUEUE OVERFLOW, the identification of the logical input device that caused the overflow is returned. The entry is removed from the error state list.

If the inquired information is not available, or the input queue has not overflowed since the last invocation of the function, the value returned identifying the logical input device is implementation dependent and the error indicator is set to one of the following error numbers to indicate the reason:

1001 GKS not open
1038 Input queue not overflowed ever or since the last invocation of INQUIRE INPUT QUEUE OVERFLOW
1039 Input queue overflowed, but associated workstation has been closed

12.13 Utility functions

EVALUATE TRANSFORMATION MATRIX 6.4.3, 10.7

In	fixed point	WCNDCPoint
In	shift vector	WCNDCPoint
In	rotation angle	Rad
In	scale factors	ScaleFactors
Out	transformation matrix	Matrix23

The transformation defined by fixed point, shift vector, rotation angle, and scale factors is evaluated and the result put in transformation matrix. When fixed point and shift vector are specified in WC, the shift vector and fixed point are transformed by the current normalization transformation matrix. The order of transformation is scale, rotate (both relative to the specified fixed point), and shift.

ACCUMULATE TRANSFORMATION MATRIX 6.4.3, 10.7

In	transformation matrix	Matrix23
In	fixed point	WCNDCPoint
In	shift vector	WCNDCPoint
In	rotation angle	Rad
In	scale factors	ScaleFactors
Out	transformation matrix	Matrix23

The transformation defined by fixed point, shift vector, rotation angle, and scale factors is composed with the input transformation matrix and the result put in transformation matrix. When fixed point and shift vector are specified in WC, the shift vector and fixed point are transformed by the current normalization transformation matrix. The order of transformation is scale, rotate (both relative to the specified fixed point), and shift.

12.14 Utility functions for output primitives

EVALUATE CIRCLE 8.4.1

In	centre	WCPoint
In	radius	WCOrdPos
Out	circle	EllipticDisc

The circle specified by the centre and radius is returned in the form of an elliptic disc.

EVALUATE ELLIPSE 8.4.1

In	ellipse specifier	EllipseSpec
Out	ellipse	EllipticDisc

The ellipse specified by the ellipse specifier is returned as an elliptic disc.

EVALUATE CIRCULAR ARC 3 POINT 8.2.1, 8.4.1

In	circular arc specifier	CircularArcSpec
Out	circular arc	ConicSection

The circular arc specified by three points is returned in the form of a conic section.

Utility functions for output primitives — **GKS functions**

EVALUATE CIRCULAR ARC CENTRE — 8.2.1, 8.4.1

In	centre	WCPoint
In	start vector	Vec
In	end vector	Vec
In	sense flag	SenseFlag
In	radius	WCOrdPos
Out	circular arc	ConicSection

The circular arc specified by the parameters is returned as a conic section.

EVALUATE ELLIPTIC ARC — 8.2.1, 8.4.1

In	ellipse specifier	EllipseSpec
In	start vector	Vec
In	end vector	Vec
Out	elliptic arc	ConicSection

The elliptic arc specified is returned as a conic section.

EVALUATE HYPERBOLIC ARC — 8.2.1

In	hyperbola specifier	HyperbolaSpec
In	start vector	Vec
In	end vector	Vec
Out	hyperbolic arc	ConicSection

The hyperbolic arc specified is returned as a conic section.

EVALUATE PARABOLIC ARC — 8.2.1

In	parabola specifier	ParabolaSpec
Out	parabolic arc	ConicSection

The parabolic arc specified is returned as a conic section.

EVALUATE WC ORDINATE — 8.7.2

In	WC ordinate	WCOrd
In	ordinate selector	OrdSelector
Out	NDC value	NDCOrd

The NDC value corresponding to the WC ordinate and ordinate selector is returned.

13 Workstation functions

13.1 Control functions

OPEN WORKSTATION 7.3

In	workstation identifier	WsId
In	workstation specifier	WsSpec
In	workstation type	WsType

GKS requests the operating system to establish a connection defined by the workstation specifier for a workstation characterized by the workstation type. The workstation type defines a generic workstation type and workstation specific information. A specific workstation description table is allocated and initialized for this workstation. The workstation state list is allocated and initialized from the specific workstation description table. The workstation identifier is added to the 'set of open workstations' entry in the GKS state list.

CLOSE WORKSTATION 6.7.4, 7.3

In	workstation identifier	WsId

The workstation state list is deallocated. The workstation identifier is deleted from the 'set of open workstations' entry in the GKS state list. The input queue is flushed of all events from all devices on the workstation being closed. If the 'input device causing queue overflow' entry in the GKS error state list refers to this workstation identifier, then all the contents of that entry become undefined.

The connection to the workstation is released. The display surface need not be cleared when CLOSE WORKSTATION is invoked, but it may be cleared.

SAVE WORKSTATION STATE LIST 7.3

In	workstation identifier	WsId
In	set of entry names	***P*** WSLEntName
Out	entry values	WSLEntName ⇸ WSLEntValue

The set of specified entries in the workstation state list for the specified workstation are returned.

RESTORE WORKSTATION STATE LIST 7.3

In	workstation identifier	WsId
In	entry values	WSLEntName ⇸ WSLEntValue

The set of previously stored entries in the workstation state list are restored for the specified workstation.

GET WORKSTATION STATUS 7.11

In	workstation identifier	WsId
Out	status	WsStatus

The status of the specified workstation is returned.

Control functions **Workstation functions**

REMOVE BACKDROP 7.10, 10.9

In	workstation identifier	WsId

The backdrop on the specified workstation is deleted.

SET REPRESENTATION 7.7, 7.8, 7.9

In	workstation identifier	WsId
In	representation	Rcp

The index specified in the representation is associated with the value specified in the appropriate table on the specified workstation.

SET VIEW PRIORITY 7.5, 9.2, 9.3

In	workstation identifier	WsId
In	view index	ViewInd
In	reference view index	ViewInd
In	relative priority	Pri

The 'view priorities' entry in the workstation state list of the specified workstation is updated as follows. The view priority of the specified view is set to the next higher or lower priority relative to the reference view according to the specified relative priority. If the specified view is the same as the reference view, the function has no effect.

SET VIEW SELECTION CRITERION 7.5

In	workstation identifier	WsId
In	view index	ViewInd
In	selection criterion	SelectCrit

The'view representations' entry in the workstation state list of the specified workstation is updated as follows. The selection criterion for the specified view on the specified workstation is set to the value specified by the parameter.

SET VIEW 7.5, 9.2

In	workstation identifier	WsId
In	view index	ViewInd1
In	view orientation matrix	Matrix23
In	view mapping matrix	Matrix23
In	view scissor	ViewScissor

The'view representation' entry in the workstation state list of the specified workstation is updated as follows. The specified view on the specified workstation is defined by the orientation matrix, mapping matrix and view scissor.

SET WORKSTATION VISUAL EFFECTS 7.11, 10.9

In	workstation identifier	WsId
In	visual effects state	VisEffSt

The 'current visual effects state' entry in the workstation state list of the specified workstation is set to the value specified by the parameter.

SET WORKSTATION WINDOW AND VIEWPORT 7.6

In	workstation identifier	WsId
In	workstation window limits	LDCWin
In	workstation viewport limits	DCVp

The 'current workstation window' and 'current workstation viewport' entries in the workstation state list of the specified workstation are set to the values specified by the parameters.

DEFINE LOGICAL INPUT DEVICE 7.13.3

In	logical input device identifier	DevId
In	measure sequence	SeqMeasureId
In	trigger set	TriggerSet
In	initial value	InitValue

The 'device compositions' and 'logical input device initial values' entries in the workstation description table of the specified workstation defined by the logical input device identifier are updated as follows. The measure sequence and trigger set of the specified logical input device are set to the values specified by the parameters. The measure sequence specifies the measures from which the measure process of the logical input device is composed. The logical input values returned by the logical input device consist of a sequence of measure values in the same order. The initial value for the specified logical input device is stored in the corresponding workstation state list entry.

INITIALIZE LOGICAL INPUT DEVICE 7.13.2, 9.4

In	logical input device identifier	DevId
In	initial value component	InitValue

The specified component of the initial value for the specified logical input device is stored in the 'logical input device initial values' entry in the workstation state list entry.

SET WORKSTATION SELECTION CRITERION 7.4

In	workstation identifier	WsId
In	selection type	SelectCritType
In	selection criterion	SelectCrit

The 'selection criteria' entry of the workstation state list of the specified workstation is updated as follows. The specified selection criterion on the specified workstation is set to the value specified by the parameter.

COPY REALIZED PICTURE TO REALIZED METAFILE 7.12

In	workstation identifier	WsId
In	metafile specifier	RealizMfSpec
In	picture identifier	PicId

Primitives in the realized picture of the specified workstation have their identification attributes removed and are then stored with the specified picture identifier in the specified metafile.

COPY BLANK REALIZED PICTURE TO REALIZED METAFILE 7.12

In	workstation identifier	WsId
In	metafile specifier	RealizMfSpec
In	picture identifier	PicId

A blank realized picture is stored with the specified picture identifier in the specified metafile.

This can be used to insert blank frames on media such as film without having to empty the realized picture.

COPY REALIZED METAFILE PICTURE TO BACKDROP 7.12

In	workstation identifier	WsId
In	metafile specifier	RealizMfSpec
In	picture identifier	PicId

The specified picture in the specified metafile is added to the backdrop of the specified workstation.

MESSAGE 7.14

In	workstation identifier	WsId
In	message	CharString

The message function:

a) may display a message at an implementation dependent location on the workstation viewport or on some separate device associated with the workstation;

b) shall not alter the GKS state list;

c) may affect the workstation in a purely local way (for example, requesting the operator to change paper). Possible effects on the execution of the application program or on subsequent commands sent to the workstation by GKS are stated explicitly in the implementation dependencies manual.

13.2 Inquiry functions

The inquiry functions that retrieve values from the workstation state lists have an input parameter of the type SetReal that may take the following values:

a) SET: the values returned are those provided by the application program.

b) REALIZED: the values returned are those used by the workstation when the actual values are mapped to the available values in the workstation.

Inquiries for predefined representations in the workstation description table (see 11.2.6) have no such parameter unlike the corresponding inquiries for the representations in the workstation state list (see 11.2.5). The values of predefined representations are available on the workstation. Thus all values returned from a predefined representation are such that, if used by an application program to set a representation, a subsequent inquiry for that representation in the workstation state list would return the same values whether SET or REALIZED was specified.

INQUIRE WORKSTATION STATE LIST ENTRY 6.8,7.8

In	workstation identifier	WsId
In	entry name	WSLEntName
In	type of returned values	SetReal
Out	error indicator	ErrInd
Out	workstation state list entry value	WSLEntValue

The specified entry in the workstation station list of the specified workstation is returned. If the inquired information is available, the error indicator is returned as 0 and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameter are implementation dependent and the error indicator is set to one of the following error numbers to indicate the reason for non-availability:

1001 GKS not open
1012 Specified workstation not open

INQUIRE GENERIC WORKSTATION DESCRIPTION TABLE ENTRY 6.8,7.2

In	workstation generic type	WsGenType
In	entry name	WDTEntName
Out	error indicator	ErrInd
Out	generic workstation description table entry value	WDTEntValue

The specified entry in the generic workstation description table of the specified workstation is returned. If the inquired information is available, the error indicator is returned as 0 and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameter are implementation dependent and the error indicator is set to one of the following error numbers to indicate the reason for non-availability:

1001 GKS not open
1011 Specified workstation type not supported

INQUIRE SPECIFIC WORKSTATION DESCRIPTION TABLE ENTRY 6.8,7.2, 7.8

In	workstation identifier	WsId
In	entry name	WDTEntName
Out	error indicator	ErrInd
Out	entry value	WDTEntValue

The specified entry in the workstation description table of the specified workstation is returned. If the inquired information is available, the error indicator is returned as 0 and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameter are implementation dependent and the error indicator is set to one of the following error numbers to indicate the reason for non-availability:

1001 GKS not open
1012 Specified workstation not open

INQUIRE SPECIFIC WORKSTATION DESCRIPTION TABLE ENTRY STATE 6.8,7.2

In	workstation identifier	WsId
In	entry name	WDTEntName
Out	error indicator	ErrInd
Out	entry state	ChangeFlag

The change flag for the specified entry in the workstation description table of the specified workstation is returned.

If the inquired information is available, the error indicator is returned as 0 and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameter are implementation dependent and the error indicator is set to one of the following error numbers to indicate the reason for non-availability:

1001 GKS not open
1012 Specified workstation not open

RESET SPECIFIC WORKSTATION DESCRIPTION TABLE ENTRY STATE 7.2

In	workstation identifier	WsId
In	entry name	WDTEntName

The change flag for the specified entry in the workstation description table of the specified workstation is set to UNCHANGED.

13.3 Retrieval functions

GET TEXT EXTENT 8.5.10

In	workstation identifier	WsId
In	text position	WCPoint
In	character string	CharString
Out	error indicator	ErrInd
Out	concatenation point	WCPoint
Out	text extent	WCPoint×WCPoint×WCPoint×WCPoint

The concatenation point and text extent for the specified TEXT primitive are returned. If the inquired information is available, the error indicator is returned as 0 and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are implementation dependent and the error indicator is set to one of the following error numbers to indicate the reason for non-availability:

1001 GKS not open
1012 Specified workstation not open

13.4 Viewing utility functions

EVALUATE VIEW ORIENTATION MATRIX 7.5

In	view reference point	NDCPoint
In	view up vector	ViewUpVec
Out	view orientation matrix	Matrix23

A view orientation matrix to perform the specified orientation is returned.

EVALUATE VIEW MAPPING MATRIX 7.5

In	window limits	NDCWin
In	viewport limits	LDCVp
Out	view mapping matrix	Matrix23

A view mapping matrix to perform the specified view mapping is returned.

13.5 Colour utility functions

CONVERT COLOUR 7.8

In	workstation identifier	WsId
In	colour specifier	ColrSpec
In	colour model	ColrModel
Out	colour coordinates	ColrCoords
Out	conversion flag	ConversionFlag

Using the same conversion algorithm as the specified workstation uses, the colour specified by colour specifier is converted to colour coordinates in the specified colour model. The conversion flag may take the following values: DONE indicates a successful conversion; APPROXIMATED indicates some approximation was made during the conversion; NOT CONVERTED indicates that the colour specified could not be represented in the requested colour model or that the workstation does not support either or both of the colour models specified.

When the colour specifier contains a colour index, the colour coordinates are obtained from the workstation colour table of the specified workstation.

14 Segment and workstation activation functions

14.1 Segment functions

The segment facilities in this part of ISO/IEC 7942 are identified as obsolescent features. It is anticipated that they will be considered for deletion for the next edition of GKS.

CREATE SEGMENT 10.5

In segment name SegName

The operating state list entry 'segment operating state' is set to SGOP indicating that a segment is open. The segment state list is allocated and initialized as indicated in 11.3.7. Terms corresponding to the default segment visibility, highlighting and detectability attributes are added to the selection criteria of all the active workstations. The segment name is recorded as the 'name of the open segment', and entered into the 'set of segment names in use', in the GKS state list (see 11.2.4). The current NAMESET attribute entry in the GKS state list is augmented by the mapped segment name. The current value of the GLOBAL TRANSFORMATION attribute in the GKS state list is set to the segment transformation attribute held in the segment state list. The segment name is entered in the 'set of stored segments for this workstation' in the workstation state list for every active workstation. All active workstations are included in the 'set of associated workstations' of the segment state list of the newly opened segment.

CLOSE SEGMENT 10.5

none

A picture part is created whose name is the mapped segment name of the open segment. The operating state list entry 'segment operating state' is set to SGCL indicating no segment is open. The primitives in the NDC picture whose namesets contain the mapped segment name of the open segment are copied into the picture part (COPY NDC PICTURE TO PICTURE PART STORE). The mapped segment name of the open segment and the mapped workstation identifiers of the active workstations are removed from the namesets of primitives stored in the picture part.

The current value of the GLOBAL TRANSFORMATION attribute entry in the GKS state list is set to the identity transformation. The mapped segment name of the open segment is removed from the value of the current NAMESET attribute entry in the GKS state list. The 'name of the open segment' in the GKS state list becomes unavailable for inquiry.

RENAME SEGMENT 10.6

In old segment name SegName
In new segment name SegName

Each occurrence of old segment name in the 'set of stored segments for this workstation' in a workstation state list (see 11.2.5) and in the 'set of segment names in use' in the GKS state list is replaced by new segment name. If old segment name is the name of the open segment, the 'name of the open segment' entry in the GKS state list (see 11.2.4) is set to new segment name.

The corresponding picture part whose name is old mapped segment name is renamed as new mapped segment name by invoking RENAME PICTURE PART. For all primitives in the NDC picture whose nameset includes old mapped segment name, the new mapped segment name is added to their nameset by invoking the function ADD NAME TO NDC PICTURE. In the selection criteria for display, highlighting and detectability, the old mapped segment name is replaced by new mapped segment name by invoking the function SET WORKSTATION SELECTION CRITERION. The old mapped segment name is removed from all primitives in the NDC picture by invoking the function REMOVE SET OF NAMES FROM NDC PICTURE.

DELETE SEGMENT 10.6

In segment name SegName

The segment is deleted. The segment name is removed from each 'set of stored segments for this workstation' entry (in the workstation state lists (see 11.2.5)) which contains it and from the 'set of segment names in use' entry in the GKS state list. The segment's state list is deallocated. The segment name may be reused.

The picture part whose name is the mapped segment name is deleted by invoking the function DELETE PICTURE PART.

All primitives in the NDC picture whose nameset includes the mapped name segment are deleted by invoking the function DELETE PRIMITIVES.

The appropriate term containing the mapped segment name is removed from the selection criteria for display, highlighting and detectability on all workstations with which the segment was associated.

DELETE SEGMENT FROM WORKSTATION 10.6

In workstation identifier WsId
In segment name SegName

The segment is deleted from the specified workstation. The segment name is removed from the 'set of stored segments for this workstation' entry in the workstation state list. The workstation identifier is removed from the 'set of associated workstations' in the segment state list.

All primitives in the NDC picture whose nameset includes the mapped workstation identifier and mapped segment name have their nameset revised so as to delete the mapped workstation identifier by invoking the function REMOVE SET OF NAMES FROM NDC PICTURE.

The appropriate term containing the mapped segment name is removed from the selection criterion for display, highlighting and detectability on the specified workstation.

If the 'set of associated workstations' entry in the segment state list becomes empty, the effect of DELETE SEGMENT (see above) is performed.

ASSOCIATE SEGMENT WITH WORKSTATION 10.8

In workstation identifier WsId
In segment name SegName

The segment name is added to the 'set of stored segments for this workstation' in the workstation state list. The workstation identifier is included in the 'set of associated workstations' in the segment state list.

Appropriate selection criteria for display, highlighting and detectability are set, as described for SET SEGMENT ATTRIBUTE below.

For all primitives in the NDC picture whose nameset includes the mapped segment name, the name of the mapped workstation identifier is added to the nameset by invoking the function ADD SET OF NAMES TO NDC PICTURE.

If the NDC picture does not include primitives with the mapped segment name, the equivalent picture part is copied to the NDC picture with the current value of the segment transformation in the 'current segment attributes' entry in the segment state list as the GLOBAL TRANSFORMATION attribute for the primitives in the segment and with the nameset of the primitives augmented by the mapped workstation identifier and mapped segment name.

COPY SEGMENT TO WORKSTATION 10.8

In	workstation identifier	WsId
In	segment name	SegName

The corresponding picture part is dispatched to the router. The GLOBAL TRANSFORMATION attribute of each primitive is set to the current value as defined in the 'current segment attributes' entry in the segment state list. The specified mapped workstation identifier is added to the nameset of each primitive in the picture part by invoking the function COPY PICTURE PART FROM PICTURE PART STORE.

INSERT SEGMENT 10.8

In	segment name	SegName
In	transformation matrix	Matrix23

The corresponding picture part is dispatched to the router. The LOCAL TRANSFORMATION attribute of each primitive is post concatenated (pre multiplied) by the segment transformation and then by the transformation specified as parameter. The GLOBAL TRANSFORMATION attribute of each primitive is set to the current GLOBAL TRANSFORMATION attribute value defined by the 'current primitive attributes' entry in the GKS state list. The namesets of each primitive are augmented by the current NAMESET attribute value from the GKS state list.

The SCISSOR SET attribute of each primitive is set to the current SCISSOR SET attribute value from the GKS state list.

SET SEGMENT ATTRIBUTE 10.7

In	segment name	SegName
In	segment attribute value	SegAttrValue

The 'current segment attributes' entry in the segment state list has the value corresponding to the specified segment attribute value set to the value specified by the parameter.

If the visibility, highlighting or detectability attribute is reset, the selection criterion for each associated workstation is set appropriately (SET WORKSTATION SELECTION CRITERION).

If the segment transformation attribute is reset, the GLOBAL TRANSFORMATION attributes of all the primitives in the NDC picture whose nameset includes the mapped segment name are changed to the specified segment transformation.

If segment priority is reset, the primitives in the NDC picture will be reordered to reflect the new priority order of segments. If the new priority of the specified segment is immediately lower than another segment, the primitives of the specified segment will appear in the NDC picture immediately before the primitives of that segment. Similarly, if the new priority of the specified segment is immediately higher than the priority of another segment, the primitives of the specified segment will appear in the picture immediately after the primitives of that segment. If the new priority of the specified segment is the lowest priority of the set of segments in the NDC picture, the primitives of the specified segment will appear at the start of the NDC picture. Similarly if the new priority of the specified segment is the highest priority of the set of segments in the NDC picture, the primitives of the specified segment will appear at the end of the NDC picture.

14.2 Workstation activation functions

ACTIVATE WORKSTATION 10.2, 10.4

In	workstation identifier	WsId

The name of the mapped workstation identifier is added to the current nameset in the 'current primitive attributes' entry in the GKS state list. The workstation identifier is added to the 'set of active workstations' entry in the GKS state list.

DEACTIVATE WORKSTATION 10.4

In workstation identifier WsId

The name of the mapped workstation identifier is deleted from the current nameset in the 'current primitive attributes entry' in the GKS state list. The workstation identifier is deleted from the 'set of active workstations' entry in the GKS state list.

CLEAR WORKSTATION 7.10, 10.9

In workstation identifier WsId

The name of the mapped workstation identifier is deleted from the namesets of all primitives in the NDC picture. The backdrop is deleted from the specified workstation.

14.3 Utility functions

GET MAPPED SEGMENT NAME 10.1

In segment name SegName

Out mapped segment name GenName

The name derived from the segment name is returned.

GET MAPPED WORKSTATION IDENTIFIER 10.1

In workstation identifier WsId

Out mapped workstation identifier GenName

The name derived from the workstation identifier is returned.

Annex A
(normative)

Function and data type list

A.1 Functions alphabetic

A.2 Functions order of appearance

Control functions (12.1)

Output functions (12.2)

Design output functions (12.3)

Output attributes (12.4)

Normalization transformation functions (12.5)

NDC picture functions (12.6)

Metafile functions (12.7)

Picture part store functions (12.8)

Input functions (12.9)

Font and glyph functions (12.10)

Audit and playback functions (12.11)

Inquiry functions (12.12)

Utility functions (12.13)

Utility functions for output primitives (12.14)

Workstation control functions (13.1)

Inquiry functions (13.2)

Retrieval functions (13.3)

Viewing utility functions (13.4)

Colour utility functions (13.5)

Segment functions (14.1)

Workstation activation functions (14.2)

Utility functions (14.3)

A.3 Data types alphabetic

Annex B
(normative)

Error list

B.1 Function error list

OPEN GKS
1002 GKS already open

CLOSE GKS
1001 GKS not open

SAVE GKS STATE LIST
1001 GKS not open

RESTORE GKS STATE LIST
1001 GKS not open

ESCAPE
1001 GKS not open
1040 Specified escape function is not supported

EMERGENCY CLOSE GKS
none

ERROR HANDLING
none

ERROR LOGGING
none

ROUTE
1001 GKS not open
1006 Segment open

CREATE OUTPUT PRIMITIVE
1001 GKS not open
1050 GDP not supported

OPEN STENCIL
1001 GKS not open
1008 Stencil already open
1029 Specified stencil name already in use

CLOSE STENCIL
1001 GKS not open
1007 Stencil not open

RENAME STENCIL
1001 GKS not open
1030 Old stencil does not exist
1031 New stencil name already in use
1062 Old stencil name and new stencil name are the same

DELETE STENCIL
1001 GKS not open
1008 Stencil already open
1028 Specified stencil does not exist

CREATE STENCIL FROM BOUNDARY
1001 GKS not open
1029 Specified stencil name already in use

CREATE STENCIL FROM CONTOURS
1001 GKS not open
1029 Specified stencil name already in use

SET CONTOUR ATTRIBUTE
1001 GKS not open

SET STENCIL ATTRIBUTE
1001 GKS not open
1028 Specified stencil does not exist

GET STENCIL ATTRIBUTE
none

INSTANCE STENCIL
1001 GKS not open
1007 Stencil not open
1028 Specified stencil does not exist
1032 Instance name same as stencil name already in use

INSTANCE STENCIL ALONG PATH
1001 GKS not open
1007 Stencil not open
1028 Specified stencil does not exist
1032 Instance name same as stencil name already in use

INSTANCE STENCIL SEQUENCE ALONG PATH
1001 GKS not open
1007 Stencil not open
1028 Specified stencil does not exist
1032 Instance name same as stencil name already in use
1058 Edge does not intersect path

OPEN TILING
1001 GKS not open
1010 Tiling already open
1034 Specified tiling name already in use

CLOSE TILING
1001 GKS not open
1009 Tiling not open

RENAME TILING
1001 GKS not open
1035 Old tiling does not exist
1036 New tiling name already in use
1063 Old tiling name and new tiling name are the same

DELETE TILING
1001 GKS not open
1010 Tiling already open
1033 Specified tiling does not exist

CREATE TILING COMPONENT
1001 GKS not open
1009 Tiling not open

SET PRIMITIVE ATTRIBUTE
1001 GKS not open

ADD SET OF NAMES TO NAMESET
1001 GKS not open

REMOVE SET OF NAMES FROM NAMESET
1001 GKS not open

ADD SET OF SCISSORS TO SCISSOR SET
1001 GKS not open

REMOVE SET OF SCISSORS FROM SCISSOR SET
1001 GKS not open

SET WINDOW AND VIEWPORT
1001 GKS not open

SET VIEWPORT INPUT PRIORITY
1001 GKS not open

SET SCISSOR MODE
1001 GKS not open

SET NORMALIZATION TRANSFORMATION NUMBER
1001 GKS not open