INTERNATIONAL STANDARD

ISO
8651-3

First edition
1988-09-15

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION
ORGANISATION INTERNATIONALE DE NORMALISATION
МЕЖДУНАРОДНАЯ ОРГАНИЗАЦИЯ ПО СТАНДАРТИЗАЦИИ

Information processing systems — Computer graphics — Graphical Kernel System (GKS) language bindings —

Part 3 :
Ada

Systèmes de traitement de l'information — Infographie — Système graphique de base (GKS) — Interface langage —

Partie 3 : Ada

Reference number
ISO 8651-3 : 1988 (E)

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

Draft International Standards adopted by the technical committees are circulated to the member bodies for approval before their acceptance as International Standards by the ISO Council. They are approved in accordance with ISO procedures requiring at least 75 % approval by the member bodies voting.

International Standard ISO 8651-3 was prepared by Technical Committee ISO/TC 97, *Information processing systems.*

Users should note that all International Standards undergo revision from time to time and that any reference made herein to any other International Standard implies its latest edition, unless otherwise stated.

ISO 8651 consists of the following parts, under the general title *Information processing systems — Computer graphics — Graphic Kernel System (GKS) language bindings :*

— *Part 1 : FORTRAN*

— *Part 2 : PASCAL*

— *Part 3 : Ada*

Annexes A to G are for information only.

Printed in Switzerland

Contents

This page intentionally left blank

Information processing systems — Computer graphics — Graphical Kernel System (GKS) language bindings —

Part 3 :
Ada

0 Introduction

The Graphical Kernel System (GKS) (ISO 7942) is specified in a language independent manner and needs to be embedded in language dependent layers (language bindings) for use with particular programming languages.

The purpose of this part of ISO 8651 is to define a standard binding for the Ada computer programming language.

1 Scope and field of application

ISO 7942 (GKS) specifies a language independent nucleus of a graphics system. For integration into a programming language, GKS is embedded in a language dependent layer obeying the particular conventions of that language. This part of ISO 8651 specifies such a language dependent layer for the Ada language.

2 References

ISO 7942, *Information processing systems — Computer graphics — Graphical Kernel System (GKS) functional description.*

ISO 8652, *Programming Languages — Ada.*

3 The Ada language binding of GKS

This binding does not assume that the compiler supports any Ada language features which are implementation dependent, but implies that the compiler shall be able to support the declarations contained in this GKS/Ada binding. This binding does not make any assumptions regarding the machine representation of the predefined Ada numeric types.

This binding assumes that the application programmer will supply an error file name and connection identifier that are in an acceptable format for the Ada implementation.

This binding makes no assumptions regarding the format of a string specifying an error file name or connection identifier for devices or metafiles.

3.1 Conformance

This binding incorporates the rules of conformance defined in the GKS Standard (ISO 7942) for GKS implementations, with these additional requirements specifically defined for Ada implementations of GKS.

The following criteria are established for determining conformance or non-conformance of an implementation to this binding:

a) An implementation of GKS in Ada conforms to a level of GKS if it makes visible exactly the declarations for that level of GKS and lower levels of GKS as stated in this binding.

b) The semantics of an implementation shall be those stated in the GKS standard as modified or extended for Ada as stated in this binding document.

c) The package corresponding to the GKS level being implemented shall be an available Ada library unit, with all names as specified by this document.

3.2 Implications of the language

3.2.1 Functional mapping

The functions of GKS are all mapped to Ada procedures. The mapping utilizes a one-to-one correspondence between the GKS functions and Ada procedures, except for the GKS functions Inquire Current Primitive Attribute Values and Inquire Current Individual Attribute Values. These are bound with one Ada procedure for each of the attributes being inquired; in addition, the attributes are bound as a single record.

3.2.2 Implementation and host dependencies

There are a number of implementation and host dependencies associated with the Ada compiler and runtime system used. These will affect the portability of application programs and their use of GKS. The application programmer should follow accepted practices for ensuring portability of Ada programs to avoid introducing problems when rehosting the application on another system. Implementation dependencies include runtime storage management and processor management.

3.2.3 Error handling

The inquiry functions utilize error indicator parameters for the error returns, and do not raise Ada exceptions. The application program must ensure that these error indicators are checked before attempting to access other parameters, since some Ada implementations do not raise an exception if an undefined value is accessed.

The error handling requirements of GKS can be summarized as follows:

1. By default, a procedure named ERROR_HANDLING will be provided that simply reports the error by calling ERROR_LOGGING. This is called from the GKS function that detects the error.

2. The ERROR_HANDLING procedure may be replaced by one defined by the user.

The procedure ERROR_HANDLING is defined as a library subprogram:

```
with GKS_TYPES;
use GKS_TYPES;
procedure ERROR_HANDLING (ERROR_INDICATOR : in ERROR_NUMBER;
                          GKS_FUNCTION    : in STRING;
                          ERROR_FILE      : in STRING
                                            :=DEFAULT_ERROR_FILE);

   -- The procedure ERROR_HANDLING  is defined as a library subprogram, and is not
   -- declared within package GKS.
```

This binding defines two different bodies for this subprogram; each must be supplied by the implementation. The default body is the one required by GKS semantics. It simply calls ERROR_LOGGING and returns. The second body calls ERROR_LOGGING and then raises the exception GKS_ERROR. The GKS function must be written so as not to handle GKS_ERROR (this is a requirement of the implementation). Thus, by Ada rules, the exception will be propagated back to the application program that called the GKS function in which the error was detected.

The means by which the user replaces the default body of either the exception-raising version or another one of his or her choosing is dependent upon the Ada library manager. Some implementations support multiple versions of a body with a single specification or otherwise allow hierarchical libraries with the sharing of common units. In other implementations it may be necessary to duplicate the GKS library for each version of ERROR_HANDLING.

GKS errors are mapped to the single exception GKS_ERROR, declared in the GKS package. The expected style in dealing with errors using exception handling is to provide a handler for the GKS_ERROR exception.

3.2.4 Data mapping

The simple and compound data types of GKS are bound to a variety of Ada scalar and compound types. Constraints on permitted values are reflected where possible in the type definitions. The general correspondence between the GKS data types and Ada binding data types is summarized below:

GKS integers are mapped to Ada integer types.

GKS reals are mapped to Ada floating-point types.

GKS strings are mapped to the predefined Ada type STRING, or to a type providing for variable length strings.

GKS points are mapped to Ada record types.

GKS names are mapped to Ada discrete types.

GKS enumeration types are mapped to Ada enumeration types.

GKS vectors are mapped to Ada record types.

GKS matrices are mapped to Ada array types.

GKS lists (of elements of a particular type) are mapped to an Ada private type declared in an instantiation of the generic GKS_LIST_UTILITIES package.

GKS arrays are mapped to either an unconstrained Ada array type, or to a record type providing for variable length arrays.

GKS ordered pairs are mapped to Ada record types.

GKS data records are mapped to Ada private types. In some cases a set of subprograms for operating on the data records are explicitly defined by this binding. This is because the content and structure of the data record is implementation-dependent. An implementation of GKS may provide other subprograms for manipulating implementation-dependent data records.

3.2.5 Multi-tasking

The Ada language definition provides explicit support for concurrency. The Ada tasking model includes facilities for declaring and allocating tasks, and operations allowing intertask communication and synchronization.

The GKS standard, and hence this binding, neither requires nor prohibits an implementation from protecting against problems which could arise from asynchronous access to the GKS data structures from concurrent tasks. Implementors of GKS should provide information in the user's documentation regarding whether protection against such problems is implemented.

Annex D contains guidelines for implementors who want to support multi-tasking application programs. This annex does not form an integral part of the binding standard, but provides additional information.

3.2.6 Packaging

The GKS standard defines nine levels of graphic functionality, with level 0a as the lowest level and level 2c as the highest level. An implementation of GKS may implement every level individually or as a single system. To support this concept this binding defines nine Ada packages which correspond to each of the GKS levels. Each of these packages is named

```
package GKS is ... end GKS;
```

to provide portability of application programs for levels of GKS. However, the contents of the packages differ depending on the level of GKS that they provide. Each of these packages provides the subprograms defined for its level and all subprograms defined in "lower" levels as described in 5.1 of this binding. Associated with each of these packages is a data type package which provides the type declarations for the appropriate level as defined in 4.2 and the GKS defined exception defined in 4.3.1. These packages are named

```
package GKS_TYPES is ... end GKS_TYPES;
```

The Ada program library facility should be used to provide the levels separation. Thus, an Ada graphics application program which uses GKS would "with" the appropriate GKS packages which provide the subprogram, types, and exceptions for that level by compiling and linking to the corresponding Ada library which contains that level of GKS. For example, an application which uses level 0a would "with" the packages as follows:

```
with GKS;
use  GKS_TYPES;
procedure APPLICATION is
begin
   null;
end APPLICATION;
```

Then the program is compiled and linked to the Ada program library that corresponds to level 0a.

Several additional Ada units are defined in this binding. These are:

- o generic package GKS_COORDINATE_SYSTEM
- o generic package GKS_LIST_UTILITIES

These generic packages support the declaration types in the GKS_TYPES package described above. The GKS_COORDINATE _SYSTEM is a generic package that defines an assortment of types for supporting each of the GKS coordinate systems. GKS_LIST_UTILITIES is also a generic package which provides type declarations and operations for list types which correspond to the GKS list types.

3.2.7 Application program environment

An application program utilizing an Ada implementation of GKS will need to be aware of the environment in which both GKS and the application program(s) reside.

One such interface is the Ada program library. The Ada language requires that the application program have access to the program library in which the GKS software resides. The ISO 8652 Ada Reference Manual does not specify whether there is a single library or multiple libraries, or how access to the libraries is granted, managed, etc. The user's documentation for the GKS implementation should specify where the GKS library exists in the system, and how access to the library is acquired.

Input/Output interfaces are also implementation-dependent, and are required to be described in the user's documentation. Besides the obvious graphics device interface information, interfaces to the file system shall be included in the documentation. Specifically, this includes the interface to the GKS error file and also the metafile storage.

3.2.8 Registration [1)]

The GKS standard reserves certain value ranges for registration as graphical items. The registered graphical items will be bound to Ada (and other programming languages). The registered item bindings will be consistent with the binding presented in the document.

1) For the purpose of this part of ISO 8651 and according to the rules for the designation and operation of registration authorities in the ISO Directives, the ISO Council has designated the National Bureau of Standards (Institute of Computer Sciences and Technology) A266 Technology Building, Gaithersburg, MD, 20899, USA to act as registration authority.

4 Tables

4.1 Procedures

Table 1 - Abbreviations used in Procedure names

ASF	aspect source flag
CHAR	character
ESC	escape
GDP	generalized drawing primitive
GKS	Graphical Kernel System
GKSM	Graphical Kernel System metafile
ID	identifier
INQ	inquire
MAX	maximum
UGDP	unregistered generalized drawing primitive
UESC	unregistered escape
WS	workstation(s)

Table 2 - List of procedures using the abbreviations

ASF	
	INQ_LIST_OF_ASF
	SET_ASF
CHAR	
	INQ_CHAR_BASE_VECTOR
	INQ_CHAR_EXPANSION_FACTOR
	INQ_CHAR_HEIGHT
	INQ_CHAR_WIDTH
	INQ_CHAR_SPACING
	INQ_CHAR_UP_VECTOR
	SET_CHAR_EXPANSION_FACTOR
	SET_CHAR_HEIGHT
	SET_CHAR_SPACING
	SET_CHAR_UP_VECTOR
ESC	
	ESC
	UESC
GDP	
	GDP
	INQ_GDP
	INQ_LIST_OF_AVAILABLE_GDP
	UGDP
GKS	
	CLOSE_GKS
	EMERGENCY_CLOSE_GKS
	INQ_LEVEL_OF_GKS
	OPEN_GKS

Table 2 - List of procedures using the abbreviations

GKSM
- GET_ITEM_TYPE_FROM_GKSM
- READ_ITEM_FROM_GKSM
- WRITE_ITEM_TO_GKSM

ID
- INQ_CURRENT_PICK_ID_VALUE
- SET_PICK_ID

INQ
- INQ_CHAR_BASE_VECTOR
- INQ_CHAR_EXPANSION_FACTOR
- INQ_CHAR_HEIGHT
- INQ_CHAR_WIDTH
- INQ_CHAR_SPACING
- INQ_CHAR_UP_VECTOR
- INQ_CHOICE_DEVICE_STATE
- INQ_CLIPPING
- INQ_COLOUR_FACILITIES
- INQ_COLOUR_REPRESENTATION
- INQ_CURRENT_NORMALIZATION_TRANSFORMATION_NUMBER
- INQ_CURRENT_INDIVIDUAL_ATTRIBUTE_VALUES
- INQ_CURRENT_PICK_ID_VALUE
- INQ_CURRENT_PRIMITIVE_ATTRIBUTE_VALUES
- INQ_DEFAULT_CHOICE_DEVICE_DATA
- INQ_DEFAULT_DEFERRAL_STATE_VALUES
- INQ_DEFAULT_LOCATOR_DEVICE_DATA
- INQ_DEFAULT_PICK_DEVICE_DATA
- INQ_DEFAULT_STRING_DEVICE_DATA
- INQ_DEFAULT_STROKE_DEVICE_DATA
- INQ_DEFAULT_VALUATOR_DEVICE_DATA
- INQ_DISPLAY_SPACE_SIZE
- INQ_DYNAMIC_MODIFICATION_OF_SEGMENT_ATTRIBUTES
- INQ_DYNAMIC_MODIFICATION_OF_WS_ATTRIBUTES
- INQ_FILL_AREA_COLOUR_INDEX
- INQ_FILL_AREA_FACILITIES
- INQ_FILL_AREA_INDEX
- INQ_FILL_AREA_INTERIOR_STYLE
- INQ_FILL_AREA_REPRESENTATION
- INQ_FILL_AREA_STYLE_INDEX
- INQ_GDP
- INQ_INPUT_QUEUE_OVERFLOW
- INQ_LEVEL_OF_GKS
- INQ_LIST_OF_ASF
- INQ_LINETYPE
- INQ_LINEWIDTH_SCALE_FACTOR
- INQ_LIST_OF_AVAILABLE_GDP
- INQ_LIST_OF_AVAILABLE_WS_TYPE
- INQ_LIST_OF_COLOUR_INDICES
- INQ_LIST_OF_FILL_AREA_INDICES

Table 2 - List of procedures using the abbreviations

INQ_LIST_OF_NORMALIZATION_TRANSFORMATION_NUMBER
INQ_LIST_OF_PATTERN_INDICES
INQ_LIST OF_POLYLINE_INDICES
INQ_LIST_OF_POLYMARKER_INDICES
INQ_LIST_OF_TEXT_INDICES
INQ_LOCATOR_DEVICE_STATE
INQ_MAX_LENGTH_OF_WS_STATE_TABLES
INQ_MAX_NORMALIZATION_TRANSFORMATION_NUMBER
INQ_MORE_SIMULTANEOUS_EVENTS
INQ_NAME_OF_OPEN_SEGMENT
INQ_NORMALIZATION_TRANSFORMATION
INQ_NUMBER_OF_SEGMENT_PRIORITIES_SUPPORTED
INQ_NUMBER_OF_AVAILABLE_LOGICAL_INPUT_DEVICES
INQ_OPERATING_STATE_VALUE
INQ_PATTERN_FACILITIES
INQ_PATTERN_HEIGHT_VECTOR
INQ_PATTERN_REFERENCE_POINT
INQ_PATTERN_REPRESENTATION
INQ_PATTERN_WIDTH_VECTOR
INQ_PICK_DEVICE_STATE
INQ_PIXEL
INQ_PIXEL_ARRAY
INQ_PIXEL_ARRAY_DIMENSIONS
INQ_POLYLINE_COLOUR_INDEX
INQ_POLYLINE_FACILITIES
INQ_POLYLINE_INDEX
INQ_POLYLINE_REPRESENTATION
INQ_POLYMARKER_REPRESENTATION
INQ_POLYMARKER_COLOUR_INDEX
INQ_POLYMARKER_INDEX
INQ_POLYMARKER_FACILITIES
INQ_POLYMARKER_SIZE_SCALE_FACTOR
INQ_POLYMARKER_TYPE
INQ_PREDEFINED_COLOUR_REPRESENTATION
INQ_PREDEFINED_FILL_AREA_REPRESENTATION
INQ_PREDEFINED_PATTERN_REPRESENTATION
INQ_PREDEFINED_POLYLINE_REPRESENTATION
INQ_PREDEFINED_POLYMARKER_REPRESENTATION
INQ_PREDEFINED_TEXT_REPRESENTATION
INQ_SEGMENT_ATTRIBUTES
INQ_SET_OF_ACTIVE_WS
INQ_SET_OF_ASSOCIATED_WS
INQ_SET_OF_OPEN_WS
INQ_SET_OF_SEGMENT_NAMES_IN_USE
INQ_SET_OF_SEGMENT_NAMES_ON_WS
INQ_STRING_DEVICE_STATE

Table 2 - List of procedures using the abbreviations

	INQ_STROKE_DEVICE_STATE
	INQ_TEXT_ALIGNMENT
	INQ_TEXT_COLOUR_INDEX
	INQ_TEXT_EXTENT
	INQ_TEXT_FACILITIES
	INQ_TEXT_FONT_AND_PRECISION
	INQ_TEXT_INDEX
	INQ_TEXT_PATH
	INQ_TEXT_REPRESENTATION
	INQ_VALUATOR_DEVICE_STATE
	INQ_WS_CATEGORY
	INQ_WS_CLASSIFICATION
	INQ_WS_CONNECTION_AND_TYPE
	INQ_WS_DEFERRAL_AND_UPDATE_STATES
	INQ_WS_MAX_NUMBER
	INQ_WS_STATE
	INQ_WS_TRANSFORMATION
MAX	
	INQ_MAX_LENGTH_OF_WS_STATE_TABLES
	INQ_MAX_NORMALIZATION_TRANSFORMATION_NUMBER
	INQ_WS_MAX_NUMBERS
WS	
	ACTIVATE_WS
	ASSOCIATE_SEGMENT_WITH_WS
	CLEAR_WS
	CLOSE_WS
	COPY_SEGMENT_TO_WS
	DEACTIVATE_WS
	DELETE_SEGMENT_FROM_WS
	INQ_DYNAMIC_MODIFICATION_OF_WS_ATTRIBUTES
	INQ_LIST_OF_AVAILABLE_WS_TYPE
	INQ_MAX_LENGTH_OF_WS_STATE_TABLES
	INQ_SET_OF_ACTIVE_WS
	INQ_SET_OF_ASSOCIATED_WS
	INQ_SET_OF_OPEN_WS
	INQ_SET_OF_SEGMENT_NAMES_ON_WS
	INQ_WS-CATEGORY
	INQ_WS_CLASSIFICATION
	INQ_WS_CONNECTION_AND_TYPE
	INQ_WS_DEFERRAL_AND_UPDATE_STATES
	INQ_WS_MAX_NUMBER
	INQ_WS_STATE
	INQ_WS_TRANSFORMATION
	OPEN_WS
	REDRAW_ALL_SEGMENTS_ON_WS
	SET_WS_VIEWPORT
	SET_WS_WINDOW
	UPDATE_WS

TABLE 3 - GKS Function Names and Ada Names Ordered Alphabetically by Ada Bound Name

Ada Bound Name	GKS Function
ACCUMULATE_TRANSFORMATION_MATRIX	ACCUMULATE TRANSFORMATION MATRIX
ACTIVATE_WS	ACTIVATE WORKSTATION
ASSOCIATE_SEGMENT_WITH_WS	ASSOCIATE SEGMENT WITH WORKSTATION
AWAIT_EVENT	AWAIT EVENT
CELL_ARRAY	CELL ARRAY
CLEAR_WS	CLEAR WORKSTATION
CLOSE_GKS	CLOSE GKS
CLOSE_SEGMENT	CLOSE SEGMENT
CLOSE_WS	CLOSE WORKSTATION
COPY_SEGMENT_TO_WS	COPY SEGMENT TO WORKSTATION
CREATE_SEGMENT	CREATE SEGMENT
DEACTIVATE_WS	DEACTIVATE WORKSTATION
DELETE_SEGMENT	DELETE SEGMENT
DELETE_SEGMENT_FROM_WS	DELETE SEGMENT FROM WORKSTATION
EMERGENCY_CLOSE_GKS	EMERGENCY CLOSE GKS
ERROR_HANDLING	ERROR HANDLING
ERROR_LOGGING	ERROR LOGGING
ESCAPE	ESCAPE
EVALUATE_TRANSFORMATION_MATRIX	EVALUATE TRANSFORMATION MATRIX
FILL_AREA	FILL AREA
FLUSH_DEVICE_EVENTS	FLUSH DEVICE EVENTS
GDP	GENERALIZED DRAWING PRIMITIVE
GET_CHOICE	GET CHOICE
GET_ITEM_TYPE_FROM_GKSM	GET ITEM TYPE FROM GKSM
GET_LOCATOR	GET LOCATOR
GET_PICK	GET PICK
GET_STRING	GET STRING
GET_STROKE	GET STROKE
GET_VALUATOR	GET VALUATOR
INITIALISE_CHOICE	INTIALISE CHOICE
INITIALISE_LOCATOR	INITIALISE LOCATOR
INITIALISE_PICK	INITIALISE PICK
INITIALISE_STRING	INITIALISE STRING
INITIALISE_STROKE	INITIALISE STROKE
INITIALISE_VALUATOR	INITIALISE VALUATOR
INQ_CHOICE_DEVICE_STATE	INQUIRE CHOICE DEVICE STATE
INQ_CLIPPING	INQUIRE CLIPPING
INQ_COLOUR_FACILITIES	INQUIRE COLOUR FACILITIES
INQ_COLOUR_REPRESENTATION	INQUIRE COLOUR REPRESENTATION

TABLE 3 - GKS Function Names and Ada Names Ordered Alphabetically by Ada Bound Name

Ada Bound Name	GKS Function
INQ_CURRENT_INDIVIDUAL_ATTRIBUTE_VALUES	INQUIRE CURRENT INDIVIDUAL ATTRIBUTE VALUES

The following functions are a one-to-many mapping of the GKS function "inquire current individual attribute values".

INQ_CHAR_EXPANSION_FACTOR
INQ_CHAR_SPACING
INQ_FILL_AREA_COLOUR_INDEX
INQ_FILL_AREA_INTERIOR_STYLE
INQ_FILL_AREA_STYLE_INDEX
INQ_LINETYPE
INQ_LINEWIDTH_SCALE_FACTOR
INQ_LIST_OF_ASF
INQ_POLYLINE_COLOUR_INDEX
INQ_POLYMARKER_COLOUR_INDEX
INQ_POLYMARKER_SIZE_SCALE_FACTOR
INQ_POLYMARKER_TYPE
INQ_TEXT_COLOUR_INDEX
INQ_TEXT_FONT_AND_PRECISION

INQ_CURRENT_NORMALIZATION_TRANSFORMATION_NUMBER	INQUIRE CURRENT NORMALIZATION TRANSFORMATION NUMBER
INQ_CURRENT_PICK_ID_VALUE	INQUIRE CURRENT PICK IDENTIFIER VALUE
INQ_CURRENT_PRIMITIVE_ATTRIBUTE_VALUES	INQUIRE CURRENT PRIMITIVE ATTRIBUTE VALUES

The following functions are a one-to-many mapping of the GKS function "inquire current primitive attribute values".

INQ_CHAR_BASE_VECTOR
INQ_CHAR_HEIGHT
INQ_CHAR_WIDTH
INQ_CHAR_UP_VECTOR
INQ_FILL_AREA_INDEX
INQ_PATTERN_HEIGHT_VECTOR
INQ_PATTERN_REFERENCE_POINT
INQ_PATTERN_WIDTH_VECTOR
INQ_POLYLINE_INDEX
INQ_POLYMARKER_INDEX
INQ_TEXT_ALIGNMENT
INQ_TEXT_INDEX
INQ_TEXT_PATH

INQ_DEFAULT_CHOICE_DEVICE_DATA	INQUIRE DEFAULT CHOICE DEVICE DATA
INQ_DEFAULT_DEFERRAL_STATE_VALUES	INQUIRE DEFAULT DEFERRAL STATE VALUES
INQ_DEFAULT_LOCATOR_DEVICE_DATA	INQUIRE DEFAULT LOCATOR DEVICE DATA
INQ_DEFAULT_PICK_DEVICE_DATA	INQUIRE DEFAULT PICK DEVICE DATA

TABLE 3 - GKS Function Names and Ada Names Ordered Alphabetically by Ada Bound Name

Ada Bound Name	GKS Function
INQ_DEFAULT_STRING_DEVICE_DATA	INQUIRE DEFAULT STRING DEVICE DATA
INQ_DEFAULT_STROKE_DEVICE_DATA	INQUIRE DEFAULT STROKE DEVICE DATA
INQ_DEFAULT_VALUATOR_DEVICE_DATA	INQUIRE DEFAULT VALUATOR DEVICE DATA
INQ_DISPLAY_SPACE_SIZE	INQUIRE DISPLAY SPACE SIZE
INQ_DYNAMIC_MODIFICATION_OF_ SEGMENT_ATTRIBUTES	INQUIRE DYNAMIC MODIFICATION OF SEGMENT ATTRIBUTES
INQ_DYNAMIC_MODIFICATION_OF_ WS_ATTRIBUTES	INQUIRE DYNAMIC MODIFICATION OF WORKSTATION ATTRIBUTES
INQ_FILL_AREA_FACILITIES	INQUIRE FILL AREA FACILITIES
INQ_FILL_AREA_REPRESENTATION	INQUIRE FILL AREA REPRESENTATION
INQ_GDP	INQUIRE GENERALIZED DRAWING PRIMITIVE
INQ_INPUT_QUEUE_OVERFLOW	INQUIRE INPUT QUEUE OVERFLOW
INQ_LEVEL_OF_GKS	INQUIRE LEVEL OF GKS
INQ_LIST_OF_AVAILABLE_GDP	INQUIRE LIST OF AVAILABLE GENERALIZED DRAWING PRIMITIVES
INQ_LIST_OF_AVAILABLE_WS_TYPES	INQUIRE LIST OF AVAILABLE WORKSTATION TYPES
INQ_LIST_OF_COLOUR_INDICES	INQUIRE LIST OF COLOUR INDICES
INQ_LIST_OF_FILL_AREA_INDICES	INQUIRE LIST OF FILL AREA INDICES
INQ_LIST_OF_NORMALIZATION_ TRANSFORMATION_NUMBERS	INQUIRE LIST OF NORMALIZATION TRANSFORMATION NUMBERS
INQ_LIST_OF_PATTERN_INDICES	INQUIRE LIST OF PATTERN INDICES
INQ_LIST_OF_POLYLINE_INDICES	INQUIRE LIST OF POLYLINE INDICES
INQ_LIST_OF_POLYMARKER_INDICES	INQUIRE LIST OF POLYMARKER INDICES
INQ_LIST_OF_TEXT_INDICES	INQUIRE LIST OF TEXT INDICES
INQ_LOCATOR_DEVICE_STATE	INQUIRE LOCATOR DEVICE STATE
INQ_MAX_LENGTH_OF_WS_STATE_TABLES	INQUIRE MAXIMUM LENGTH OF WORKSTATION STATE TABLES
INQ_MAX_NORMALIZATION_ TRANSFORMATION_NUMBER	INQUIRE MAXIMUM NORMALIZATION TRANSFORMATION NUMBER
INQ_MORE_SIMULTANEOUS_EVENTS	INQUIRE MORE SIMULTANEOUS EVENTS
INQ_NAME_OF_OPEN_SEGMENT	INQUIRE NAME OF OPEN SEGMENT
INQ_NORMALIZATION_TRANSFORMATION	INQUIRE NORMALIZATION TRANSFORMATION
INQ_NUMBER_OF_AVAILABLE_LOGICAL_ INPUT_DEVICES	INQUIRE NUMBER OF AVAILABLE LOGICAL INPUT DEVICES
INQ_NUMBER_OF_SEGMENT_ PRIORITIES_SUPPORTED	INQUIRE NUMBER OF SEGMENT PRIORITIES SUPPORTED
INQ_OPERATING_STATE_VALUE	INQUIRE OPERATING STATE VALUE
INQ_PATTERN_FACILITIES	INQUIRE PATTERN FACILITIES
INQ_PATTERN_REPRESENTATION	INQUIRE PATTERN REPRESENTATION
INQ_PICK_DEVICE_STATE	INQUIRE PICK DEVICE STATE
INQ_PIXEL	INQUIRE PIXEL
INQ_PIXEL_ARRAY	INQUIRE PIXEL ARRAY
INQ_PIXEL_ARRAY_DIMENSION	INQUIRE PIXEL ARRAY DIMENSIONS
INQ_POLYLINE_FACILITIES	INQUIRE POLYLINE FACILITIES
INQ_POLYLINE_REPRESENTATION	INQUIRE POLYLINE REPRESENTATION
INQ_POLYMARKER_FACILITIES	INQUIRE POLYMARKER FACILITIES

TABLE 3 - GKS Function Names and Ada Names Ordered Alphabetically by Ada Bound Name

Ada Bound Name	GKS Function
INQ_POLYMARKER_REPRESENTATION	INQUIRE POLYMARKER REPRESENTATION
INQ_PREDEFINED_COLOUR_REPRESENTATION	INQUIRE PREDEFINED COLOUR REPRESENTATION
INQ_PREDEFINED_FILL_AREA_REPRESENTATION	INQUIRE PREDEFINED FILL AREA REPRESENTATION
INQ_PREDEFINED_PATTERN_REPRESENTATION	INQUIRE PREDEFINED PATTERN REPRESENTATION
INQ_PREDEFINED_POLYLINE_REPRESENTATION	INQUIRE PREDEFINED POLYLINE REPRESENTATION
INQ_PREDEFINED_POLYMARKER_REPRESENTATION	INQUIRE PREDEFINED POLYMARKER REPRESENTATION
INQ_PREDEFINED_TEXT_REPRESENTATION	INQUIRE PREDEFINED TEXT REPRESENTATION
INQ_SEGMENT_ATTRIBUTES	INQUIRE SEGMENT ATTRIBUTES
INQ_SET_OF_ACTIVE_WS	INQUIRE SET OF ACTIVE WORKSTATIONS
INQ_SET_OF_ASSOCIATED_WS	INQUIRE SET OF ASSOCIATED WORKSTATIONS
INQ_SET_OF_OPEN_WS	INQUIRE SET OF OPEN WORKSTATIONS
INQ_SET_OF_SEGMENT_NAMES_IN_USE	INQUIRE SET OF SEGMENT NAMES IN USE
INQ_SET_OF_SEGMENT_NAMES_ON_WS	INQUIRE SET OF SEGMENT NAMES ON WORKSTATION
INQ_STRING_DEVICE_STATE	INQUIRE STRING DEVICE STATE
INQ_STROKE_DEVICE_STATE	INQUIRE STROKE DEVICE STATE
INQ_TEXT_EXTENT	INQUIRE TEXT EXTENT
INQ_TEXT_FACILITIES	INQUIRE TEXT FACILITIES
INQ_TEXT_REPRESENTATION	INQUIRE TEXT REPRESENTATION
INQ_VALUATOR_DEVICE_STATE	INQUIRE VALUATOR DEVICE STATE
INQ_WS_CATEGORY	INQUIRE WORKSTATION CATEGORY
INQ_WS_CLASSIFICATION	INQUIRE WORKSTATION CLASSIFICATION
INQ_WS_CONNECTION_AND_TYPE	INQUIRE WORKSTATION CONNECTION AND TYPE
INQ_WS_DEFERRAL_AND_UPDATE_STATES	INQUIRE WORKSTATION DEFERRAL AND UPDATE STATES
INQ_WS_MAX_NUMBERS	INQUIRE WORKSTATION MAXIMUM NUMBER
INQ_WS_STATE	INQUIRE WORKSTATION STATE
INQ_WS_TRANSFORMATION	INQUIRE WORKSTATION TRANSFORMATION
INSERT_SEGMENT	INSERT SEGMENT
INTERPRET_ITEM	INTERPRET ITEM
MESSAGE	MESSAGE
OPEN_GKS	OPEN GKS
OPEN_WS	OPEN WORKSTATION
POLYLINE	POLYLINE
POLYMARKER	POLYMARKER
READ_ITEM_FROM_GKSM	READ ITEM FROM GKSM
REDRAW_ALL_SEGMENTS_ON_WS	REDRAW ALL SEGMENTS ON WORKSTATION
RENAME_SEGMENT	RENAME SEGMENT

TABLE 3 - GKS Function Names and Ada Names Ordered Alphabetically by Ada Bound Name

Ada_Bound_Name	GKS_Function
REQUEST_CHOICE	REQUEST CHOICE
REQUEST_LOCATOR	REQUEST LOCATOR
REQUEST_PICK	REQUEST PICK
REQUEST_STRING	REQUEST STRING
REQUEST_STROKE	REQUEST STROKE
REQUEST_VALUATOR	REQUEST VALUATOR
SAMPLE_CHOICE	SAMPLE CHOICE
SAMPLE_LOCATOR	SAMPLE LOCATOR
SAMPLE_PICK	SAMPLE PICK
SAMPLE_STRING	SAMPLE STRING
SAMPLE_STROKE	SAMPLE STROKE
SAMPLE_VALUATOR	SAMPLE VALUATOR
SELECT_NORMALIZATION_TRANSFORMATION	SELECT NORMALIZATION TRANSFORMATION
SET_ASF	SET ASPECT SOURCE FLAGS
SET_CHAR_EXPANSION_FACTOR	SET CHARACTER EXPANSION FACTOR
SET_CHAR_HEIGHT	SET CHARACTER HEIGHT
SET_CHAR_SPACING	SET CHARACTER SPACING
SET_CHAR_UP_VECTOR	SET CHARACTER UP VECTOR
SET_CHOICE_MODE	SET CHOICE MODE
SET_CLIPPING_INDICATOR	SET CLIPPING INDICATOR
SET_COLOUR_REPRESENTATION	SET COLOUR REPRESENTATION
SET_DEFERRAL_STATE	SET DEFERRAL STATE
SET_DETECTABILITY	SET DETECTABILITY
SET_FILL_AREA_COLOUR_INDEX	SET FILL AREA COLOUR INDEX
SET_FILL_AREA_INDEX	SET FILL AREA INDEX
SET_FILL_AREA_INTERIOR_STYLE	SET FILL AREA INTERIOR STYLE
SET_FILL_AREA_REPRESENTATION	SET FILL AREA REPRESENTATION
SET_FILL_AREA_STYLE_INDEX	SET FILL AREA STYLE INDEX
SET_HIGHLIGHTING	SET HIGHLIGHTING
SET_LINETYPE	SET LINETYPE
SET_LINEWIDTH_SCALE_FACTOR	SET LINEWIDTH SCALE FACTOR
SET_LOCATOR_MODE	SET LOCATOR MODE
SET_MARKER_SIZE_SCALE_FACTOR	SET MARKER SIZE SCALE FACTOR
SET_MARKER_TYPE	SET MARKER TYPE
SET_PATTERN_REFERENCE_POINT	SET PATTERN REFERENCE POINT
SET_PATTERN_REPRESENTATION	SET PATTERN REPRESENTATION
SET_PATTERN_SIZE	SET PATTERN SIZE
SET_PICK_ID	SET PICK IDENTIFIER
SET_PICK_MODE	SET PICK MODE
SET_POLYLINE_COLOUR_INDEX	SET POLYLINE COLOUR INDEX
SET_POLYLINE_INDEX	SET POLYLINE INDEX
SET_POLYLINE_REPRESENTATION	SET POLYLINE REPRESENTATION
SET_POLYMARKER_COLOUR_INDEX	SET POLYMARKER COLOUR INDEX
SET_POLYMARKER_INDEX	SET POLYMARKER INDEX
SET_POLYMARKER_REPRESENTATION	SET POLYMARKER REPRESENTATION
SET_SEGMENT_PRIORITY	SET SEGMENT PRIORITY
SET_SEGMENT_TRANSFORMATION	SET SEGMENT TRANSFORMATION

TABLE 3 - GKS Function Names and Ada Names Ordered Alphabetically by Ada Bound Name

Ada Bound Name	GKS Function
SET_STRING_MODE	SET STRING MODE
SET_STROKE_MODE	SET STROKE MODE
SET_TEXT_ALIGNMENT	SET TEXT ALIGNMENT
SET_TEXT_COLOUR_INDEX	SET TEXT COLOUR INDEX
SET_TEXT_FONT_AND_PRECISION	SET TEXT FONT AND PRECISION
SET_TEXT_INDEX	SET TEXT INDEX
SET_TEXT_PATH	SET TEXT PATH
SET_TEXT_REPRESENTATION	SET TEXT REPRESENTATION
SET_VALUATOR_MODE	SET VALUATOR MODE
SET_VIEWPORT	SET VIEWPORT
SET_VIEWPORT_INPUT_PRIORITY	SET VIEWPORT INPUT PRIORITY
SET_VISIBILITY	SET VISIBILITY
SET_WINDOW	SET WINDOW
SET_WS_VIEWPORT	SET WORKSTATION VIEWPORT
SET_WS_WINDOW	SET WORKSTATION WINDOW
TEXT	TEXT
UPDATE_WS	UPDATE WORKSTATION
WRITE_ITEM_TO_GKSM	WRITE ITEM TO GKSM

Alphabetical GKS functions

The functions are in the same order when listed alphabetically according to the GKS function names as they are when listed alphabetically according to the bound names. Therefore, Table 3, provided above, alphabetically lists the GKS functions.

Table 4 - Alphabetical list of GKS functions by level

LEVEL 0A
- ACTIVATE_WS
- CELL_ARRAY
- CLEAR_WS
- CLOSE_GKS
- CLOSE_WS
- DEACTIVATE_WS
- EMERGENCY_CLOSE_GKS
- ERROR_HANDLING
- ERROR_LOGGING
- ESCAPE
- FILL_AREA
- GDP
- GET_ITEM_TYPE_FROM_GKSM
- INQ_CLIPPING
- INQ_COLOUR_FACILITIES
- INQ_COLOUR_REPRESENTATION

Table 4 - Alphabetical list of GKS functions by level

INQ_CURRENT_INDIVIDUAL_ATTRIBUTE_VALUES

The following functions are a one-to-many mapping of the GKS function "inquire current individual attribute values":

- INQ_CHAR_EXPANSION_FACTOR
- INQ_CHAR_SPACING
- INQ_FILL_AREA_COLOUR_INDEX
- INQ_FILL_AREA_INTERIOR_STYLE
- INQ_FILL_AREA_STYLE_INDEX
- INQ_LINETYPE
- INQ_LINEWIDTH_SCALE_FACTOR
- INQ_LIST_OF_ASF
- INQ_POLYLINE_COLOUR_INDEX
- INQ_POLYMARKER_COLOUR_INDEX
- INQ_POLYMARKER_SIZE_SCALE_FACTOR
- INQ_POLYMARKER_TYPE
- INQ_TEXT_COLOUR_INDEX
- INQ_TEXT_FONT_AND_PRECISION

INQ_CURRENT_NORMALIZATION_TRANSFORMATION_NUMBER

INQ_CURRENT_PRIMITIVE_ATTRIBUTE_VALUES

The following functions are a one-to-many mapping of the GKS function "inquire current primitive attribute values":

- INQ_POLYLINE_INDEX
- INQ_POLYMARKER_INDEX
- INQ_TEXT_INDEX
- INQ_CHAR_HEIGHT
- INQ_CHAR_UP_VECTOR
- INQ_CHAR_WIDTH
- INQ_CHAR_BASE_VECTOR
- INQ_TEXT_PATH
- INQ_TEXT_ALIGNMENT
- INQ_FILL_AREA_INDEX
- INQ_PATTERN_HEIGHT_VECTOR
- INQ_PATTERN_WIDTH_VECTOR
- INQ_PATTERN_REFERENCE_POINT

INQ_DISPLAY_SPACE_SIZE
INQ_FILL_AREA_FACILITIES
INQ_GDP
INQ_LEVEL_OF_GKS
INQ_LIST_OF_AVAILABLE_GDP
INQ_LIST_OF_AVAILABLE_WS_TYPES
INQ_LIST_OF_COLOUR_INDICES
INQ_LIST_OF_NORMALIZATION_TRANSFORMATION_NUMBER
INQ_MAX_LENGTH_WS_STATE_TABLES
INQ_MAX_NORMALIZATION_TRANSFORMATION_NUMBER
INQ_NORMALIZATION_TRANSFORMATION
INQ_OPERATING_STATE_VALUE

Table 4 - Alphabetical list of GKS functions by level

INQ_PATTERN_FACILITIES
INQ_PIXEL
INQ_PIXEL_ARRAY
INQ_PIXEL_ARRAY_DIMENSIONS
INQ_POLYLINE_FACILITIES
INQ_POLYMARKER_FACILITIES
INQ_PREDEFINED_COLOUR_REPRESENTATION
INQ_PREDEFINED_FILL_AREA_REPRESENTATION
INQ_PREDEFINED_PATTERN_REPRESENTATION
INQ_PREDEFINED_POLYLINE_REPRESENTATION
INQ_PREDEFINED_POLYMARKER_REPRESENTATION
INQ_PREDEFINED_TEXT_REPRESENTATION
INQ_SET_OF_OPEN_WS
INQ_TEXT_EXTENT
INQ_TEXT_FACILITIES
INQ_WS_CATEGORIES
INQ_WS_CLASSIFICATION
INQ_WS_CONNECTION_AND_TYPE
INQ_WS_DEFERRAL_AND_UPDATE_STATES
INQ_WS_STATE
INQ_WS_TRANSFORMATION
INTERPRET_ITEM
OPEN_GKS
OPEN_WS
POLYLINE
POLYMARKER
READ_ITEM_FROM_GKSM
SELECT_NORMALIZATION _TRANSFORMATION
SET_ASF
SET_CHAR_EXPANSION_FACTOR
SET_CHAR_HEIGHT
SET_CHAR_SPACING
SET_CHAR_UP_VECTOR
SET_CLIPPING_INDICATOR
SET_COLOUR_REPRESENTATION
SET_FILL_AREA_COLOUR_INDEX
SET_FILL_AREA_INDEX
SET_FILL_AREA_INTERIOR_STYLE
SET_FILL_AREA_STYLE_INDEX
SET_LINETYPE
SET_LINEWIDTH_SCALE_FACTOR
SET_MARKER_SIZE_SCALE_FACTOR
SET_MARKER_TYPE
SET_PATTERN_REFERENCE_POINT
SET_PATTERN_SIZE
SET_POLYLINE_COLOUR_INDEX
SET_POLYMARKER_COLOUR_INDEX
SET_POLYLINE_INDEX
SET_POLYMARKER_INDEX

Table 4 - Alphabetical list of GKS functions by level

SET_TEXT_ALIGNMENT
SET_TEXT_COLOUR_INDEX
SET_TEXT_FONT_AND_PRECISION
SET_TEXT_INDEX
SET_TEXT_PATH
SET_VIEWPORT
SET_WINDOW
SET_WS_VIEWPORT
SET_WS_WINDOW

TEXT

UPDATE_WS

WRITE_ITEM_TO_GKSM

LEVEL 0b

INITIALISE_CHOICE
INITIALISE_LOCATOR
INITIALISE_STRING
INITIALISE_STROKE
INITIALISE_VALUATOR
INQ_CHOICE_DEVICE_STATE
INQ_DEFAULT_CHOICE_DEVICE_STATE
INQ_DEFAULT_LOCATOR_DEVICE_DATA
INQ_DEFAULT_STRING_DEVICE_DATA
INQ_DEFAULT_STROKE_DEVICE_DATA
INQ_DEFAULT_VALUATOR_DEVICE_DATA
INQ_LOCATOR_DEVICE_STATE
INQ_NUMBER_OF_AVAILABLE_LOGICAL_INPUT_DEVICES
INQ_STRING_DEVICE_STATE
INQ_STROKE_DEVICE_STATE
INQ_VALUATOR_DEVICE_STATE
REQUEST_CHOICE
REQUEST_LOCATOR
REQUEST_STRING
REQUEST_STROKE
REQUEST_VALUATOR
SET_CHOICE_MODE
SET_LOCATOR_MODE
SET_STRING_MODE
SET_STROKE_MODE
SET_VALUATOR_MODE
SET_VIEWPORT_INPUT_PRIORITY

LEVEL 0c

AWAIT_EVENT
FLUSH_DEVICE_EVENTS
GET_CHOICE
GET_LOCATOR

Table 4 - Alphabetical list of GKS functions by level

GET_STRING
GET_STROKE
GET_VALUATOR
INQ_INPUT_QUEUE_OVERFLOW
INQ_MORE_SIMULTANEOUS_EVENTS
SAMPLE_CHOICE
SAMPLE_LOCATOR
SAMPLE_STRING
SAMPLE_STROKE
SAMPLE_VALUATOR

LEVEL 1a

ACCUMULATE_TRANSFORMATION_MATRIX
CLOSE_SEGMENT
CREATE_SEGMENT
DELETE_SEGMENT
DELETE_SEGMENT_FROM_WS
EVALUATE_TRANSFORMATION_MATRIX
INQ_DEFAULT_DEFERRAL_STATE_VALUES
INQ_DYNAMIC_MODIFICATION_OF_SEGMENT_ATTRIBUTES
INQ_DYNAMIC_MODIFICATION_OF_WS_ATTRIBUTES
INQ_FILL_AREA_REPRESENTATION
INQ_LIST_OF_FILL_AREA_INDICES
INQ_LIST_OF_PATTERN_INDICES
INQ_LIST_OF_POLYLINE_INDICES
INQ_LIST_OF_POLYMARKER_INDICES
INQ_LIST_OF_TEXT_INDICES
INQ_NAME_OF_OPEN_SEGMENT
INQ_NUMBER_OF_SEGMENT_PRIORITIES_SUPPORTED
INQ_PATTERN_REPRESENTATION
INQ_POLYLINE_REPRESENTATION
INQ_POLYMARKER_REPRESENTATION
INQ_SEGMENT_ATTRIBUTES
INQ_SET_OF_ACTIVE_WS
INQ_SET_OF_ASSOCIATED_WS
INQ_SET_OF_SEGMENT_NAMES_IN_USE
INQ_SET_OF_SEGMENT_NAMES_ON_WS
INQ_TEXT_REPRESENTATION
INQ_WS_MAX_NUMBERS
MESSAGE
REDRAW_ALL_SEGMENTS_ON_WS
RENAME_SEGMENT
SET_DEFERRAL_STATE
SET_FILL_AREA_REPRESENTATION
SET_HIGHLIGHTING
SET_PATTERN_REPRESENTATION
SET_POLYLINE_REPRESENTATION
SET_POLYMARKER_REPRESENTATION
SET_SEGMENT_PRIORITY

Table 4 - Alphabetical list of GKS functions by level

SET_SEGMENT_TRANSFORMATION
SET_TEXT_REPRESENTATION
SET_VISIBILITY

LEVEL 1b
INITIALISE_PICK
INQ_CURRENT_PICK_ID_VALUE
INQ_DEFAULT_PICK_DEVICE_DATA
INQ_PICK_DEVICE_STATE
REQUEST_PICK
SET_DETECTABLILITY
SET_PICK_ID
SET_PICK_MODE

LEVEL 1c
GET_PICK
SAMPLE_PICK

LEVEL 2a
ASSOCIATE_SEGMENT_WITH_WS
COPY_SEGMENT_TO_WS
INSERT_SEGMENT

LEVEL 2b
NONE

LEVEL 2c
NONE

4.2 Data Type Definitions

4.2.1 Abbreviations used in the data type definitions

ASF	aspect source flag
CHAR	character
DC	device coordinate
GDP	generalized drawing primitive
GKS	graphical kernel system
GKSM	graphical kernel system metafile
ID	identifier
MAX	maximum
NDC	normalized device coordinates
WC	world coordinate
WS	workstation

4.2.2 Alphabetical list of type definitions

This section contains an alphabetical listing of all of the data type definitions used to define the Ada binding to GKS. Each of these declarations specifies the level of GKS at which the data type declaration must be available in an implementation of GKS of that level or any level "above" the level in which the type declaration is first needed (same as for functions). Each element declared also includes a comment about the type and/or use of the type. Some of the declarations employ constant values in the definition of the type. All of these constant declarations are included in the GKS_TYPES package.

ASF Level 0a

```
type ASF is (BUNDLED, INDIVIDUAL);

   -- This type defines an aspect source flag whose value indicates whether
   -- an aspect of a primitive should be set from a bundle table or from an
   -- individual attribute.
```

ASF_LIST Level 0a

```
type ASF_LIST is
      record
         TYPE_OF_LINE_ASF          : ASF;
         WIDTH_ASF                 : ASF;
         LINE_COLOUR_ASF           : ASF;
         TYPE_OF_MARKER_ASF        : ASF;
         SIZE_ASF                  : ASF;
         MARKER_COLOUR_ASF         : ASF;
         FONT_PRECISION_ASF        : ASF;
         EXPANSION_ASF             : ASF;
         SPACING_ASF               : ASF;
         TEXT_COLOUR_ASF           : ASF;
         INTERIOR_ASF              : ASF;
         STYLE_ASF                 : ASF;
         FILL_AREA_COLOUR_ASF      : ASF;
      end record;
   -- A record containing all of the aspect source flags, with components indicating the
   -- specific flag.
```

ATTRIBUTES_USED Level 0a

```
package ATTRIBUTES_USED is
        new GKS_LIST_UTILITIES (ATTRIBUTES_USED_TYPE);

        -- Provides for a list of the attributes used.
```

ATTRIBUTES_USED_TYPE Level 0a

```
        type ATTRIBUTES_USED_TYPE is   (POLYLINE_ATTRIBUTES,
                                        POLYMARKER_ATTRIBUTES,
                                        TEXT_ATTRIBUTES,
                                        FILL_AREA_ATTRIBUTES);

        -- The types of attributes which may be used in generating output for a GDP and in
        -- generating prompt and echo information for certain prompt and echo types of
        -- certain classes of input devices.
```

CHAR_EXPANSION Level 0a

```
        type CHAR_EXPANSION is new SCALE_FACTOR range
             SCALE_FACTOR'SAFE_SMALL..SCALE_FACTOR'LAST;

        -- Defines a character expansion factor.  Factors are unitless, and must be greater than zero.
```

CHAR_SPACING Level 0a

```
        type CHAR_SPACING is new SCALE_FACTOR;

        -- Defines a character spacing factor.  The factors are unitless.  A positive value indicates
        -- the amount of extra space between characters in a text string, and a negative value
        -- indicates the amount of overlap between character boxes in a text string.
```

CHOICE_DEVICE_NUMBER Level 0b

```
        type CHOICE_DEVICE_NUMBER is new DEVICE_NUMBER;

        -- Provides for choice device identifiers.
```

CHOICE_PROMPT Level 0b

```
type CHOICE_PROMPT is (OFF, ON);

-- Indicates for a choice prompt and echo type whether a specified prompt is to
-- be displayed or not.
```

CHOICE_PROMPTS Level 0b

```
package CHOICE_PROMPTS is
        new GKS_LIST_UTILITIES (CHOICE_PROMPT);

 -- Provides for lists of prompts.
```

CHOICE_PROMPT_ECHO_TYPE Level 0b

```
type CHOICE_PROMPT_ECHO_TYPE is new INTEGER;

-- Defines the choice prompt and echo type.
```

CHOICE_PROMPT_ECHO_TYPES Level 0b

```
package CHOICE_PROMPT_ECHO_TYPES is
        new GKS_LIST_UTILITIES (CHOICE_PROMPT_ECHO_TYPE);

-- Provides for lists of choice prompt and echo types.
```

CHOICE_PROMPT_STRING Level 0b

```
type CHOICE_PROMPT_STRING (LENGTH : STRING_SMALL_NATURAL := 0) is
        record
           CONTENTS : STRING (1..LENGTH);
        end record;

-- Provides for a variable length prompt.  Objects of type should be declared
-- unconstrained to allow for dynamic modification of the length.
```

CHOICE_PROMPT_STRING_ARRAY Level 0b

```
type CHOICE_PROMPT_STRING_ARRAY is array (POSITIVE range <>)
        of CHOICE_PROMPT_STRING;

-- Provides for an array of prompt strings.
```

CHOICE_PROMPT_STRING_LIST Level 0b

```
type CHOICE_PROMPT_STRING_LIST(LENGTH:CHOICE_SMALL_NATURAL:= 0)
        is record
            LIST : CHOICE_PROMPT_STRING_ARRAY (1..LENGTH);
        end record;

-- Provides for lists of prompt strings.
```

CHOICE_REQUEST_STATUS Level 0b

```
type CHOICE_REQUEST_STATUS is (OK, NOCHOICE, NONE);

-- Defines the status of a choice input operation for the request function.
```

CHOICE_SMALL_NATURAL Level 0b

```
subtype CHOICE_SMALL_NATURAL
        is NATURAL RANGE 0..CHOICE_SMALL_NATURAL_MAX;

-- This is a subtype declaration which allows for unconstrained record
-- objects for CHOICE_PROMPT_STRING_LIST  type without causing the
-- exception STORAGE_ERROR to be raised.
```

CHOICE_STATUS Level 0b

```
subtype CHOICE_STATUS is CHOICE_REQUEST_STATUS range OK..NOCHOICE;

-- Indicates if a choice was made by the operator for the sample, get, and inquiry functions.
```

CHOICE_VALUE Level 0b

```
type CHOICE_VALUE is new POSITIVE;

-- Defines the choice values available on an implementation.
```

CLIPPING_INDICATOR Level 0a

```
type CLIPPING_INDICATOR is (CLIP, NOCLIP);

-- Indicates whether or not clipping is to be performed.
```

COLOUR_AVAILABLE Level 0a

```
type COLOUR_AVAILABLE is (COLOUR, MONOCHROME);

-- Indicates whether colour output is available on a workstation.
```

COLOUR_INDEX Level 0a

```
subtype COLOUR_INDEX is PIXEL_COLOUR_INDEX
        range 0..PIXEL_COLOUR_INDEX'LAST;
-- Indices into colour tables are of this type.
```

COLOUR_INDICES Level 0a

```
package COLOUR_INDICES is new GKS_LIST_UTILITIES (COLOUR_INDEX);

-- Provides for a set of colour indices which are available on a particular workstation.
```

COLOUR_MATRIX Level 0a

```
type COLOUR_MATRIX is array (POSITIVE range <>,POSITIVE range <>)
                      of COLOUR_INDEX;

-- Provides for matrices containing colour indices corresponding to a
-- cell array or pattern array.
```

COLOUR_REPRESENTATION Level 0a

```
type COLOUR_REPRESENTATION is
    record
      RED    : INTENSITY;
      GREEN  : INTENSITY;
      BLUE   : INTENSITY;
    end record;

-- Defines the representation of a colour as a combination of intensities in
-- an RGB colour system.
```

CONTROL_FLAG Level 0a

```
type CONTROL_FLAG is (CONDITIONALLY, ALWAYS);

-- The control flag is used to indicate the conditions under which the display
-- surface should be cleared.
```

DC Level 0a

```
package DC is new GKS_COORDINATE_SYSTEM (DC_TYPE);

-- Defines the Device Coordinate System.
```

DC_TYPE Level 0a

```
type DC_TYPE is digits PRECISION;

-- The type of a coordinate in the Device Coordinate System.
```

DC_UNITS Level 0a

```
type DC_UNITS is (METRES, OTHER);

-- Device coordinate units for a particular workstation should be in metres unless
-- the device is incapable of producing a precisely scaled image, and appropriate work-
-- station dependent units otherwise.
```

DEFERRAL_MODE Level 0a

```
type DEFERRAL_MODE is (ASAP, BNIG, BNIL, ASTI);

-- Defines the four GKS deferral modes.
```

DEVICE_NUMBER Level 0b

```
package DEVICE_NUMBER_TYPE is
        DEVICE_NUMBER_TYPE is new POSITIVE;
end DEVICE_NUMBER_TYPE;

-- Logical input devices are referenced as device numbers.
```

DISPLAY_CLASS Level 0a

```
type DISPLAY_CLASS is (VECTOR_DISPLAY,
                       RASTER_DISPLAY,
                       OTHER_DISPLAY);

-- The classification of a workstation of category  OUTPUT or OUTIN.
```

DISPLAY_SURFACE_EMPTY Level 0a

```
type DISPLAY_SURFACE_EMPTY is (EMPTY, NOTEMPTY);

-- Indicates whether the display surface is empty.
```

DYNAMIC_MODIFICATION Level 1a

```
type DYNAMIC_MODIFICATION is (IRG, IMM);

-- Indicates whether an update to the state list is performed immediately (IMM)
-- or requires implicit regeneration (IRG).
```

ECHO_SWITCH Level 0b

```
type ECHO_SWITCH is (ECHO, NOECHO);

-- Indicates whether or not echoing of the prompt is performed.
```

ERROR_NUMBER Level 0a

```
type ERROR_NUMBER is new INTEGER;

-- Defines the type for error indicator values.
```

EVENT_DEVICE_NUMBER Level 0c

```
type EVENT_DEVICE_NUMBER (CLASS : INPUT_CLASS := NONE) is
    record
     case CLASS is
       when  NONE                => null;
       when LOCATOR_INPUT        => LOCATOR_EVENT_DEVICE
                                    : LOCATOR_ DEVICE_NUMBER;
       when STROKE_INPUT         => STROKE_EVENT_DEVICE
                                    : STROKE_DEVICE_NUMBER;
       when VALUATOR_INPUT       => VALUATOR_EVENT_DEVICE
                                    : VALUATOR_DEVICE_NUMBER;
       when CHOICE_INPUT         => CHOICE_EVENT_DEVICE
                                    : CHOICE_DEVICE_NUMBER;
       when PICK_INPUT           => PICK_EVENT_DEVICE
                                    : PICK_DEVICE_NUMBER;
       when STRING_INPUT         => STRING_EVENT_DEVICE
                                    : STRING_DEVICE_NUMBER;
     end case;
    end record;
-- Provides for returning any class of device number from the event queue.
```

EVENT_OVERFLOW_DEVICE_NUMBER Level 0c

```
type EVENT_OVERFLOW_DEVICE_NUMBER
        (CLASS : INPUT_QUEUE_CLASS := LOCATOR_INPUT) is
    record
     case CLASS is
       when LOCATOR_INPUT      => LOCATOR_EVENT_DEVICE
                                    : LOCATOR_ DEVICE_NUMBER;
       when STROKE_INPUT       => STROKE_EVENT_DEVICE
                                    : STROKE_DEVICE_NUMBER;
       when VALUATOR_INPUT     => VALUATOR_EVENT_DEVICE
                                    : VALUATOR_DEVICE_NUMBER;
       when CHOICE_INPUT       => CHOICE_EVENT_DEVICE
                                    : CHOICE_DEVICE_NUMBER;
       when PICK_INPUT         => PICK_EVENT_DEVICE
                                    : PICK_DEVICE_NUMBER;
       when STRING_INPUT       => STRING_EVENT_DEVICE
                                    : STRING_DEVICE_NUMBER;
     end case;
    end record;
```

FILL_AREA_INDEX Level 0a

```
type FILL_AREA_INDEX is new POSITIVE;

-- Defines fill area bundle table indices.
```

FILL_AREA_INDICES Level 0a

```
package FILL_AREA_INDICES is
            new GKS_LIST_UTILITIES (FILL_AREA_INDEX);

-- Provides for lists of fill area bundle table indices.
```

GDP_ID Level 0a

```
type GDP_ID is new INTEGER;

-- Selects among the kinds of Generalized Drawing Primitives.
```

GDP_IDS Level 0a

```
package GDP_IDS is new GKS_LIST_UTILITIES (GDP_ID);

-- Provides for lists of Generalized Drawing Primitive IDs.
```

GKS_Level Level 0a

```
type GKS_Level is (L0a, L0b, L0c, L1a, L1b, L1c, L2a, L2b, L2c);

-- The valid Levels of GKS.
```

GKSM_ITEM_TYPE Level 0a

```
type GKSM_ITEM_TYPE is new NATURAL;

-- The type of an item contained in a GKSM metafile.
```

HATCH_STYLE Level 0a

```
subtype HATCH_STYLE is STYLE_INDEX;
-- Defines the fill area hatch styles type.
```

HATCH_STYLES Level 0a

```
package HATCH_STYLES is new GKS_LIST_UTILITIES (HATCH_STYLE);

-- Provides for lists of hatch styles.
```

HORIZONTAL_ALIGNMENT Level 0a

```
type HORIZONTAL_ALIGNMENT is (NORMAL, LEFT, CENTRE, RIGHT);

-- The alignment of the text extent parallelogram with respect to the horizontal
-- positioning of the text.
```

IMPLEMENTATION_DEFINED_ERROR Level 0a

```
subtype IMPLEMENTATION_DEFINED_ERROR is ERROR_NUMBER
        range ERROR_NUMBER'FIRST .. -1;

-- Defines the range of ERROR_NUMBERS to indicate that an implementation
-- defined error has occurred.
```

INDIVIDUAL_ATTRIBUTE_VALUES Level 0a

```
type INDIVIDUAL_ATTRIBUTE_VALUES is
    record
        TYPE_OF_LINE            : LINETYPE;
        WIDTH                   : LINEWIDTH;
        LINE_COLOUR             : COLOUR_INDEX;
        TYPE_OF_MARKER          : MARKER_TYPE;
        SIZE                    : MARKER_SIZE;
        MARKER_COLOUR           : COLOUR_INDEX;
        FONT_PRECISION          : TEXT_FONT_PRECISION;
        EXPANSION               : CHAR_EXPANSION;
        SPACING                 : CHAR_SPACING;
        TEXT_COLOUR             : COLOUR_INDEX;
        INTERIOR                : INTERIOR_STYLE;
        STYLE                   : STYLE_INDEX;
        FILL_AREA_COLOUR        : COLOUR_INDEX;
        ASF                     : ASF_LIST;
    end record;

-- A record containing all of the current individual attributes for the procedure
-- INQ_CURRENT_INDIVIDUAL_ATTRIBUTE_VALUES.
```

INPUT_CLASS Level 0b

```
type INPUT_CLASS is   (NONE,
                       LOCATOR_INPUT,
                       STROKE_INPUT,
                       VALUATOR_INPUT,
                       CHOICE_INPUT,
                       PICK_INPUT,
                       STRING_INPUT);

-- Defines the input device classifications for workstations of category INPUT or OUTIN.
```

INPUT_QUEUE_CLASS Level 0c

```
subtype INPUT_QUEUE_CLASS is INPUT_CLASS range
        LOCATOR_INPUT .. STRING_INPUT;

-- Defines the input device classifications for situations in which the
-- NONE classification is impossible.
```

INPUT_STATUS Level 0b

```
type INPUT_STATUS is (OK, NONE);

-- Defines the status of a locator, stroke, valuator, or string operation.
```

INPUT_STRING Level 0b

```
type INPUT_STRING (LENGTH : STRING_SMALL_NATURAL := 0) is
    record
        CONTENTS : STRING (1..LENGTH);
    end record;

-- Provides a variable length string.  Objects of this type should be declared
-- unconstrained to allow for dynamic modification of the length.
```

INTENSITY Level 0a

```
type INTENSITY is digits PRECISION range 0.0..1.0;

-- Defines the range of possible intensities of a colour.
```

INTERIOR_STYLE Level 0a

```
type INTERIOR_STYLE is (HOLLOW, SOLID, PATTERN, HATCH);
-- Defines the fill area interior styles.
```

INTERIOR_STYLES Level 0a

```
package INTERIOR_STYLES is
        new GKS_LIST_UTILITIES (INTERIOR_STYLE);
-- Provides for lists of interior styles.
```

INVALID_VALUES_INDICATOR Level 0a

```
type INVALID_VALUES_INDICATOR is (ABSENT,PRESENT);
-- Indicates whether the value -1 (i.e. "invalid") is absent from or present
-- in the PIXEL_ARRAY parameter returned by INQ_PIXEL_ARRAY.
```

LANGUAGE_BINDING_ERROR Level 0a

```
subtype LANGUAGE_BINDING_ERROR is ERROR_NUMBER range 2500..2999;
-- Defines the range of ERROR_NUMBERS to indicate a language binding error has occurred.
```

LINETYPE Level 0a

```
type LINETYPE is new INTEGER;

-- Defines the types of line styles provided by GKS.
```

LINETYPES Level 0a

```
package LINETYPES is new GKS_LIST_UTILITIES (LINETYPE);

-- Provides for lists of linc types.
```

LINEWIDTH Level 0a

```
type LINEWIDTH is new SCALE_FACTOR range 0.0..SCALE_FACTOR'LAST;

-- The width of a line is indicated by a scale factor.
```

LOCATOR_DEVICE_NUMBER Level 0b

```
type LOCATOR_DEVICE_NUMBER is new DEVICE_NUMBER;

-- Provides for locator device identifiers.
```

LOCATOR_PROMPT_ECHO_TYPE Level 0b

```
type LOCATOR_PROMPT_ECHO_TYPE is new INTEGER;

-- Defines the locator prompt and echo types supported by the implementation.
```

LOCATOR_PROMPT_ECHO_TYPES Level 0b

```
package LOCATOR_PROMPT_ECHO_TYPES is
        new GKS_LIST_UTILITIES (LOCATOR_PROMPT_ECHO_TYPE);

-- Provides for lists of locator prompt and echo types.
```

MARKER_SIZE Level 0a

```
type MARKER_SIZE is new SCALE_FACTOR range 0.0..SCALE_FACTOR'LAST;

-- The size of a marker is indicated by a scale factor.
```

MARKER_TYPE Level 0a

```
type MARKER_TYPE is new INTEGER;

-- Defines the type for markers provided by GKS.
```

MARKER_TYPES Level 0a

```
package MARKER_TYPES is new GKS_LIST_UTILITIES (MARKER_TYPE);
-- Provides for lists of marker types.
```

MORE_EVENTS Level 0c

```
type MORE_EVENTS is (NOMORE, MORE);
-- Indicates whether more events are contained in the input event queue.
```

NDC Level 0a

```
package NDC is new GKS_COORDINATE_SYSTEM (NDC_TYPE);
-- Defines the Normalized Device Coordinate System.
```

NDC_TYPE Level 0a

```
type NDC_TYPE is digits PRECISION;
-- Defines the type of a coordinate in the Normalized Device Coordinate System.
```

NEW_FRAME_NECESSARY Level 0a

```
type NEW_FRAME_NECESSARY is (NO, YES);
-- Indicates whether a new frame action is necessary at update.
```

OPERATING_MODE Level 0b

```
type OPERATING_MODE is (REQUEST_MODE, SAMPLE_MODE, EVENT_MODE);
-- Defines the operating modes of an input device.
```

OPERATING_STATE Level 0a

```
type OPERATING_STATE is (GKCL, GKOP, WSOP, WSAC, SGOP);
-- Defines the five GKS operating states.
```

PATTERN_INDEX Level 0a

```
subtype PATTERN_INDEX is STYLE_INDEX range 1..STYLE_INDEX'LAST;

-- Defines the range of pattern table indices.
```

PATTERN_INDICES Level 0a

```
package PATTERN_INDICES is
        new GKS_LIST_UTILITIES (PATTERN_INDEX);

-- Provides for lists of pattern table indices.
```

PICK_DEVICE_NUMBER Level 1b

```
type PICK_DEVICE_NUMBER is new DEVICE_NUMBER;

-- Provides for pick devices.
```

PICK_ID Level 1b

```
type PICK_ID is new POSITIVE;

-- Defines the range of pick identifiers available on an implementation.
```

PICK_IDS Level 1b

```
package PICK_IDS is new GKS_LIST_UTILITIES (PICK_ID);

-- Provides for lists of pick identifiers.
```

PICK_PROMPT_ECHO_TYPE Level 0b

```
type PICK_PROMPT_ECHO_TYPE is new INTEGER;

-- Defines the pick prompt and echo type.
```

PICK_PROMPT_ECHO_TYPES Level 0b

```
package PICK_PROMPT_ECHO_TYPES is new GKS_LIST_UTILITIES
        (PICK_PROMPT_ECHO_TYPE);

-- Provides for lists of pick prompt and echo types.
```

PICK_REQUEST_STATUS Level 1b

```
type PICK_REQUEST_STATUS is (OK, NOPICK, NONE);

-- Defines the status of a pick input operation  for the request function.
```

PICK_STATUS Level 1b

```
subtype PICK_STATUS is PICK_REQUEST_STATUS range OK..NOPICK;

-- Defines the status of a pick input operation for the sample, get, and inquiry functions.
```

PIXEL_COLOUR_INDEX Level 0a

```
type PIXEL_COLOUR_INDEX is new INTEGER range -1..INTEGER'LAST;

-- A type for the pixel colour where the value -1 represents an invalid colour index.
```

PIXEL_COLOUR_MATRIX Level 0a

```
type PIXEL_COLOUR_MATRIX is array (POSITIVE range <>,
     POSITIVE range <>) of PIXEL_COLOUR_INDEX;

-- Provides for matrices of pixel colours.
```

POLYLINE_INDEX Level 0a

```
type POLYLINE_INDEX is new POSITIVE;

-- Defines the range of polyline indices.
```

POLYLINE_INDICES Level 0a

```
package POLYLINE_INDICES is
     new GKS_LIST_UTILITIES (POLYLINE_INDEX);

-- Provides for lists of polyline indices.
```

POLYMARKER_INDEX Level 0a

```
type POLYMARKER_INDEX is new POSITIVE;

-- Defines the range of polymarker bundle table indices.
```

POLYMARKER_INDICES Level 0a

```
package POLYMARKER_INDICES is
        new GKS_LIST_UTILITIES (POLYMARKER_INDEX);

-- Provides for lists of polymarker indices.
```

POSITIVE_TRANSFORMATION_NUMBER Level 0a

```
subtype POSITIVE_TRANSFORMATION_NUMBER is TRANSFORMATION_NUMBER
    range 1 .. TRANSFORMATION_NUMBER'LAST

-- A normalization transformation number corresponding to a settable transformation
```

PRIMITIVE_ATTRIBUTE_VALUES Level 0a

```
type PRIMITIVE_ATTRIBUTE_VALUES is
        record
            INDEX_POLYLINE                  : POLYLINE_INDEX;
            INDEX_POLYMARKER                : POLYMARKER_INDEX;
            INDEX_TEXT                      : TEXT_INDEX;
            CHAR_HEIGHT                     : WC.MAGNITUDE;
            CHAR_UP                         : WC.VECTOR;
            CHAR_WIDTH                      : WC.MAGNITUDE;
            CHAR_BASE                       : WC.VECTOR;
            PATH                            : TEXT_PATH;
            ALIGNMENT                       : TEXT_ALIGNMENT;
            INDEX_FILL_AREA                 : FILL_AREA_INDEX;
            PATTERN_WIDTH_VECTOR            : WC.VECTOR;
            PATTERN_HEIGHT_VECTOR           : WC.VECTOR;
            PATTERN_REFERENCE_POINT         : WC.POINT;
        end record;

-- A record containing all of the current primitive attributes for the procedure
-- INQ_CURRENT_PRIMITIVE_ATTRIBUTE_VALUES.
```

RADIANS Level 1a

```
type RADIANS is digits PRECISION;

-- Values used in performing segment transformations (rotation angle).  Positive indicates
-- an anticlockwise direction.
```

RANGE_OF_EXPANSIONS Level 0a

```
type RANGE_OF_EXPANSIONS is
      record
         MIN    : CHAR_EXPANSION;
         MAX    : CHAR_EXPANSION;
       end record;

-- Provides a range of character expansion factors.
```

RASTER_UNITS Level 0a

```
type RASTER_UNITS is new POSITIVE;

-- Defines the range of raster units.
```

RASTER_UNIT_SIZE Level 0a

```
type RASTER_UNIT_SIZE is
      record
         X : RASTER_UNITS;
         Y : RASTER_UNITS;
      end record;

-- Defines the size of a display screen in raster units on a raster device.
```

REGENERATION_MODE Level 0a

```
type REGENERATION_MODE is (SUPPRESSED, ALLOWED);

-- Indicates whether implicit regeneration of the display is suppressed or allowed.
```

RELATIVE_PRIORITY Level 0a

```
type RELATIVE_PRIORITY is (HIGHER, LOWER);

-- Indicates the relative priority between two normalization transformations.
```

RETURN_VALUE_TYPE Level 0a

```
type RETURN_VALUE_TYPE is (SET, REALIZED);

-- Indicates whether the returned values should be as they were set by the program or as they were
-- actually realized on the device.
```

SCALE_FACTOR Level 0a

```
package SCALE_FACTOR_TYPE is
        type SCALE_FACTOR is digits PRECISION;
end SCALE_FACTOR_TYPE;

-- The type used for unitless scaling factors.
```

SEGMENT_DETECTABILITY Level 1a

```
type SEGMENT_DETECTABILITY is (UNDETECTABLE, DETECTABLE);

-- Indicates whether a segment is detectable or not.
```

SEGMENT_HIGHLIGHTING Level 1a

```
type SEGMENT_HIGHLIGHTING is (NORMAL, HIGHLIGHTED);

-- Indicates whether a segment is highlighted or not.
```

SEGMENT_NAME Level 1a

```
type SEGMENT_NAME is new POSITIVE;

-- Defines the range of segment names.
```

SEGMENT_NAMES Level 1a

```
package SEGMENT_NAMES is new GKS_LIST_UTILITIES (SEGMENT_NAME);

-- Provides for lists of segment names.
```

SEGMENT_PRIORITY Level 1a

```
type SEGMENT_PRIORITY is digits PRECISION range 0.0..1.0;

-- Defines the priority of a segment.
```

SEGMENT_VISIBILITY Level 1a

```
type SEGMENT_VISIBILITY is (VISIBLE, INVISIBLE);

-- Indicates whether a segment is visible or not.
```

SMALL_NATURAL Level 0a

```
subtype SMALL_NATURAL is NATURAL range 0..SMALL_NATURAL_MAX;

-- This is a subtype declaration which allows for unconstrained record objects for various record types
-- without causing the exception STORAGE_ERROR to be raised.
```

STRING_DEVICE_NUMBER Level 0b

```
type STRING_DEVICE_NUMBER is new DEVICE_NUMBER;

-- Provides for string device number.
```

STRING_PROMPT_ECHO_TYPE Level 0b

```
type STRING_PROMPT_ECHO_TYPE is new INTEGER;

-- Defines the string prompt and echo types.
```

STRING_PROMPT_ECHO_TYPES Level 0b

```
package STRING_PROMPT_ECHO_TYPES is
        new GKS_LIST_UTILITIES (STRING_PROMPT_ECHO_TYPE);

-- Provides for lists of string prompt and echo types.
```

STRING_SMALL_NATURAL Level 0a

```
subtype STRING_SMALL_NATURAL is NATURAL
        range 0..STRING_SMALL_NATURAL_MAX;

-- This is a subtype declaration which allows for unconstrained record objects for various string record
-- types without causing the exception STORAGE_ERROR to be raised.
```

STROKE_DEVICE_NUMBER Level 0b

```
type STROKE_DEVICE_NUMBER is new DEVICE_NUMBER;

-- Provides for stroke device numbers.
```

STROKE_PROMPT_ECHO_TYPE Level 0b

```
type STROKE_PROMPT_ECHO_TYPE is new INTEGER;

-- Defines the stroke prompt and echo types.
```

STROKE_PROMPT_ECHO_TYPES Level 0b

```
package STROKE_PROMPT_ECHO_TYPES is
        new GKS_LIST_UTILITIES (STROKE_PROMPT_ECHO_TYPE);
-- Provides for lists of stroke prompt and echo types.
```

STYLE_INDEX Level 0a

```
type STYLE_INDEX is new INTEGER;

-- A style index is either a HATCH_STYLE or a  PATTERN_STYLE.
```

TEXT_ALIGNMENT Level 0a

```
type TEXT_ALIGNMENT is
        record
            HORIZONTAL      : HORIZONTAL_ALIGNMENT;
            VERTICAL        : VERTICAL_ALIGNMENT;
        end record;

-- The type of the attribute controlling the positioning of the text extent parallelogram
-- in relation to the text position, having horizontal and vertical components as
-- defined above.
```

TEXT_EXTENT_PARALLELOGRAM Level 0a

```
type TEXT_EXTENT_PARALLELOGRAM is
        record
            LOWER_LEFT      : WC.POINT;
            LOWER_RIGHT     : WC.POINT;
            UPPER_RIGHT     : WC.POINT;
            UPPER_LEFT      : WC.POINT;
        end record;

-- Defines the corner points of the text extent parallelogram with respect to the
-- vertical positioning of the text.
```

TEXT_FONT Level 0a

```
type TEXT_FONT is new INTEGER;

-- Defines the types of fonts provided by the implementation.
```

TEXT_FONT_PRECISION Level 0a

```
type TEXT_FONT_PRECISION is
        record
            FONT            : TEXT_FONT;
            PRECISION       : TEXT_PRECISION;
        end record;

-- This type defines a record describing the text font and precision aspect.
```

TEXT_FONT_PRECISIONS Level 0a

```
package TEXT_FONT_PRECISIONS is
        new GKS_LIST_UTILITIES (TEXT_FONT_PRECISION);

-- Provides for lists of text font and precision pairs.
```

TEXT_INDEX Level 0a

```
type TEXT_INDEX is new POSITIVE;

-- Defines the range of text bundle table indices.
```

TEXT_INDICES Level 0a

```
package TEXT_INDICES is new GKS_LIST_UTILITIES (TEXT_INDEX);

-- Provides for lists of text indices.
```

TEXT_PATH Level 0a

```
type TEXT_PATH is (RIGHT, LEFT, UP, DOWN);

-- The direction taken by a text string.
```

TEXT_PRECISION Level 0a

```
type TEXT_PRECISION is (STRING_PRECISION,
                        CHAR_PRECISION,
                        STROKE_PRECISION);

-- The precision with which text appears.
```

TRANSFORMATION_FACTOR Level 1a

```
type TRANSFORMATION_FACTOR is
    record
        X : NDC_TYPE;
        Y : NDC_TYPE;
    end record;

-- Scale factors used in building transformation matrices for performing segment transformations.
```

TRANSFORMATION_MATRIX Level 1a

```
type TRANSFORMATION_MATRIX is array (1..2, 1..3) of NDC_TYPE;

-- For segment transformations mapping within NDC space.
```

TRANSFORMATION_NUMBER Level 0a

```
type TRANSFORMATION_NUMBER is new NATURAL;

-- A normalization transformation number.
```

TRANSFORMATION_PRIORITY_ARRAY Level 0a

```
type TRANSFORMATION_PRIORITY_ARRAY is array (POSITIVE range <>)
      of TRANSFORMATION_NUMBER;

-- Type to store transformation numbers.
```

TRANSFORMATION_PRIORITY_LIST Level 0a

```
type TRANSFORMATION_PRIORITY_LIST(LENGTH : SMALL_NATURAL:=0) is
      record
         CONTENTS : TRANSFORMATION_PRIORITY_ARRAY (1..LENGTH);
      end record;

-- Provides for a prioritised list of transformation numbers.
```

UPDATE_REGENERATION_FLAG Level 0a

```
type UPDATE_REGENERATION_FLAG is (PERFORM, POSTPONE);

-- Flag indicating regeneration action on display.
```

UPDATE_STATE Level 0a

```
type UPDATE_STATE is (NOTPENDING, PENDING);

-- Indicates whether or not a workstation transformation change has been requested and not yet provided.
```

VALUATOR_DEVICE_NUMBER Level 0b

```
type VALUATOR DEVICE_NUMBER is new DEVICE_NUMBER;

-- Provides for valuator device identifiers.
```

VALUATOR_INPUT_VALUE Level 0b

```
type VALUATOR_INPUT_VALUE is digits PRECISION;

-- Defines the range of accuracy of input values on an implementation.
```

VALUATOR_PROMPT_ECHO_TYPE Level 0b

```
type VALUATOR_PROMPT_ECHO_TYPE is new INTEGER;

-- Defines the possible range of valuator prompt and echo types.
```

VALUATOR_PROMPT_ECHO_TYPES Level 0b

```
package VALUATOR_PROMPT_ECHO_TYPES is
        new GKS_LIST_UTILITIES (VALUATOR_PROMPT_ECHO_TYPE);

-- Provides for lists of valuator prompt and echo types.
```

VARIABLE_COLOUR_MATRIX Level 0a

```
type VARIABLE_COLOUR_MATRIX     (DX : SMALL_NATURAL := 0;
                                 DY : SMALL_NATURAL := 0) is
    record
       MATRIX : COLOUR_MATRIX (1..DX, 1..DY);
    end record;

-- Provides for variable sized matrices containing colour indices corresponding to
-- a cell array or pattern array.
```

VARIABLE_CONNECTION_ID Level 0a

```
type VARIABLE_CONNECTION_ID (LENGTH : STRING_SMALL_NATURAL := 0) is
    record
       CONNECT : STRING (1..LENGTH);
    end record;

-- Defines a variable length connection identifier for INQ_WS_CONNECTION_AND_TYPE
```

VARIABLE_PIXEL_COLOUR_MATRIX Level 0a

```
type VARIABLE_PIXEL_COLOUR_MATRIX ( DX : SMALL_NATURAL := 0;
                                    DY : SMALL_NATURAL := 0) is
    record
       MATRIX : PIXEL_COLOUR_MATRIX (1..DX, 1..DY);
    end record;
```

-- Provides for variable sized matrices of pixel colours.

VERTICAL_ALIGNMENT Level 0a

```
type VERTICAL_ALIGNMENT is (NORMAL, TOP, CAP, HALF, BASE, BOTTOM);
```

-- The alignment of the text extent parallelogram with respect to the vertical positioning of the text.

WC Level 0a

```
package WC is new GKS_COORDINATE_SYSTEM (WC_TYPE);
```

-- Defines the World Coordinate System.

WC_TYPE Level 0a

```
type WC_TYPE is digits PRECISION;
```

-- Defines the range of accuracy for World Coordinate types.

WS_CATEGORY Level 0a

```
type WS_CATEGORY is (OUTPUT, INPUT, OUTIN, WISS, MO, MI);
```

-- Type for GKS workstation categories.

WS_ID Level 0a

```
type WS_ID is new POSITIVE;
```

-- Defines the range of workstation identifiers.

WS_IDS Level 0a

```
package WS_IDS is new GKS_LIST_UTILITIES (WS_ID);

-- Provides for lists of workstation identifiers.
```

WS_STATE Level 0a

```
type WS_STATE is (INACTIVE, ACTIVE);

-- The state of a workstation.
```

WS_TYPE Level 0a

```
type WS_TYPE is new POSITIVE;

-- Range of values corresponding to valid workstation types.  Constants specifying names for the
-- various types of workstations should be provided by an implementation.
```

WS_TYPES Level 0a

```
package WS_TYPES is new GKS_LIST_UTILITIES (WS_TYPE);

-- Provides for lists of workstation types.
```

4.2.3 Alphabetical list of Private type definitions

This section contains an alphabetical listing of all the private type definitions used to define the Ada binding to GKS. Each of these declarations specifies the level of GKS at which the data type declaration must be available in an implementation of GKS of that level or any level "above" the level in which the type declaration is first needed (same as for functions). All of these elements are Ada PRIVATE type declarations. These declarations are included in the GKS package to facilitate the manipulations of the private types.

CHOICE_DATA_RECORD Level 0b

```
type CHOICE_DATA_RECORD    (PROMPT_ECHO_TYPE :
                                CHOICE_PROMPT_ECHO_TYPE := DEFAULT_CHOICE) is private;

-- Defines a record for initialising choice input.  The structure of the record is implementation-defined.
-- Since it is a private type, the components of the record may be retrieved only through the use of the
-- subprograms for manipulating the input data records (5.2.1).
```

GKSM_DATA_RECORD Level 0a

```
type GKSM_DATA_RECORD (TYPE_OF_ITEM : GKSM_ITEM_TYPE : =0;
                       LENGTH       : NATURAL : =0) is private;

-- A data record for GKSM metafiles. Since it is a private type, the components of the record may be
--retrieved only through the use of the subprograms for manipulating the input data records
-- (5.2.4).
```

LOCATOR_DATA_RECORD Level 0b

```
type LOCATOR_DATA_RECORD (PROMPT_ECHO_TYPE :
                          LOCATOR_PROMPT_ECHO_TYPE := DEFAULT_LOCATOR)
                          is private;

-- Defines a record for initialising locator input. The structure of the record is implementation-defined.
-- Since it is a private type, the components of the record may be retrieved only through the use of the
-- subprograms for manipulating the input data records ( 5.2.1).
```

PICK_DATA_RECORD Level 0b

```
type PICK_DATA_RECORD (PROMPT_ECHO_TYPE :
                       PICK_PROMPT_ECHO_TYPE := DEFAULT_PICK) is private;

-- Defines a record for initialising pick input. The structure of the record is implementation-defined.
-- Since it is a private type, the components of the record may be retrieved only through the use of the
-- subprograms for manipulating the input data records ( 5.2.1).
```

STRING_DATA_RECORD Level 0b

```
type STRING_DATA_RECORD (PROMPT_ECHO_TYPE :
                         STRING_PROMPT_ECHO_TYPE := DEFAULT_STRING) is private;

-- Defines a record for initialising string input. The structure of the record is implementation-defined.
-- Since it is a private type, the components of the record may be retrieved only through the use of the
-- subprograms for manipulating the input data records (5.2.1).
```

STROKE_DATA_RECORD Level 0b

```
type STROKE_DATA_RECORD (PROMPT_ECHO_TYPE :
                         STROKE_PROMPT_ECHO_TYPE := DEFUALT_STROKE) is private;

-- Defines a record for initialising stroke input. The structure of the record is implementation-defined.
-- Since it is a private type, the components of the record may be retrieved only through the use of the
-- subprograms for manipulating the input data records (5.2.1).
```

VALUATOR_DATA_RECORD Level 0b

```
type VALUATOR_DATA_RECORD   (PROMPT_ECHO_TYPE :
                             VALUATOR_PROMPT_ECHO_TYPE := DEFAULT_VALUATOR)
                             is private;

-- Defines a record for initialising valuator input. The structure of the record is implementation-defined.
-- Since it is a private type, the components of the record may be retrieved only through the use of the
-- subprograms for manipulating the input data records ( 5.2.1).
```

4.2.4 List of constant declarations

This section contains the declarations of implementation dependent constants for defining GKS/Ada types. Some of the constants are used for defining default parameter values for GKS procedures defined in Section 5.0. This section also contains the constants that provide the GKS standard values defined for some GKS/Ada types.

The following constants define the GKS standard line types:

```
SOLID_LINE                  : constant LINETYPE                  := 1;
DASHED_LINE                 : constant LINETYPE                  := 2;
DOTTED_LINE                 : constant LINETYPE                  := 3;
DASHED_DOTTED_LINE          : constant LINETYPE                  := 4;
```

The following constants define the GKS standard marker types:

```
DOT_MARKER                  : constant MARKER_TYPE               := 1;
PLUS_MARKER                 : constant MARKER_TYPE               := 2;
STAR_MARKER                 : constant MARKER_TYPE               := 3;
ZERO_MARKER                 : constant MARKER_TYPE               := 4;
X_MARKER                    : constant MARKER_TYPE               := 5;
```

The following constants define the prompt and echo types supported by GKS:

```
DEFAULT_LOCATOR             : constant LOCATOR_PROMPT_ECHO_TYPE  := 1;
CROSS_HAIR_LOCATOR          : constant LOCATOR_PROMPT_ECHO_TYPE  := 2;
TRACKING_CROSS_LOCATOR      : constant LOCATOR_PROMPT_ECHO_TYPE  := 3;
RUBBER_BAND_LINE_LOCATOR    : constant LOCATOR_PROMPT_ECHO_TYPE  := 4;
RECTANGLE_LOCATOR           : constant LOCATOR_PROMPT_ECHO_TYPE  := 5;
DIGITAL_LOCATOR             : constant LOCATOR_PROMPT_ECHO_TYPE  := 6;

DEFAULT_STROKE              : constant STROKE_PROMPT_ECHO_TYPE   := 1;
DIGITAL_STROKE              : constant STROKE_PROMPT_ECHO_TYPE   := 2;
MARKER_STROKE               : constant STROKE_PROMPT_ECHO_TYPE   := 3;
LINE_STROKE                 : constant STROKE_PROMPT_ECHO_TYPE   := 4;
DEFAULT_VALUATOR            : constant VALUATOR_PROMPT_ECHO_TYPE := 1;
GRAPHICAL_VALUATOR          : constant VALUATOR_PROMPT_ECHO_TYPE := 2;
DIGITAL_VALUATOR            : constant VALUATOR_PROMPT_ECHO_TYPE := 3;

DEFAULT_CHOICE              : constant CHOICE_PROMPT_ECHO_TYPE   := 1;
PROMPT_ECHO_CHOICE          : constant CHOICE_PROMPT_ECHO_TYPE   := 2;
STRING_PROMPT_CHOICE        : constant CHOICE_PROMPT_ECHO_TYPE   := 3;
STRING_INPUT_CHOICE         : constant CHOICE_PROMPT_ECHO_TYPE   := 4;
SEGMENT_CHOICE              : constant CHOICE_PROMPT_ECHO_TYPE   := 5;
```

```
DEFAULT_STRING                : constant STRING_PROMPT_ECHO_TYPE    := 1;

DEFAULT_PICK                  : constant PICK_PROMPT_ECHO_TYPE      := 1;
GROUP_HIGHLIGHT_PICK          : constant PICK_PROMPT_ECHO_TYPE      := 2;
SEGMENT_HIGHLIGHT_PICK        : constant PICK_PROMPT_ECHO_TYPE      := 3;
```

The following constants are used for defining default parameter value for GKS procedures defined in Section 5.0.

```
DEFAULT_MEMORY_UNITS          : constant  := implementation_defined;
PRECISION                     : constant  := implementation_defined;
SMALL_NATURAL_MAX             : constant  := implementation_defined;
CHOICE_SMALL_NATURAL_MAX      : constant  := implementation_defined;
STRING_SMALL_NATURAL_MAX      : constant  := implementation_defined;
DEFAULT_ERROR_FILE            : string    := implementation_defined;
```

The following defines the predefined exception GKS_ERROR defined in 3.2.3.

```
GKS_ERROR                     : exception;
```

4.3 Error codes

This binding requires the use of the GKS procedure ERROR_HANDLING to process any errors that occur in GKS procedures, except the inquiry procedures. A complete description of the error handling requirements of GKS is available in section 3.2.3 of this binding.

The GKS inquiry functions do not raise exceptions. Instead, they return an error indicator parameter containing the number of the "error" which was detected. This is consistent with the GKS philosophy that no errors occur during inquiries. The error numbers correspond to the error numbers from Annex B of the GKS specification, plus additional errors defined in this binding. Note that certain known error conditions may be detected outside the control of GKS due to the nature of the Ada language, and may result in an exception being raised on an inquiry.

4.3.1 Error Code Definition

ISO 7942 provides a mapping of error numbers for each GKS function. Certian of the known GKS errors will never be detected by an Ada GKS implementation due to features of the Ada language, such as strong data typing. These errors are listed in the precluded error codes section.

In addition to the GKS defined errors, there can be errors that are implementation defined, and errors that are defined by this language binding.

IMPLEMENTATION_DEFINED_ERROR

These errors are defined in the User's Manual for an implementation and are in the range less than zero.

LANGUAGE_BINDING_ERROR

Language_Binding_Error indicates an error detected that is specific to this binding of GKS to Ada. Error numbers 2500 to 2999 are reserved for Ada language binding dependent errors. The following error numbers are defined by this binding for the specific identification of language binding errors:

2500 Invalid use of input data record

When the following error occurs, the predefined Ada exception that caused the error is raised automatically.

2501 Unknown error occurred during processing
2502 Usage error in GKS List Utility

4.3.2 Precluded error codes

The following GKS errors are listed separately because due to some feature of the Ada language or its use by this binding, they could never be detected by the GKS implementation. The errors might be detected by the Ada compiler, or at run-time outside the scope of GKS.

Error Codes Precluded by Function

20	Specified workstation identifier is invalid
22	Specified workstation type is invalid
65	Linewidth scale factor is less than zero
71	Marker size scale factor is less than zero
77	Character expansion factor is less than or equal to zero
78	Character height is less than or equal to zero
87	Pattern size value is not positive
91	Dimension of colour array are invalid
92	Colour index is less than zero
96	Colour is outside range [0,1]
97	Pick identifier is invalid
120	Specified segment name is invalid
126	Segment priority is outside the range [0,1]
151	Timeout is invalid
166	Maximum item data record length is invalid

5 FUNCTIONS IN THE ADA BINDING TO GKS

5.1 GKS Functions

OPEN GKS — Level 0a

```
procedure OPEN_GKS
   (ERROR_FILE              : in STRING    := DEFAULT_ERROR_FILE;
    AMOUNT_OF_MEMORY        : in NATURAL   := DEFAULT_MEMORY_UNITS);
```

CLOSE GKS — Level 0a

```
procedure CLOSE_GKS;
```

OPEN WORKSTATION — Level 0a

```
procedure OPEN_WS
   (WS                      : in  WS_ID;
    CONNECTION              : in  STRING;
    TYPE_OF_WS              : in  WS_TYPE);
```

CLOSE WORKSTATION — Level 0a

```
procedure CLOSE_WS
   (WS                      : in WS_ID);
```

ACTIVATE WORKSTATION — Level 0a

```
procedure ACTIVATE_WS
   (WS                      : in WS_ID);
```

DEACTIVATE WORKSTATION — Level 0a

```
procedure DEACTIVATE_WS
   (WS                      : in WS_ID);
```

CLEAR WORKSTATION — Level 0a

```
procedure CLEAR_WS
   (WS                      : in WS_ID;
    FLAG                    : in CONTROL_FLAG);
```

REDRAW ALL SEGMENTS ON WORKSTATION Level 1a

```
procedure REDRAW_ALL_SEGMENTS_ON_WS
   (WS                  : in WS_ID);
```

UPDATE WORKSTATION Level 0a

```
procedure UPDATE_WS
   (WS                  : in WS_ID;
    REGENERATION        : in UPDATE_REGENERATION_FLAG);
```

SET DEFERRAL STATE Level 1a

```
procedure SET_DEFERRAL_STATE
   (WS                  : in WS_ID;
    DEFERRAL            : in DEFERRAL_MODE;
    REGENERATION        : in REGENERATION_MODE);
```

MESSAGE Level 1a

```
procedure MESSAGE
   (WS                  : in WS_ID;
    CONTENTS            : in STRING);
```

ESCAPE Level 0a

Escape functions are bound in Ada as separate procedures for each unique type of escape provided by the implementation, each with a formal parameter list appropriate to the procedure implemented. The registered ESCAPE procedures will be in a library package named GKS_ESCAPE. ESCAPE names and parameters are registered in the ISO International Register of Graphical Items which is maintained by the Registration Authority.

Each unregistered ESCAPE procedure will be a library package using the following naming convention:

```
package GKS_UESC_<name of the escape procedure> is
      procedure ESC;
      -- Ada code for UESC procedure
end GKS_UESC_<name of the escape procedure>;
-- the only procedure name used in the package will be ESC
```

In order to support the ability to write an ESCAPE, that is not implemented, to a metafile these registered ESCAPEs may be invoked using the data types and the form of the procedure GENERALIZED_ESC which have the specifications given below:

```
package GKS_ESCAPE is
      type ESCAPE_ID is new INTEGER;
      type ESCAPE_FLOAT is digits PRECISION;
      type ESC_INTEGER_ARRAY is array (SMALL_NATURAL range <>)
            of INTEGER;
      type ESC_FLOAT_ARRAY is array (SMALL_NATURAL range <>)
            of ESCAPE_FLOAT;
      type ESC_STRING_ARRAY is array (SMALL_NATURAL range <>)
            of STRING (1..80);

      type ESC_DATA_RECORD      (NUM_OF_INTEGERS : SMALL_NATURAL := 0;
                                 NUM_OF_REALS    : SMALL_NATURAL := 0;
                                 NUM_OF_STRINGS  : SMALL_NATURAL := 0) is
      record
        INTEGER_ARRAY : ESC_INTEGER_ARRAY   (1..NUM_OF_INTEGERS);
        REAL_ARRAY    : ESC_FLOAT_ARRAY     (1..NUM_OF_REALS);
        ESC_STRINGS   : ESC_STRING_ARRAY    (1..NUM_OF_STRINGS);
      end record;

      procedure GENERALIZED_ESC (ESCAPE_NAME    : in ESCAPE_ID;
                                 ESC_DATA_IN    : in ESC_DATA_RECORD;
                                 ESC_DATA_OUT   : out ESC_DATA_RECORD);
end GKS_ESCAPE;

-- Provides data types and procedures to implement unsupported ESC'S.
```

POLYLINE Level 0a

```
procedure POLYLINE
   (POINTS              : in WC.POINT_ARRAY);
```

POLYMARKER Level 0a

```
procedure POLYMARKER
   (POINTS              : in WC.POINT_ARRAY);
```

TEXT Level 0a

```
procedure TEXT
   (POSITION            : in WC.POINT;
    CHAR_STRING         : in STRING);
```

FILL AREA Level 0a

```
procedure FILL_AREA
   (POINTS              : in WC.POINT_ARRAY);
```

CELL ARRAY Level 0a

```
procedure CELL_ARRAY
   (CORNER_1_1          : in WC.POINT;
    CORNER_DX_DY        : in WC.POINT;
    CELLS               : in COLOUR_MATRIX);
```

GENERALIZED DRAWING PRIMITIVE Level 0a

The Generalized Drawing Primitive (GDP) is bound in a one-to-many fashion, with a separate procedure implemented for each GDP, each with its own parameter interface. Registered GDP's are in a library package named GKS_GDP. GDP names and parameters are registered in the ISO International Register of Graphical Items which is maintained by the Registration Authority.

Each unregistered GDP procedure will be a library package using the following naming convention:

```
package GKS_UGDP_<name of the GDP procedure> is
      procedure GDP;
        -- Ada code for UGDP procedure
      end GKS_UGDP_<name of the GDP procedure>;
-- The only procedure name used in the package will be GDP
```

In order to support the ability to write a GDP that is not implemented, at a given implementation, to a metafile these registered GDPs may be invoked using the data types and the form of the procedure GENERALIZED_GDP which have the specifications given below:

```
package GKS_GDP is
      type GDP_FLOAT is digits PRECISION;
      type GDP_INTEGER_ARRAY is array (SMALL_NATURAL range <>)
            of INTEGER;
      type GDP_FLOAT_ARRAY is array (SMALL_NATURAL range <>)
            of GDP_FLOAT;
      type GDP_STRING_ARRAY is array (SMALL_NATURAL range <>)
            of STRING (1..80);

      type GDP_DATA_RECORD (NUM_OF_INTEGERS    : SMALL_NATURAL := 0;
                            NUM_OF_REAL        : SMALL_NATURAL := 0;
                            NUM_OF_STRINGS     : SMALL_NATURAL : = 0) is
       record
        INTEGER_ARRAY  : GDP_INTEGER_ARRAY (1..NUM_OF_INTEGERS);
        REAL_ARRAY     : GDP_FLOAT_ARRAY   (1..NUM_OF_REALS);
        GDP_STRINGS    : GDP_STRING_ARRAY  (1..NUM_OF_STRINGS);
      end record;

procedure GENERALIZED_GDP (GDP_NAME    : in GDP_ID;
                           POINTS      : in WC.POINT_LIST;
                           GDP_DATA    : in GDP_DATA_RECORD);
end GKS_GDP;

-- Provides data types and procedure to implement unsupported GDP'S.
```

SET POLYLINE INDEX Level 0a

```
procedure SET_POLYLINE_INDEX
   (INDEX              : in POLYLINE_INDEX);
```

SET LINETYPE Level 0a

```
procedure SET_LINETYPE
   (TYPE_OF_LINE      : in LINETYPE);
```

SET LINEWIDTH SCALE FACTOR Level 0a

```
procedure SET_LINEWIDTH_SCALE_FACTOR
   (WIDTH              : in LINEWIDTH);
```

SET POLYLINE COLOUR INDEX Level 0a

```
procedure SET_POLYLINE_COLOUR_INDEX
   (LINE_COLOUR       : in COLOUR_INDEX);
```

SET POLYMARKER INDEX Level 0a

```
procedure SET_POLYMARKER_INDEX
   (INDEX              : in POLYMARKER_INDEX);
```

SET MARKER TYPE Level 0a

```
procedure SET_MARKER_TYPE
   (TYPE_OF_MARKER          : in MARKER_TYPE);
```

SET MARKER SIZE SCALE FACTOR Level 0a

```
procedure SET_MARKER_SIZE_SCALE_FACTOR
   (SIZE               : in MARKER_SIZE);
```

SET POLYMARKER COLOUR INDEX Level 0a

```
procedure SET_POLYMARKER_COLOUR_INDEX
   (MARKER_COLOUR           : in COLOUR_INDEX);
```

SET TEXT INDEX Level 0a

```
procedure SET_TEXT_INDEX
   (INDEX              : in TEXT_INDEX);
```

SET TEXT FONT AND PRECISION Level 0a

```
procedure SET_TEXT_FONT_AND_PRECISION
   (FONT_PRECISION          : in TEXT_FONT_PRECISION);
```

SET CHARACTER EXPANSION FACTOR Level 0a

```
procedure SET_CHAR_EXPANSION_FACTOR
   (EXPANSION        : in CHAR_EXPANSION);
```

SET CHARACTER SPACING Level 0a

```
procedure SET_CHAR_SPACING
   (SPACING          : in CHAR_SPACING);
```

SET TEXT COLOUR INDEX Level 0a

```
procedure SET_TEXT_COLOUR_INDEX
   (TEXT_COLOUR      : in COLOUR_INDEX);
```

SET CHARACTER HEIGHT Level 0a

```
procedure SET_CHAR_HEIGHT
   (HEIGHT           : in WC.MAGNITUDE);
```

SET CHARACTER UP VECTOR Level 0a

```
procedure SET_CHAR_UP_VECTOR
   (CHAR_UP_VECTOR          : in WC.VECTOR);
```

SET TEXT PATH Level 0a

```
procedure SET_TEXT_PATH
   (PATH              : in TEXT_PATH);
```

SET TEXT ALIGNMENT Level 0a

```
procedure SET_TEXT_ALIGNMENT
   (ALIGNMENT         : in TEXT_ALIGNMENT);
```

SET FILL AREA INDEX Level 0a

```
procedure SET_FILL_AREA_INDEX
   (INDEX             : in FILL_AREA_INDEX);
```

SET FILL AREA INTERIOR STYLE Level 0a

```
procedure SET_FILL_AREA_INTERIOR_STYLE
   (INTERIOR          : in INTERIOR_STYLE);
```

SET FILL AREA STYLE INDEX Level 0a

```
procedure SET_FILL_AREA_STYLE_INDEX
   (STYLE             : in STYLE_INDEX);
```

SET FILL AREA COLOUR INDEX Level 0a

```
procedure SET_FILL_AREA_COLOUR_INDEX
   (FILL_AREA_COLOUR  : in COLOUR_INDEX);
```

SET PATTERN SIZE Level 0a

```
procedure SET_PATTERN_SIZE
   (SIZE              : in WC.SIZE);
```

SET PATTERN REFERENCE POINT Level 0a

```
procedure SET_PATTERN_REFERENCE_POINT
   (POINT             : in WC.POINT);
```

SET ASPECT SOURCE FLAGS — Level 0a

```
procedure SET_ASF
   (ASF                  : in ASF_LIST);
```

SET PICK IDENTIFIER — Level 1b

```
procedure SET_PICK_ID
   (PICK                 : in PICK_ID);
```

SET POLYLINE REPRESENTATION — Level 1a

```
procedure SET_POLYLINE_REPRESENTATION
   (WS                   : in WS_ID;
    INDEX                : in POLYLINE_INDEX;
    TYPE_OF_LINE         : in LINETYPE;
    WIDTH                : in LINEWIDTH;
    LINE_COLOUR          : in COLOUR_INDEX);
```

SET POLYMARKER REPRESENTATION — Level 1a

```
procedure SET_POLYMARKER_REPRESENTATION
   (WS                   : in WS_ID;
    INDEX                : in POLYMARKER_INDEX;
    TYPE_OF_MARKER       : in MARKER_TYPE;
    SIZE                 : in MARKER_SIZE;
    MARKER_COLOUR        : in COLOUR_INDEX);
```

SET TEXT REPRESENTATION — Level 1a

```
procedure SET_TEXT_REPRESENTATION
   (WS                   : in WS_ID;
    INDEX                : in TEXT_INDEX;
    FONT_PRECISION       : in TEXT_FONT_PRECISION;
    EXPANSION            : in CHAR_EXPANSION;
    SPACING              : in CHAR_SPACING;
    TEXT_COLOUR          : in COLOUR_INDEX);
```

SET FILL AREA REPRESENTATION Level 1a

```
procedure SET_FILL_AREA_REPRESENTATION
   (WS                    : in WS_ID;
    INDEX                 : in FILL_AREA_INDEX;
    INTERIOR              : in INTERIOR_STYLE;
    STYLE                 : in STYLE_INDEX;
    FILL_AREA_COLOUR      : in COLOUR_INDEX);
```

SET PATTERN REPRESENTATION Level 1a

```
procedure SET_PATTERN_REPRESENTATION
   (WS                    : in WS_ID;
    INDEX                 : in PATTERN_INDEX;
    PATTERN               : in COLOUR_MATRIX);
```

SET COLOUR REPRESENTATION Level 0a

```
procedure SET_COLOUR_REPRESENTATION
   (WS              : in WS_ID;
    INDEX           : in COLOUR_INDEX;
    RGB_COLOUR      : in COLOUR_REPRESENTATION);
```

SET WINDOW Level 0a

```
procedure SET_WINDOW
   (TRANSFORMATION        : in POSITIVE_TRANSFORMATION_NUMBER;
    WINDOW_LIMITS         : in WC.RECTANGLE_LIMITS);
```

SET VIEWPORT Level 0a

```
procedure SET_VIEWPORT
   (TRANSFORMATION        : in POSITIVE_TRANSFORMATION_NUMBER;
    VIEWPORT_LIMITS       : in NDC.RECTANGLE_LIMITS);
```

SET VIEWPORT INPUT PRIORITY Level 0b

```
procedure SET_VIEWPORT_INPUT_PRIORITY
   (TRANSFORMATION                : in TRANSFORMATION_NUMBER;
    REFERENCE_TRANSFORMATION      : in TRANSFORMATION_NUMBER;
    PRIORITY                      : in RELATIVE_PRIORITY);
```

SELECT NORMALIZATION TRANSFORMATION Level 0a

```
procedure SELECT_NORMALIZATION_TRANSFORMATION
   (TRANSFORMATION          : in TRANSFORMATION_NUMBER);
```

SET CLIPPING INDICATOR Level 0a

```
procedure SET_CLIPPING_INDICATOR
   (CLIPPING                : in CLIPPING_INDICATOR);
```

SET WORKSTATION WINDOW Level 0a

```
procedure SET_WS_WINDOW
   (WS                      : in WS_ID;
    WS_WINDOW_LIMITS        : in NDC.RECTANGLE_LIMITS);
```

SET WORKSTATION VIEWPORT Level 0a

```
procedure SET_WS_VIEWPORT
   (WS                      : in WS_ID;
    WS_VIEWPORT_LIMITS      : in DC.RECTANGLE_LIMITS);
```

CREATE SEGMENT Level 1a

```
procedure CREATE_SEGMENT
   (SEGMENT          : in SEGMENT_NAME);
```

CLOSE SEGMENT Level 1a

```
procedure CLOSE_SEGMENT;
```

RENAME SEGMENT Level 1a

```
procedure RENAME_SEGMENT
   (OLD_NAME         : in SEGMENT_NAME;
    NEW_NAME         : in SEGMENT_NAME);
```

DELETE SEGMENT Level 1a

```
procedure DELETE_SEGMENT
   (SEGMENT          : in SEGMENT_NAME);
```

DELETE SEGMENT FROM WORKSTATION Level 1a

```
procedure DELETE_SEGMENT_FROM_WS
   (WS               : in WS_ID;
    SEGMENT          : in SEGMENT_NAME);
```

ASSOCIATE SEGMENT WITH WORKSTATION Level 2a

```
procedure ASSOCIATE_SEGMENT_WITH_WS
   (WS               : in WS_ID;
    SEGMENT          : in SEGMENT_NAME);
```

COPY SEGMENT TO WORKSTATION Level 2a

```
procedure COPY_SEGMENT_TO_WS
   (WS               : in WS_ID;
    SEGMENT          : in SEGMENT_NAME);
```

INSERT SEGMENT Level 2a

```
procedure INSERT_SEGMENT
   (SEGMENT          : in SEGMENT_NAME;
    TRANSFORMATION   : in TRANSFORMATION_MATRIX);
```

SET SEGMENT TRANSFORMATION Level 1a

```
procedure SET_SEGMENT_TRANSFORMATION
   (SEGMENT          : in SEGMENT_NAME;
    TRANSFORMATION   : in TRANSFORMATION_MATRIX);
```

SET VISIBILITY Level 1a

```
procedure SET_VISIBILITY
   (SEGMENT          : in SEGMENT_NAME;
    VISIBILITY       : in SEGMENT_VISIBILITY);
```

SET HIGHLIGHTING Level 1a

```
procedure SET_HIGHLIGHTING
   (SEGMENT          : in SEGMENT_NAME;
    HIGHLIGHTING     : in SEGMENT_HIGHLIGHTING);
```

SET SEGMENT PRIORITY Level 1a

```
procedure SET_SEGMENT_PRIORITY
   (SEGMENT          : in SEGMENT_NAME;
    PRIORITY         : in SEGMENT_PRIORITY);
```

SET DETECTABILITY Level 1b

```
procedure SET_DETECTABILITY
   (SEGMENT                  : in SEGMENT_NAME;
    DETECTABILITY            : in SEGMENT_DETECTABILITY);
```

INITIALISE LOCATOR Level 0b

```
procedure INITIALISE_LOCATOR
   (WS                       : in WS_ID;
    DEVICE                   : in LOCATOR_DEVICE_NUMBER;
    INITIAL_TRANSFORMATION   : in TRANSFORMATION_NUMBER;
    INITIAL_POSITION         : in WC.POINT;
    ECHO_AREA                : in DC.RECTANGLE_LIMITS;
    DATA_RECORD              : in LOCATOR_DATA_RECORD);
```

INITIALISE STROKE Level 0b

```
procedure INITIALISE_STROKE
   (WS                       : in WS_ID;
    DEVICE                   : in STROKE_DEVICE_NUMBER;
    INITIAL_TRANSFORMATION   : in TRANSFORMATION_NUMBER;
    INITIAL_STROKE           : in WC.POINT_ARRAY;
    ECHO_AREA                : in DC.RECTANGLE_LIMITS;
    DATA_RECORD              : in STROKE_DATA_RECORD);
```

INITIALISE VALUATOR Level 0b

```
procedure INITIALISE_VALUATOR
   (WS              : in WS_ID;
    DEVICE          : in VALUATOR_DEVICE_NUMBER;
    INITIAL_VALUE   : in VALUATOR_INPUT_VALUE;
    ECHO_AREA       : in DC.RECTANGLE_LIMITS;
    DATA_RECORD     : in VALUATOR_DATA_RECORD);
```

INITIALISE CHOICE Level 0b

```
procedure INITIALISE_CHOICE
   (WS                  : in WS_ID;
    DEVICE              : in CHOICE_DEVICE_NUMBER;
    INITIAL_STATUS      : in CHOICE_STATUS;
    INITIAL_CHOICE      : in CHOICE_VALUE;
    ECHO_AREA           : in DC.RECTANGLE_LIMITS;
    DATA_RECORD         : in CHOICE_DATA_RECORD);
```

INITIALISE PICK Level 1b

```
procedure INITIALISE_PICK
   (WS                  : in WS_ID
    DEVICE              : in PICK_DEVICE_NUMBER;
    INITIAL_STATUS      : in PICK_STATUS;
    INITIAL_SEGMENT     : in SEGMENT_NAME;
    INITIAL_PICK        : in PICK_ID;
    ECHO_AREA           : in DC.RECTANGLE_LIMITS;
    DATA_RECORD         : in PICK_DATA_RECORD);
```

INITIALISE STRING Level 0b

```
procedure INITIALISE_STRING
   (WS              : in WS_ID;
    DEVICE          : in STRING_DEVICE_NUMBER;
    INITIAL_STRING  : in INPUT_STRING;
    ECHO_AREA       : in DC.RECTANGLE_LIMITS;
    DATA_RECORD     : in STRING_DATA_RECORD);
```

SET LOCATOR MODE Level 0b

```
procedure SET_LOCATOR_MODE
   (WS              : in WS_ID;
    DEVICE          : in LOCATOR_DEVICE_NUMBER;
    MODE            : in OPERATING_MODE;
    SWITCH          : in ECHO_SWITCH);
```

SET STROKE MODE Level 0b

```
procedure SET_STROKE_MODE
   (WS          : in WS_ID;
    DEVICE      : in STROKE_DEVICE_NUMBER;
    MODE        : in OPERATING_MODE;
    SWITCH      : in ECHO_SWITCH);
```

SET VALUATOR MODE Level 0b

```
procedure SET_VALUATOR_MODE
   (WS          : in WS_ID;
    DEVICE      : in VALUATOR_DEVICE_NUMBER;
    MODE        : in OPERATING_MODE;
    SWITCH      : in ECHO_SWITCH);
```

SET CHOICE MODE Level 0b

```
procedure SET_CHOICE_MODE
   (WS          : in WS_ID;
    DEVICE      : in CHOICE_DEVICE_NUMBER;
    MODE        : in OPERATING_MODE;
    SWITCH      : in ECHO_SWITCH);
```

SET PICK MODE Level 1b

```
procedure SET_PICK_MODE
   (WS          : in WS_ID;
    DEVICE      : in PICK_DEVICE_NUMBER;
    MODE        : in OPERATING_MODE;
    SWITCH      : in ECHO_SWITCH);
```

SET STRING MODE Level 0b

```
procedure SET_STRING_MODE
   (WS          : in WS_ID;
    DEVICE      : in STRING_DEVICE_NUMBER;
    MODE        : in OPERATING_MODE;
    SWITCH      : in ECHO_SWITCH);
```

REQUEST LOCATOR Level 0b

```
procedure REQUEST_LOCATOR
    (WS                 : in WS_ID;
     DEVICE             : in LOCATOR_DEVICE_NUMBER;
     STATUS             : out INPUT_STATUS;
     TRANSFORMATION     : out TRANSFORMATION_NUMBER;
     POSITION           : out WC.POINT);
```

REQUEST STROKE Level 0b

```
procedure REQUEST_STROKE
    (WS                 : in WS_ID;
     DEVICE             : in STROKE_DEVICE_NUMBER;
     STATUS             : out INPUT_STATUS;
     TRANSFORMATION     : out TRANSFORMATION_NUMBER;
     STROKE_POINTS      : out WC.POINT_LIST);
```

REQUEST VALUATOR Level 0b

```
procedure REQUEST_VALUATOR
    (WS                 : in WS_ID;
     DEVICE             : in VALUATOR_DEVICE_NUMBER;
     STATUS             : out INPUT_STATUS;
     VALUE              : out VALUATOR_INPUT_VALUE);
```

REQUEST CHOICE Level 0b

```
procedure REQUEST_CHOICE
    (WS                 : in WS_ID;
     DEVICE             : in CHOICE_DEVICE_NUMBER;
     STATUS             : out CHOICE_REQUEST_STATUS;
     CHOICE_NUMBER      : out CHOICE_VALUE);
```

REQUEST PICK Level 1b

```
procedure REQUEST_PICK
    (WS                 : in WS_ID;
     DEVICE             : in PICK_DEVICE_NUMBER;
     STATUS             : out PICK_REQUEST_STATUS;
     SEGMENT            : out SEGMENT_NAME;
     PICK               : out PICK_ID);
```

REQUEST STRING Level 0b

```
procedure REQUEST_STRING
   (WS              : in WS_ID;
    DEVICE          : in STRING_DEVICE_NUMBER;
    STATUS          : out INPUT_STATUS;
    CHAR_STRING     : out INPUT_STRING);
```

SAMPLE LOCATOR Level 0c

```
procedure SAMPLE_LOCATOR
   (WS                  : in WS_ID;
    DEVICE              : in LOCATOR_DEVICE_NUMBER;
    TRANSFORMATION      : out TRANSFORMATION_NUMBER;
    POSITION            : out WC.POINT);
```

SAMPLE STROKE Level 0c

```
procedure SAMPLE_STROKE
   (WS                  : in WS_ID;
    DEVICE              : in STROKE_DEVICE_NUMBER;
    TRANSFORMATION      : out TRANSFORMATION_NUMBER;
    STROKE_POINTS       : out WC.POINT_LIST);
```

SAMPLE VALUATOR Level 0c

```
procedure SAMPLE_VALUATOR
   (WS              : in WS_ID;
    DEVICE          : in VALUATOR_DEVICE_NUMBER;
    VALUE           : out VALUATOR_INPUT_VALUE);
```

SAMPLE CHOICE Level 0c

```
procedure SAMPLE_CHOICE
   (WS              : in WS_ID;
    DEVICE          : in CHOICE_DEVICE_NUMBER;
    STATUS          : out CHOICE_STATUS;
    CHOICE_NUMBER   : out CHOICE_VALUE);
```

SAMPLE PICK Level 1c

```
procedure SAMPLE_PICK
   (WS                  : in WS_ID;
    DEVICE              : in PICK_DEVICE_NUMBER;
    STATUS              : out PICK_STATUS;
    SEGMENT             : out SEGMENT_NAME;
    PICK                : out PICK_ID);
```

SAMPLE STRING Level 0c

```
procedure SAMPLE_STRING
   (WS                  : in WS_ID;
    DEVICE              : in STRING_DEVICE_NUMBER;
    CHAR_STRING         : out INPUT_STRING);
```

AWAIT EVENT Level 0c

```
procedure AWAIT_EVENT
   (TIMEOUT             : in DURATION;
    WS                  : out WS_ID;
    CLASS               : out INPUT_CLASS;
    DEVICE              : out EVENT_DEVICE_NUMBER);
```

FLUSH DEVICE EVENTS Level 0c

```
procedure FLUSH_DEVICE_EVENTS
   (WS                  : in WS_ID;
    CLASS               : in INPUT_QUEUE_CLASS;
    DEVICE              : in EVENT_OVERFLOW_DEVICE_NUMBER);
```

GET LOCATOR Level 0c

```
procedure GET_LOCATOR
   (TRANSFORMATION      : out TRANSFORMATION_NUMBER;
    POSITION            : out WC.POINT);
```

GET STROKE Level 0c

```
procedure GET_STROKE
   (TRANSFORMATION      : out TRANSFORMATION_NUMBER;
    STROKE_POINTS       : out WC.POINT_LIST);
```

GET VALUATOR Level 0c

```
procedure GET_VALUATOR
   (VALUE              : out VALUATOR_INPUT_VALUE);
```

GET CHOICE Level 0c

```
procedure GET_CHOICE
   (STATUS             : out CHOICE_STATUS;
    CHOICE_NUMBER      : out CHOICE_VALUE);
```

GET PICK Level 1c

```
procedure GET_PICK
   (STATUS             : out PICK_STATUS;
    SEGMENT            : out SEGMENT_NAME;
    PICK               : out PICK_ID);
```

GET STRING Level 0c

```
procedure GET_STRING
   (CHAR_STRING        : out INPUT_STRING);
```

WRITE ITEM TO GKSM Level 0a

```
procedure WRITE_ITEM_TO_GKSM
   (WS                 : in WS_ID;
    ITEM               : in GKSM_DATA_RECORD);
```

GET ITEM TYPE FROM GKSM Level 0a

```
procedure GET_ITEM_TYPE_FROM_GKSM
   (WS                 : in WS_ID;
    TYPE_OF_ITEM       : out GKSM_ITEM_TYPE;
    LENGTH             : out NATURAL);
```

READ ITEM FROM GKSM Level 0a

```
procedure READ_ITEM_FROM_GKSM
   (WS                  : in WS_ID;
    MAX_LENGTH          : in NATURAL;
    ITEM                : out GKSM_DATA_RECORD);
```

INTERPRET ITEM Level 0a

```
procedure INTERPRET_ITEM
   (ITEM                : in GKSM_DATA_RECORD);
```

INQUIRE OPERATING STATE VALUE Level 0a

```
procedureINQ_OPERATING_STATE_VALUE
   (VALUE               :out OPERATING_STATE);
```

INQUIRE LEVEL OF GKS Level 0a

```
procedure INQ_LEVEL_OF_GKS
   (ERROR_INDICATOR     : out ERROR_NUMBER;
    LEVEL               : out GKS_LEVEL);
```

INQUIRE LIST OF AVAILABLE WORKSTATION TYPES Level 0a

```
procedure INQ_LIST_OF_AVAILABLE_WS_TYPES
   (ERROR_INDICATOR     : out ERROR_NUMBER;
    TYPES               : out WS_TYPES.LIST_OF);
```

INQUIRE WORKSTATION MAXIMUM NUMBERS Level 1a

```
procedure INQ_WS_MAX_NUMBERS
   (ERROR_INDICATOR     : out ERROR_NUMBER;
    MAX_OPEN_WS         : out POSITIVE;
    MAX_ACTIVE_WS       : out POSITIVE;
    MAX_SEGMENT_WS      : out POSITIVE);
```

INQUIRE MAXIMUM NORMALIZATION TRANSFORMATION NUMBER Level 0a

```
procedure INQ_MAX_NORMALIZATION_TRANSFORMATION_NUMBER
   (ERROR_INDICATOR     : out ERROR_NUMBER;
    TRANSFORMATION      : out TRANSFORMATION_NUMBER);
```

INQUIRE SET OF OPEN WORKSTATIONS Level 0a

```
procedure INQ_SET_OF_OPEN_WS
   (ERROR_INDICATOR        : out ERROR_NUMBER;
    WS                     : out WS_IDS.LIST_OF);
```

INQUIRE SET OF ACTIVE WORKSTATIONS Level 1a

```
procedure INQ_SET_OF_ACTIVE_WS
   (ERROR_INDICATOR        : out ERROR_NUMBER;
    WS                     : out WS_IDS.LIST_OF);
```

INQUIRE CURRENT PRIMITIVE ATTRIBUTE VALUES Level 0a

```
procedure INQ_CURRENT_PRIMITIVE_ATTRIBUTE_VALUES
   (ERROR_INDICATOR        : out ERROR_NUMBER;
    ATTRIBUTES             : out PRIMITIVE_ATTRIBUTE_VALUES);

procedure INQ_POLYLINE_INDEX
   (ERROR_INDICATOR        : out ERROR_NUMBER;
    INDEX                  : out POLYLINE_INDEX);

procedure INQ_POLYMARKER_INDEX
   (ERROR_INDICATOR        : out ERROR_NUMBER;
    INDEX                  : out POLYMARKER_INDEX);

procedure INQ_TEXT_INDEX
   (ERROR_INDICATOR        : out ERROR_NUMBER;
    INDEX                  : out TEXT_INDEX);

procedure INQ_CHAR_HEIGHT
   (ERROR_INDICATOR        : out ERROR_NUMBER;
    HEIGHT                 : out WC.MAGNITUDE);

procedure INQ_CHAR_UP_VECTOR
   (ERROR_INDICATOR        : out ERROR_NUMBER;
    VECTOR                 : out WC.VECTOR);

procedure INQ_CHAR_WIDTH
   (ERROR_INDICATOR        : out ERROR_NUMBER;
    WIDTH                  : out WC.MAGNITUDE);

procedure INQ_CHAR_BASE_VECTOR
   (ERROR_INDICATOR        : out ERROR_NUMBER;
    VECTOR                 : out WC.VECTOR);

procedure INQ_TEXT_PATH
   (ERROR_INDICATOR        : out ERROR_NUMBER;
    PATH                   : out TEXT_PATH);
```

```
procedure INQ_TEXT_ALIGNMENT
    (ERROR_INDICATOR        : out ERROR_NUMBER;
     ALIGNMENT              : out TEXT_ALIGNMENT);

procedure INQ_FILL_AREA_INDEX
    (ERROR_INDICATOR        : out ERROR_NUMBER;
     INDEX                  : out FILL_AREA_INDEX);

procedure INQ_PATTERN_WIDTH_VECTOR
    (ERROR_INDICATOR        : out ERROR_NUMBER;
     WIDTH                  : out WC.VECTOR);

procedure INQ_PATTERN_HEIGHT_VECTOR
    (ERROR_INDICATOR        : out ERROR_NUMBER;
     VECTOR                 : out WC.VECTOR);

procedure INQ_PATTERN_REFERENCE_POINT
    (ERROR_INDICATOR        : out ERROR_NUMBER;
     REFERENCE_POINT        : out WC.POINT);
```

INQUIRE CURRENT PICK IDENTIFIER VALUE — Level 1b

```
procedure INQ_CURRENT_PICK_ID_VALUE
    (ERROR_INDICATOR        : out ERROR_NUMBER;
     PICK                   : out PICK_ID);
```

INQUIRE CURRENT INDIVIDUAL ATTRIBUTE VALUES — Level 0a

```
procedure INQ_CURRENT_INDIVIDUAL_ATTRIBUTE_VALUES
    (ERROR_INDICATOR        : out ERROR_NUMBER;
     ATTRIBUTES             : out INDIVIDUAL_ATTRIBUTE_VALUES);

procedure INQ_LINETYPE
    (ERROR_INDICATOR        : out ERROR_NUMBER;
     TYPE_OF_LINE           : out LINETYPE);

procedure INQ_LINEWIDTH_SCALE_FACTOR
    (ERROR_INDICATOR        : out ERROR_NUMBER;
     WIDTH                  : out LINEWIDTH);

procedure INQ_POLYLINE_COLOUR_INDEX
    (ERROR_INDICATOR        : out ERROR_NUMBER;
     LINE_COLOUR            : out COLOUR_INDEX);

procedure INQ_POLYMARKER_TYPE
    (ERROR_INDICATOR        : out ERROR_NUMBER;
     TYPE_OF_MARKER         : out MARKER_TYPE);

procedure INQ_POLYMARKER_SIZE_SCALE_FACTOR
    (ERROR_INDICATOR        : out ERROR_NUMBER;
     SIZE                   : out MARKER_SIZE);
```

```
procedure INQ_POLYMARKER_COLOUR_INDEX
   (ERROR_INDICATOR          : out ERROR_NUMBER;
    MARKER_COLOUR            : out COLOUR_INDEX);

proccdurc INQ_TEXT_FONT_AND_PRECISION
   (ERROR_INDICATOR          : out ERROR_NUMBER;
    FONT_PRECISION           : out TEXT_FONT_PRECISION);

procedurc INQ_CHAR_EXPANSION_FACTOR
   (ERROR_INDICATOR          : out ERROR_NUMBER;
    EXPANSION                : out CHAR_EXPANSION);

procedure INQ_CHAR_SPACING
   (ERROR_INDICATOR          : out ERROR_NUMBER;
    SPACING                  : out CHAR_SPACING);

procedure INQ_TEXT_COLOUR_INDEX
   (ERROR_INDICATOR          : out ERROR_NUMBER;
    TEXT_COLOUR              : out COLOUR_INDEX);

procedure INQ_FILL_AREA_INTERIOR_STYLE
   (ERROR_INDICATOR          : out ERROR_NUMBER;
    INTERIOR                 : out INTERIOR_STYLE);

procedure INQ_FILL_AREA_STYLE_INDEX
   (ERROR_INDICATOR          : out ERROR_NUMBER;
    STYLE                    : out STYLE_INDEX);

procedure INQ_FILL_AREA_COLOUR_INDEX
   (ERROR_INDICATOR          : out ERROR_NUMBER;
    FILL_AREA_COLOUR         : out COLOUR_INDEX);

procedure INQ_LIST_OF_ASF
   (ERROR_INDICATOR          : out ERROR_NUMBER;
    LIST                     : out ASF_LIST);
```

INQUIRE CURRENT NORMALIZATION TRANSFORMATION NUMBER — Level 0a

```
procedure INQ_CURRENT_NORMALIZATION_TRANSFORMATION_NUMBER
   (ERROR_INDICATOR          : out ERROR_NUMBER;
    TRANSFORMATION           : out TRANSFORMATION_NUMBER);
```

INQUIRE LIST OF NORMALIZATION TRANSFORMATION NUMBERS — Level 0a

```
procedure INQ_LIST_OF_NORMALIZATION_TRANSFORMATION_NUMBERS
   (ERROR_INDICATOR          : out ERROR_NUMBER;
    LIST                     : out TRANSFORMATION_PRIORITY_LIST);
```

INQUIRE NORMALIZATION TRANSFORMATION Level 0a

```
procedure INQ_NORMALIZATION_TRANSFORMATION
   (TRANSFORMATION          : in TRANSFORMATION_NUMBER;
    ERROR_INDICATOR         : out ERROR_NUMBER;
    WINDOW_LIMITS           : out WC.RECTANGLE_LIMITS;
    VIEWPORT_LIMITS         : out NDC.RECTANGLE_LIMITS);
```

INQUIRE CLIPPING Level 0a

```
procedure INQ_CLIPPING
   (ERROR_INDICATOR         : out ERROR_NUMBER;
    CLIPPING                : out CLIPPING_INDICATOR;
    CLIPPING_RECTANGLE      : out NDC.RECTANGLE_LIMITS);
```

INQUIRE NAME OF OPEN SEGMENT Level 1a

```
procedure INQ_NAME_OF_OPEN_SEGMENT
   (ERROR_INDICATOR         : out ERROR_NUMBER;
    SEGMENT                 : out SEGMENT_NAME);
```

INQUIRE SET OF SEGMENT NAMES IN USE Level 1a

```
procedure INQ_SET_OF_SEGMENT_NAMES_IN_USE
   (ERROR_INDICATOR         : out ERROR_NUMBER;
    SEGMENTS                : out SEGMENT_NAMES.LIST_OF);
```

INQUIRE MORE SIMULTANEOUS EVENTS Level 0c

```
procedure INQ_MORE_SIMULTANEOUS_EVENTS
   (ERROR_INDICATOR         : out ERROR_NUMBER;
    EVENTS                  : out MORE_EVENTS);
```

INQUIRE WORKSTATION CONNECTION AND TYPE Level 0a

```
procedure INQ_WS_CONNECTION_AND_TYPE
   (WS                      : in WS_ID;
    ERROR_INDICATOR         : out ERROR_NUMBER;
    CONNECTION              : out VARIABLE_CONNECTION_ID;
    TYPE_OF_WS              : out WS_TYPE);
```

INQUIRE WORKSTATION STATE Level 0a

```
procedure INQ_WS_STATE
   (WS                  : in WS_ID;
    ERROR_INDICATOR     : out ERROR_NUMBER;
    STATE               : out WS_STATE);
```

INQUIRE WORKSTATION DEFERRAL AND UPDATE STATES Level 0a

```
procedure INQ_WS_DEFERRAL_AND_UPDATE_STATES
   (WS                  : in WS_ID;
    ERROR_INDICATOR     : out ERROR_NUMBER;
    DEFERRAL            : out DEFERRAL_MODE;
    REGENERATION        : out REGENERATION_MODE;
    DISPLAY             : out DISPLAY_SURFACE_EMPTY;
    FRAME_ACTION        : out NEW_FRAME_NECESSARY);
```

INQUIRE LIST OF POLYLINE INDICES Level 1a

```
procedure INQ_LIST_OF_POLYLINE_INDICES
   (WS                  : in WS_ID;
    ERROR_INDICATOR     : out ERROR_NUMBER;
    INDICES             : out POLYLINE_INDICES.LIST_OF);
```

INQUIRE POLYLINE REPRESENTATION Level 1a

```
procedure INQ_POLYLINE_REPRESENTATION
   (WS                  : in WS_ID;
    INDEX               : in POLYLINE_INDEX;
    RETURNED_VALUES     : in RETURN_VALUE_TYPE;
    ERROR_INDICATOR     : out ERROR_NUMBER;
    TYPE_OF_LINE        : out LINETYPE;
    WIDTH               : out LINEWIDTH;
    LINE_COLOUR         : out COLOUR_INDEX);
```

INQUIRE LIST OF POLYMARKER INDICES Level 1a

```
procedure INQ_LIST_OF_POLYMARKER_INDICES
   (WS                  : in WS_ID;
    ERROR_INDICATOR     : out ERROR_NUMBER;
    INDICES             : out POLYMARKER_INDICES.LIST_OF);
```

INQUIRE POLYMARKER REPRESENTATION Level 1a

```
procedure INQ_POLYMARKER_REPRESENTATION
   (WS                    : in WS_ID;
    INDEX                 : in POLYMARKER_INDEX;
    RETURNED_VALUES       : in RETURN_VALUE_TYPE;
    ERROR_INDICATOR       : out ERROR_NUMBER;
    TYPE_OF_MARKER        : out MARKER_TYPE;
    SIZE                  : out MARKER_SIZE;
    MARKER_COLOUR         : out COLOUR_INDEX);
```

INQUIRE LIST OF TEXT INDICES Level 1a

```
procedure INQ_LIST_OF_TEXT_INDICES
   (WS                    : in WS_ID;
    ERROR_INDICATOR       : out ERROR_NUMBER;
    INDICES               : out TEXT_INDICES.LIST_OF);
```

INQUIRE TEXT REPRESENTATION Level 1a

```
procedure INQ_TEXT_REPRESENTATION
   (WS                    : in WS_ID;
    INDEX                 : in TEXT_INDEX;
    RETURNED_VALUES       : in RETURN_VALUE_TYPE;
    ERROR_INDICATOR       : out ERROR_NUMBER;
    FONT_PRECISION        : out TEXT_FONT_PRECISION;
    EXPANSION             : out CHAR_EXPANSION;
    SPACING               : out CHAR_SPACING;
    TEXT_COLOUR           : out COLOUR_INDEX);
```

INQUIRE TEXT EXTENT Level 0a

```
procedure INQ_TEXT_EXTENT
   (WS                    : in WS_ID;
    POSITION              : in WC.POINT;
    CHAR_STRING           : in STRING;
    ERROR_INDICATOR       : out ERROR_NUMBER;
    CONCATENATION_POINT   : out WC.POINT;
    TEXT_EXTENT           : out TEXT_EXTENT_PARALLELOGRAM);
```

INQUIRE LIST OF FILL AREA INDICES Level 1a

```
procedure INQ_LIST_OF_FILL_AREA_INDICES
   (WS                    : in WS_ID;
    ERROR_INDICATOR       : out ERROR_NUMBER;
    INDICES               : out FILL_AREA_INDICES.LIST_OF);
```

INQUIRE FILL AREA REPRESENTATION Level 1a

```
procedure INQ_FILL_AREA_REPRESENTATION
   (WS                    : in WS_ID;
    INDEX                 : in FILL_AREA_INDEX;
    RETURNED_VALUES       : in RETURN_VALUE_TYPE;
    ERROR_INDICATOR       : out ERROR_NUMBER;
    INTERIOR              : out INTERIOR_STYLE;
    STYLE                 : out STYLE_INDEX;
    FILL_AREA_COLOUR      : out COLOUR_INDEX);
```

INQUIRE LIST OF PATTERN INDICES Level 1a

```
procedure INQ_LIST_OF_PATTERN_INDICES
   (WS                    : in WS_ID;
    ERROR_INDICATOR       : out ERROR_NUMBER;
    INDICES               : out PATTERN_INDICES.LIST_OF);
```

INQUIRE PATTERN REPRESENTATION Level 1a

```
procedure INQ_PATTERN_REPRESENTATION
   (WS                    : in WS_ID;
    INDEX                 : in PATTERN_INDEX;
    RETURNED_VALUES       : in RETURN_VALUE_TYPE;
    ERROR_INDICATOR       : out ERROR_NUMBER;
    PATTERN               : out VARIABLE_COLOUR_MATRIX);
```

INQUIRE LIST OF COLOUR INDICES Level 0a

```
procedure INQ_LIST_OF_COLOUR_INDICES
   (WS                    : in WS_ID;
    ERROR_INDICATOR       : out ERROR_NUMBER;
    INDICES               : out COLOUR_INDICES.LIST_OF);
```

INQUIRE COLOUR REPRESENTATION Level 0a

```
procedure INQ_COLOUR_REPRESENTATION
   (WS                    : in WS_ID;
    INDEX                 : in COLOUR_INDEX;
    RETURNED_VALUES       : in RETURN_VALUE_TYPE;
    ERROR_INDICATOR       : out ERROR_NUMBER;
    RGB_COLOUR            : out COLOUR_REPRESENTATION);
```

INQUIRE WORKSTATION TRANSFORMATION Level 0a

```
procedure INQ_WS_TRANSFORMATION
    (WS                         : in WS_ID;
     ERROR_INDICATOR            : out ERROR_NUMBER;
     UPDATE                     : out UPDATE_STATE;
     REQUESTED_WINDOW           : out NDC.RECTANGLE_LIMITS;
     CURRENT_WINDOW             : out NDC.RECTANGLE_LIMITS;
     REQUESTED_VIEWPORT         : out DC.RECTANGLE_LIMITS;
     CURRENT_VIEWPORT           : out DC.RECTANGLE_LIMITS);
```

INQUIRE SET OF SEGMENT NAMES ON WORKSTATION Level 1a

```
procedure INQ_SET_OF_SEGMENT_NAMES_ON_WS
    (WS                         : in WS_ID;
     ERROR_INDICATOR            : out ERROR_NUMBER;
     SEGMENTS                   : out SEGMENT_NAMES.LIST_OF);
```

INQUIRE LOCATOR DEVICE STATE Level 0b

```
procedure INQ_LOCATOR_DEVICE_STATE
    (WS                         : in WS_ID;
     DEVICE                     : in LOCAOTR_DEVICE_NUMBER;
     RETURNED_VALUES            : in RETURN_VALUE_TYPE;
     ERROR_INDICATOR            : out ERROR_NUMBER;
     MODE                       : out OPERATING_MODE;
     SWITCH                     : out ECHO_SWITCH;
     INITIAL_TRANSFORMATION     : out TRANSFORMATION_NUMBER;
     INITIAL_POSITION           : out WC.POINT;
     ECHO_AREA                  : out DC.RECTANGLE_LIMITS;
     DATA_RECORD                : out LOCATOR_DATA_RECORD);
```

INQUIRE STROKE DEVICE STATE Level 0b

```
procedure INQ_STROKE_DEVICE_STATE
    (WS                         : in WS_ID;
     DEVICE                     : in STROKE_DEVICE_NUMBER;
     RETURNED_VALUES            : in RETURN_VALUE_TYPE;
     ERROR_INDICATOR            : out ERROR_NUMBER;
     MODE                       : out OPERATING_MODE;
     SWITCH                     : out ECHO_SWITCH;
     INITIAL_TRANSFORMATION     : out TRANSFORMATION_NUMBER;
     INITIAL_STROKE_POINTS      : out WC.POINT_LIST;
     ECHO_AREA                  : out DC.RECTANGLE_LIMITS;
     DATA_RECORD                : out STROKE_DATA_RECORD);
```

INQUIRE VALUATOR DEVICE STATE Level 0b

```
procedure INQ_VALUATOR_DEVICE_STATE
   (WS                    : in WS_ID;
    DEVICE                : in VALUATOR_DEVICE_NUMBER;
    ERROR_INDICATOR       : out ERROR_NUMBER;
    MODE                  : out OPERATING_MODE;
    SWITCH                : out ECHO_SWITCH;
    INITIAL_VALUE         : out VALUATOR_INPUT_VALUE;
    ECHO_AREA             : out DC.RECTANGLE_LIMITS;
    DATA_RECORD           : out VALUATOR_DATA_RECORD);
```

INQUIRE CHOICE DEVICE STATE Level 0b

```
procedure INQ_CHOICE_DEVICE_STATE
   (WS                    : in WS_ID;
    DEVICE                : in CHOICE_DEVICE_NUMBER;
    ERROR_INDICATOR       : out ERROR_NUMBER;
    MODE                  : out OPERATING_MODE;
    SWITCH                : out ECHO_SWITCH;
    INITIAL_STATUS        : out CHOICE_STATUS;
    INITIAL_CHOICE        : out CHOICE_VALUE;
    ECHO_AREA             : out DC.RECTANGLE_LIMITS;
    DATA_RECORD           : out CHOICE_DATA_RECORD);
```

INQUIRE PICK DEVICE STATE Level 1b

```
procedure INQ_PICK_DEVICE_STATE
   (WS                    : in WS_ID;
    DEVICE                : in PICK_DEVICE_NUMBER;
    RETURNED_VALUES       : in RETURN_VALUE_TYPE;
    ERROR_INDICATOR       : out ERROR_NUMBER;
    MODE                  : out OPERATING_MODE;
    SWITCH                : out ECHO_SWITCH;
    INITIAL_STATUS        : out PICK_STATUS;
    INITIAL_SEGMENT       : out SEGMENT_NAME;
    INITIAL_PICK          : out PICK_ID;
    ECHO_AREA             : out DC.RECTANGLE_LIMITS;
    DATA_RECORD           : out PICK_DATA_RECORD);
```

INQUIRE STRING DEVICE STATE Level 0b

```
procedure INQ_STRING_DEVICE_STATE
   (WS                    : in WS_ID;
    DEVICE                : in STRING_DEVICE_NUMBER;
    ERROR_INDICATOR       : out ERROR_NUMBER;
    MODE                  : out OPERATING_MODE;
    SWITCH                : out ECHO_SWITCH;
    INITIAL_STRING        : out INPUT_STRING;
    ECHO_AREA             : out DC.RECTANGLE_LIMITS;
    DATA_RECORD           : out STRING_DATA_RECORD);
```

INQUIRE WORKSTATION CATEGORY Level 0a

```
procedure INQ_WS_CATEGORY
   (TYPE_OF_WS            : in WS_TYPE;
    ERROR_INDICATOR       : out ERROR_NUMBER;
    CATEGORY              : out WS_CATEGORY);
```

INQUIRE WORKSTATION CLASSIFICATION Level 0a

```
procedure INQ_WS_CLASSIFICATION
   (TYPE_OF_WS            : in WS_TYPE;
    ERROR_INDICATOR       : out ERROR_NUMBER;
    CLASS                 : out DISPLAY_CLASS);
```

INQUIRE DISPLAY SPACE SIZE Level 0a

```
procedure INQ_DISPLAY_SPACE_SIZE
   (TYPE_OF_WS            : in WS_TYPE;
    ERROR_INDICATOR       : out ERROR_NUMBER;
    UNITS                 : out DC_UNITS;
    MAX_DC_SIZE           : out DC.SIZE;
    MAX_RASTER_UNIT_SIZE  : out RASTER_UNIT_SIZE);
```

INQUIRE DYNAMIC MODIFICATION OF WORKSTATION ATTRIBUTES Level 1a

```
procedure INQ_DYNAMIC_MODIFICATION_OF_WS_ATTRIBUTES
   (TYPE_OF_WS                    : in WS_TYPE;
    ERROR_INDICATOR               : out ERROR_NUMBER;
    POLYLINE_REPRESENTATION       : out DYNAMIC_MODIFICATION;
    POLYMARKER_REPRESENTATION     : out DYNAMIC_MODIFICATION;
    TEXT_REPRESENTATION           : out DYNAMIC_MODIFICATION;
    FILL_AREA_REPRESENTATION      : out DYNAMIC_MODIFICATION;
    PATTERN_REPRESENTATION        : out DYNAMIC_MODIFICATION;
    COLOUR_REPRESENTATION         : out DYNAMIC_MODIFICATION;
    TRANSFORMATION                : out DYNAMIC_MODIFICATION);
```

INQUIRE DEFAULT DEFERRAL STATE VALUES Level 1a

```
procedure INQ_DEFAULT_DEFERRAL_STATE_VALUES
   (TYPE_OF_WS                    : in WS_TYPE;
    ERROR_INDICATOR               : out ERROR_NUMBER;
    DEFERRAL                      : out DEFERRAL_MODE;
    REGENERATION                  : out REGENERATION_MODE);
```

INQUIRE POLYLINE FACILITIES Level 0a

```
procedure INQ_POLYLINE_FACILITIES
   (TYPE_OF_WS                    : in WS_TYPE;
    ERROR_INDICATOR               : out ERROR_NUMBER;
    LIST_OF_TYPES                 : out LINETYPES.LIST_OF;
    NUMBER_OF_WIDTHS              : out NATURAL;
    NOMINAL_WIDTH                 : out DC.MAGNITUDE;
    RANGE_OF_WIDTHS               : out DC.RANGE_OF_MAGNITUDES;
    NUMBER_OF_INDICES             : out NATURAL);
```

INQUIRE PREDEFINED POLYLINE REPRESENTATION Level 0a

```
procedure INQ_PREDEFINED_POLYLINE_REPRESENTATION
   (TYPE_OF_WS                    : in WS_TYPE;
    INDEX                         : in POLYLINE_INDEX;
    ERROR_INDICATOR               : out ERROR_NUMBER;
    TYPE_OF_LINE                  : out LINETYPE;
    WIDTH                         : out LINEWIDTH;
    LINE_COLOUR                   : out COLOUR_INDEX);
```

INQUIRE POLYMARKER FACILITIES Level 0a

```
procedure INQ_POLYMARKER_FACILITIES
   (TYPE_OF_WS              : in WS_TYPE;
    ERROR_INDICATOR         : out ERROR_NUMBER;
    LIST_OF_TYPES           : out MARKER_TYPES.LIST_OF;
    NUMBER_OF_SIZES         : out NATURAL;
    NOMINAL_SIZE            : out DC.MAGNITUDE;
    RANGE_OF_SIZES          : out DC.RANGE_OF_MAGNITUDES;
    NUMBER_OF_INDICES       : out NATURAL);
```

INQUIRE PREDEFINED POLYMARKER REPRESENTATION Level 0a

```
procedure INQ_PREDEFINED_POLYMARKER_REPRESENTATION
   (TYPE_OF_WS              : in WS_TYPE;
    INDEX                   : in POLYMARKER_INDEX;
    ERROR_INDICATOR         : out ERROR_NUMBER;
    TYPE_OF_MARKER          : out MARKER_TYPE;
    SIZE                    : out MARKER_SIZE;
    MARKER_COLOUR           : out COLOUR_INDEX);
```

INQUIRE TEXT FACILITIES Level 0a

```
procedure INQ_TEXT_FACILITIES
   (TYPE_OF_WS                  : in WS_TYPE;
    ERROR_INDICATOR             : out ERROR_NUMBER;
    LIST_OF_FONT_PRECISION_PAIRS : out TEXT_FONT_PRECISIONS.LIST_OF;
    NUMBER_OF_HEIGHTS           : out NATURAL;
    RANGE_OF_HEIGHTS            : out DC.RANGE_OF_MAGNITUDES;
    NUMBER_OF_EXPANSIONS        : out NATURAL;
    EXPANSION_RANGE             : out RANGE_OF_EXPANSIONS;
    NUMBER_OF_INDICES           : out NATURAL);
```

INQUIRE PREDEFINED TEXT REPRESENTATION Level 0a

```
procedure INQ_PREDEFINED_TEXT_REPRESENTATION
   (TYPE_OF_WS              : in WS_TYPE;
    INDEX                   : in TEXT_INDEX;
    ERROR_INDICATOR         : out ERROR_NUMBER;
    FONT_PRECISION          : out TEXT_FONT_PRECISION;
    EXPANSION               : out CHAR_EXPANSION;
    SPACING                 : out CHAR_SPACING;
    TEXT_COLOUR             : out COLOUR_INDEX);
```

INQUIRE FILL AREA FACILITIES Level 0a

```
procedure INQ_FILL_AREA_FACILITIES
   (TYPE_OF_WS                  : in WS_TYPE;
    ERROR_INDICATOR             : out ERROR_NUMBER;
    LIST_OF_INTERIOR_STYLES     : out INTERIOR_STYLES.LIST_OF;
    LIST_OF_HATCH_STYLES        : out HATCH_STYLES.LIST_OF;
    NUMBER_OF_INDICES           : out NATURAL);
```

INQUIRE PREDEFINED FILL AREA REPRESENTATION Level 0a

```
procedure INQ_PREDEFINED_FILL_AREA_REPRESENTATION
   (TYPE_OF_WS                  : in WS_TYPE;
    INDEX                       : in FILL_AREA_INDEX;
    ERROR_INDICATOR             : out ERROR_NUMBER;
    INTERIOR                    : out INTERIOR_STYLE;
    STYLE                       : out STYLE_INDEX;
    FILL_AREA_COLOUR            : out COLOUR_INDEX);
```

INQUIRE PATTERN FACILITIES Level 0a

```
procedure INQ_PATTERN_FACILITIES
   (TYPE_OF_WS                  : in WS_TYPE;
    ERROR_INDICATOR             : out ERROR_NUMBER;
    NUMBER_OF_INDICES           : out NATURAL);
```

INQUIRE PREDEFINED PATTERN REPRESENTATION Level 0a

```
procedure INQ_PREDEFINED_PATTERN_REPRESENTATION
   (TYPE_OF_WS                  : in WS_TYPE;
    INDEX                       : in PATTERN_INDEX;
    ERROR_INDICATOR             : out ERROR_NUMBER;
    PATTERN                     : out VARIABLE_COLOUR_MATRIX);
```

INQUIRE COLOUR FACILITIES Level 0a

```
procedure INQ_COLOUR_FACILITIES
   (TYPE_OF_WS                  : in WS_TYPE;
    ERROR_INDICATOR             : out ERROR_NUMBER;
    NUMBER_OF_COLOURS           : out NATURAL;
    AVAILABLE_COLOUR            : out COLOUR_AVAILABLE;
    NUMBER_OF_COLOUR_INDICES    : out NATURAL);
```

INQUIRE PREDEFINED COLOUR REPRESENTATION Level 0a

```
procedure INQ_PREDEFINED_COLOUR_REPRESENTATION
   (TYPE_OF_WS              : in WS_TYPE;
    INDEX                   : in COLOUR_INDEX;
    ERROR_INDICATOR         : out ERROR_NUMBER;
    RGB_COLOUR              : out COLOUR_REPRESENTATION);
```

INQUIRE LIST OF AVAILABLE GENERALIZED DRAWING PRIMITIVES Level 0a

```
procedure INQ_LIST_OF_AVAILABLE_GDP
   (TYPE_OF_WS              : in WS_TYPE;
    ERROR_INDICATOR         : out ERROR_NUMBER;
    LIST_OF_GDP             : out GDP_IDS.LIST_OF);
```

INQUIRE GENERALIZED DRAWING PRIMITIVE Level 0a

```
procedure INQ_GDP
   (TYPE_OF_WS                  : in WS_TYPE;
    GDP                         : in GDP_ID;
    ERROR_INDICATOR             : out ERROR_NUMBER;
    LIST_OF_ATTRIBUTES_USED     : out ATTRIBUTES_USED.LIST_OF);
```

INQUIRE MAXIMUM LENGTH OF WORKSTATION STATE TABLES Level 0a

```
procedure INQ_MAX_LENGTH_OF_WS_STATE_TABLES
   (TYPE_OF_WS                  : in WS_TYPE;
    ERROR_INDICATOR             : out ERROR_NUMBER;
    MAX_POLYLINE_ENTRIES        : out NATURAL;
    MAX_POLYMARKER_ENTRIES      : out NATURAL;
    MAX_TEXT_ENTRIES            : out NATURAL;
    MAX_FILL_AREA_ENTRIES       : out NATURAL;
    MAX_PATTERN_INDICES         : out NATURAL;
    MAX_COLOUR_INDICES          : out NATURAL);
```

INQUIRE NUMBER OF SEGMENT PRIORITIES SUPPORTED Level 1a

```
procedure INQ_NUMBER_OF_SEGMENT_PRIORITIES_SUPPORTED
   (TYPE_OF_WS                  : in WS_TYPE;
    ERROR_INDICATOR             : out ERROR_NUMBER;
    NUMBER_OF_PRIORITIES        : out NATURAL);
```

INQUIRE DYNAMIC MODIFICATION OF SEGMENT ATTRIBUTES Level 1a

```
procedure INQ_DYNAMIC_MODIFICATION_OF_SEGMENT_ATTRIBUTES
   (TYPE_OF_WS                : in WS_TYPE;
    ERROR_INDICATOR           : out ERROR_NUMBER;
    TRANSFORMATION            : out DYNAMIC_MODIFICATION;
    VISIBLE_TO_INVISIBLE      : out DYNAMIC_MODIFICATION;
    INVISIBLE_TO_VISIBLE      : out DYNAMIC_MODIFICATION;
    HIGHLIGHTING              : out DYNAMIC_MODIFICATION;
    PRIORITY                  : out DYNAMIC_MODIFICATION;
    ADDING_PRIMITIVES         : out DYNAMIC_MODIFICATION;
    DELETION_VISIBLE          : out DYNAMIC_MODIFICATION);
```

INQUIRE NUMBER OF AVAILABLE LOGICAL INPUT DEVICES Level 0b

```
procedure INQ_NUMBER_OF_AVAILABLE_LOGICAL_INPUT_DEVICES
   (TYPE_OF_WS                : in WS_TYPE;
    ERROR_INDICATOR           : out ERROR_NUMBER;
    LOCATOR                   : out NATURAL;
    STROKE                    : out NATURAL;
    VALUATOR                  : out NATURAL;
    CHOICE                    : out NATURAL;
    PICK                      : out NATURAL;
    STRING                    : out NATURAL);
```

INQUIRE DEFAULT LOCATOR DEVICE DATA Level 0b

```
procedure INQ_DEFAULT_LOCATOR_DEVICE_DATA
   (TYPE_OF_WS                : in WS_TYPE;
    DEVICE                    : in LOCATOR_DEVICE_NUMBER;
    ERROR_INDICATOR           : out ERROR_NUMBER;
    INITIAL_POSITION          : out WC.POINT;
    LIST_OF_PROMPT_ECHO_TYPES : out LOCATOR_PROMPT_ECHO_TYPES.LIST_OF;
    ECHO_AREA                 : out DC.RECTANGLE_LIMITS;
    DATA_RECORD               : out LOCATOR_DATA_RECORD);
```

INQUIRE DEFAULT STROKE DEVICE DATA Level 0b

```
procedure INQ_DEFAULT_STROKE_DEVICE_DATA
   (TYPE_OF_WS                : in WS_TYPE;
    DEVICE                    : in STROKE_DEVICE_NUMBER;
    ERROR_INDICATOR           : out ERROR_NUMBER;
    MAX_BUFFER_SIZE           : out NATURAL;
    LIST_OF_PROMPT_ECHO_TYPES : out STROKE_PROMPT_ECHO_TYPES.LIST_OF;
    ECHO_AREA                 : out DC.RECTANGLE_LIMITS;
    DATA_RECORD               : out STROKE_DATA_RECORD);
```

INQUIRE DEFAULT VALUATOR DEVICE DATA Level 0b

```
procedure INQ_DEFAULT_VALUATOR_DEVICE_DATA
   (TYPE_OF_WS                    : in WS_TYPE;
    DEVICE                        : in VALUATOR_DEVICE_NUMBER;
    ERROR_INDICATOR               : out ERROR_NUMBER;
    INITIAL_VALUE                 : out VALUATOR_INPUT_VALUE;
    LIST_OF_PROMPT_ECHO_TYPES     : out VALUATOR_PROMPT_ECHO_TYPES.LIST_OF;
    ECHO_AREA                     : out DC.RECTANGLE_LIMITS;
    DATA_RECORD                   : out VALUATOR_DATA_RECORD);
```

INQUIRE DEFAULT CHOICE DEVICE DATA Level 0b

```
procedure INQ_DEFAULT_CHOICE_DEVICE_DATA
   (TYPE_OF_WS                    : in WS_TYPE;
    DEVICE                        : in CHOICE_DEVICE_NUMBER;
    ERROR_INDICATOR               : out ERROR_NUMBER;
    MAX_CHOICES                   : out CHOICE_VALUE;
    LIST_OF_PROMPT_ECHO_TYPES     : out CHOICE_PROMPT_ECHO_TYPES.LIST_OF;
    ECHO_AREA                     : out DC.RECTANGLE_LIMITS;
    DATA_RECORD                   : out CHOICE_DATA_RECORD);
```

INQUIRE DEFAULT PICK DEVICE DATA Level 1b

```
procedure INQ_DEFAULT_PICK_DEVICE_DATA
   (TYPE_OF_WS                    : in WS_TYPE;
    DEVICE                        : in PICK_DEVICE_NUMBER;
    ERROR_INDICATOR               : out ERROR_NUMBER;
    LIST_OF_PROMPT_ECHO_TYPES     : out PICK_PROMPT_ECHO_TYPES.LIST_OF;
    ECHO_AREA                     : out DC.RECTANGLE_LIMITS;
    DATA_RECORD                   : out PICK_DATA_RECORD);
```

INQUIRE DEFAULT STRING DEVICE DATA Level 0b

```
procedure INQ_DEFAULT_STRING_DEVICE_DATA
   (TYPE_OF_WS                    : WS_TYPE;
    DEVICE                        : in STRING_DEVICE_NUMBER;
    ERROR_INDICATOR               : out ERROR_NUMBER;
    MAX_STRING_BUFFER_SIZE        : out NATURAL;
    LIST_OF_PROMPT_ECHO_TYPES     : out STRING_PROMPT_ECHO_TYPES.LIST_OF;
    ECHO_AREA                     : out DC.RECTANGLE_LIMITS;
    DATA_RECORD                   : out STRING_DATA_RECORD);
```

INQUIRE SET OF ASSOCIATED WORKSTATIONS Level 1a

```
procedure INQ_SET_OF_ASSOCIATED_WS
   (SEGMENT                    : in SEGMENT_NAME;
    ERROR_INDICATOR            : out ERROR_NUMBER;
    LIST_OF_WS                 : out WS_IDS.LIST_OF);
```

INQUIRE SEGMENT ATTRIBUTES Level 1a

```
procedure INQ_SEGMENT_ATTRIBUTES
   (SEGMENT                    : in SEGMENT_NAME;
    ERROR_INDICATOR            : out ERROR_NUMBER;
    TRANSFORMATION             : out TRANSFORMATION_MATRIX;
    VISIBILITY                 : out SEGMENT_VISIBILITY;
    HIGHLIGHTING               : out SEGMENT_HIGHLIGHTING;
    PRIORITY                   : out SEGMENT_PRIORITY;
    DETECTABILITY              : out SEGMENT_DETECTABILITY);
```

INQUIRE PIXEL ARRAY DIMENSIONS Level 0a

```
procedure INQ_PIXEL_ARRAY_DIMENSIONS
   (WS                         : in WS_ID;
    CORNER_1_1                 : in WC.POINT;
    CORNER_DX_DY               : in WC.POINT;
    ERROR_INDICATOR            : out ERROR_NUMBER;
    DIMENSIONS                 : out RASTER_UNIT_SIZE);
```

INQUIRE PIXEL ARRAY Level 0a

```
procedure INQ_PIXEL_ARRAY
   (WS                         : in WS_ID;
    CORNER                     : in WC.POINT;
    DX                         : in RASTER_UNITS;
    DY                         : in RASTER_UNITS;
    ERROR_INDICATOR            : out ERROR_NUMBER;
    INVALID_VALUES             : out INVALID_VALUES_INDICATOR;
    PIXEL_ARRAY                : out VARIABLE_PIXEL_COLOUR_MATRIX);
```

INQUIRE PIXEL Level 0a

```
procedure INQ_PIXEL
   (WS                         : in WS_ID;
    POINT                      : in WC.POINT;
    ERROR_INDICATOR            : out ERROR_NUMBER;
    PIXEL_COLOUR               : out PIXEL_COLOUR_INDEX);
```

INQUIRE INPUT QUEUE OVERFLOW Level 0c

```
procedure INQ_INPUT_QUEUE_OVERFLOW
    (ERROR_INDICATOR        : out ERROR_NUMBER;
     WS                     : out WS_ID;
     CLASS                  : out INPUT_QUEUE_CLASS;
     DEVICE                 : out EVENT_OVERFLOW_DEVICE_NUMBER);
```

EVALUATE TRANSFORMATION MATRIX Level 1a

```
procedure EVALUATE_TRANSFORMATION_MATRIX
    (FIXED_POINT            : in WC.POINT;
     SHIFT_VECTOR           : in WC.VECTOR;
     ROTATION_ANGLE         : in RADIANS;
     SCALE_FACTORS          : in TRANSFORMATION_FACTOR;
     TRANSFORMATION         : out TRANSFORMATION_MATRIX);

procedure EVALUATE_TRANSFORMATION_MATRIX
    (FIXED_POINT            : in NDC.POINT;
     SHIFT_VECTOR           : in NDC.VECTOR;
     ROTATION_ANGLE         : in RADIANS;
     SCALE_FACTORS          : in TRANSFORMATION_FACTOR;
     TRANSFORMATION         : out TRANSFORMATION_MATRIX);
```

ACCUMULATE TRANSFORMATION MATRIX Level 1a

```
procedure ACCUMULATE_TRANSFORMATION_MATRIX
    (SOURCE_TRANSFORMATION  : in TRANSFORMATION_MATRIX;
     FIXED_POINT            : in WC.POINT;
     SHIFT_VECTOR           : in WC.VECTOR;
     ROTATION_ANGLE         : in RADIANS;
     SCALE_FACTORS          : in TRANSFORMATION_FACTOR;
     RESULT_TRANSFORMATION  : out TRANSFORMATION_MATRIX);

procedure ACCUMULATE_TRANSFORMATION_MATRIX
    (SOURCE_TRANSFORMATION  : in TRANSFORMATION_MATRIX;
     FIXED_POINT            : in NDC.POINT;
     SHIFT_VECTOR           : in NDC.VECTOR;
     ROTATION_ANGLE         : in RADIANS;
     SCALE_FACTORS          : in TRANSFORMATION_FACTOR;
     RESULT_TRANSFORMATION  : out TRANSFORMATION_MATRIX);
```

EMERGENCY CLOSE GKS Level 0a

```
procedure EMERGENCY_CLOSE_GKS;
```

ERROR HANDLING Level 0a

```
procedure ERROR_HANDLING
   (ERROR_INDICATOR        : in ERROR_NUMBER;
    GKS_FUNCTION           : in STRING;
    ERROR_FILE             : in STRING    := DEFAULT_ERROR_FILE);
```

ERROR LOGGING Level 0a

```
procedure ERROR_LOGGING
   (ERROR_INDICATOR        : in ERROR_NUMBER;
    GKS_FUNCTION           : in STRING;
    ERROR_FILE             : in STRING    := DEFAULT_ERROR_FILE);
```

5.2 Additional functions

5.2.1 Subprograms for Manipulating Input Data Records

The procedures and functions defined in this section are those that are necessary for constructing and inquiring the input data records, declared as private types in this binding for each of the six classes of input devices defined by the GKS specification -- the Locator, Stroke, Valuator, Choice, Pick, and String logical devices. The procedures listed here are used to construct the data records for each of the registered prompt and echo types of a device class to be used for initialising a particular input device. Assorted functions are also provided so that an application of GKS/Ada may examine the parts of the data record which are defined by GKS. Any implementation specific information in the data records is kept private and unavailable. The exception GKS_ERROR (error class LANGUAGE_BINDING_ERROR) is raised if any of the below procedures are used incorrectly. That is, if an illegal prompt and echo type is used for a build procedure, then error number 2500 is logged onto the error file.

These subprograms are required at level 0b.

To implement implementation-dependent and registered items, an implementation may provide additional overloaded versions of the BUILD procedures in this section, and additional functions for extracting information from the private data records.

```
-- Locator Data Record Operations

procedure BUILD_LOCATOR_DATA_RECORD
        (PROMPT_ECHO_TYPE          : in LOCATOR_PROMPT_ECHO_TYPE;
         DATA_RECORD               : out LOCATOR_DATA_RECORD);

-- Constructs and returns a locator data record.
```

```
-- Stroke Data Record Operations.

procedure BUILD_STROKE_DATA_RECORD
          (PROMPT_ECHO_TYPE          : in STROKE_PROMPT_ECHO_TYPE;
           BUFFER_SIZE               : in POSITIVE;
           DATA_RECORD               : out STROKE_DATA_RECORD);

-- Constructs and returns a stroke data record.
-------------------------------------------------------------------------------------------------------------------

function BUFFER_SIZE (DATA_RECORD : in STROKE_DATA_RECORD)
          return POSITIVE;

-- Returns the size of the input stroke buffer stored in the data record.

-- Valuator Data Record Operations.

procedure BUILD_VALUATOR_DATA_RECORD
          (PROMPT_ECHO_TYPE          : in VALUATOR_PROMPT_ECHO_TYPE;
           LOW_VALUE                 : in VALUATOR_INPUT_VALUE;
           HIGH_VALUE                : in VALUATOR_INPUT_VALUE;
           DATA_RECORD               : out VALUATOR_DATA_RECORD);

-- Constructs and returns a valuator data record..

function HIGH_VALUE (DATA_RECORD      : in VALUATOR_DATA_RECORD)
          return VALUATOR_INPUT_VALUE;

-- Returns the high value for the valuator stored in the valuator data record.

function LOW_VALUE (DATA_RECORD       : in VALUATOR_DATA_RECORD)
          return VALUATOR_INPUT_VALUE;

-- Returns the low value for the valuator stored in the valuator data record.


-- Choice Data Record Operations.

procedure BUILD_CHOICE_DATA_RECORD
          (PROMPT_ECHO_TYPE          : in CHOICE_PROMPT_ECHO_TYPE;
           DATA_RECORD               : out CHOICE_DATA_RECORD);

-- Constructs and returns a choice data record.


-- Pick Data Record Operation.

procedure BUILD_PICK_DATA_RECORD
          (PROMPT_ECHO_TYPE          : in PICK_PROMPT_ECHO_TYPE;
           DATA_RECORD               : out PICK_DATA_RECORD);

-- Constructs and returns a pick data record.
```

```
-- String Data Record Operations.

procedure BUILD_STRING_DATA_RECORD
          (PROMPT_ECHO_TYPE                : in STRING_PROMPT_ECHO_TYPE;
           INPUT_BUFFER_SIZE               : in NATURAL;
           INITIAL_CURSOR_POSITION         : in NATURAL;
           DATA_RECORD                     : out STRING_DATA_RECORD);

-- Constructs and returns a string data record.

function INPUT_BUFFER_SIZE (DATA_RECORD          : in STRING_DATA_RECORD)
          return NATURAL;

-- Returns the size of the buffer used for storing string input stored in the string data record.

function INITIAL_CURSOR_POSITION
          (DATA_RECORD    : in STRING_DATA_RECORD) return NATURAL;

-- Returns the initial cursor position for string input stored in the string data record.
```

5.2.2 GKS Generic coordinate system package

The generic package declared in this section is the specification of a generic Cartesian Coordinate System for GKS. This package is instantiated three times in the GKS_TYPES package for World Coordinates, Normalized Device Coordinates, and Device Coordinates. The package defines the representation of a POINT, a POINT_ARRAY, a POINT_LIST, a VECTOR, and RECTANGLE_LIMITS for a coordinate system. Also defined is a MAGNITUDE type for measuring lengths within a coordinate space. The type SIZE measures lengths parallel to both axes, and the RANGE_OF_MAGNITUDES type specifies two lengths within a coordinate system, a minimum and maximum for values such as the range of Character Heights available on a device. This generic is included in the GKS types package.

```
generic
        type COORDINATE_COMPONENT_TYPE is digits <>;

package GKS_COORDINATE_SYSTEM is

        type POINT is
          record
            X : COORDINATE_COMPONENT_TYPE;
            Y : COORDINATE_COMPONENT_TYPE;
          end record;

        type POINT_ARRAY is array (POSITIVE range <>) of POINT;

        type POINT_LIST (LENGTH : SMALL_NATURAL := 0) is
          record
            POINTS : POINT_ARRAY (1..LENGTH);
          end record;

        type VECTOR is new POINT;

        type RECTANGLE_LIMITS is
          record
            XMIN          : COORDINATE_COMPONENT_TYPE;
            XMAX          : COORDINATE_COMPONENT_TYPE;
            YMIN          : COORDINATE_COMPONENT_TYPE;
            YMAX          : COORDINATE_COMPONENT_TYPE;
      end record;

        type MAGNITUDE_BASE_TYPE is digits PRECISION;

        subtype MAGNITUDE is MAGNITUDE_BASE_TYPE range
          COORDINATE_COMPONENT_TYPE'SAFE_SMALL..
          COORDINATE_COMPONENT_TYPE'SAFE_LARGE;

        type SIZE is
          record
            XAXIS  : MAGNITUDE;
            YAXIS  : MAGNITUDE;
          end record;

        type RANGE_OF_MAGNITUDES is
          record
            MIN: MAGNITUDE;
            MAX: MAGNITUDE;
          end record;

end GKS_COORDINATE_SYSTEM;
```

5.2.3 GKS Generic list utility package

The generic package GKS_LIST_UTILITIES is instantiated several times in the GKS_TYPES package to define several LIST_OF types and their manipulation subprograms. Each LIST_OF type contains different element type values.

The LIST_OF type is declared as a private type in GKS_LIST_UTILITIES to restrict the operations on the LIST_OF type that are available to outside program units. The LIST_OF private type declaration includes a discriminant part that defines the current size of the lists. LIST_OF objects are declared as unconstrained objects (by using the default discriminant value) to allow dynamic modification of the list size.

A LIST_OF object is a sequence of element type values. Each element type value is associated with an index. Index values begin at one and increase in steps of one.

The size of a LIST_OF object is the number of element type values stored within it. A single element type value may be stored more than once within a LIST_OF object. A LIST_OF object may be empty. The size of an empty LIST_OF object is zero. The maximum size of a LIST_OF object is given by the MAX_LIST_SIZE generic parameter. If this parameter is not specified in the instantiation, an implementation dependent default value is used.

```
-- The LIST_OF manipulation subprograms are:

function NULL_LIST return LIST_OF;

-- This function returns an empty LIST_OF object. This list is intended primarily for use by GKS implementors.

procedure ADD_TO_LIST
          (ELEMENT                  : in ELEMENT_TYPE;
           LIST                     : in out LIST_OF);

-- This procedure stores the element parameter value in the list parameter object, and increases the size of the list by
-- one. An index value equal to the incremented list size is associated with the stored element value. The
-- ADD_TO_LIST procedure will generate GKS_ERROR 2502 if it is called when the list parameter has a size
-- equal to the maximum size. If desired, the user can ensure duplicate values are not stored. This is accomplished
-- by calling ADD_TO_LIST with a particular element value only if the function IS_IN_LIST returns FALSE for
-- That element value.

procedure DELETE_FROM_LIST
          (ELEMENT                  : in ELEMENT_TYPE;
           LIST                     : in out LIST_OF);

-- If the list parameter object does not contain the element parameter value, this procedure has no effect. Otherwise,
-- the first occurrence of the element value is deleted. The size of the list object is decreased by one, and the
-- indices associated with the remaining element values are adjusted so that the indices begin at
-- one and increment in steps of one. If desired, the user can delete all occurrences of an element value. This is
-- accomplished by calling DELETE_FROM_LIST repeatedly with a particular element value while the function
-- IS_IN_LIST returns TRUE for that element value.

function SIZE_OF_LIST (LIST         : in LIST_OF) return NATURAL;

-- This function returns the number of element type values stored in the list object.

function IS_IN_LIST
          (ELEMENT                  : in ELEMENT_TYPE;
           LIST                     : in LIST_OF) return BOOLEAN;

-- This function returns the value TRUE if the element parameter value is in the list object, otherwise it
-- returns FALSE.
```

```
function LIST_ELEMENT
            (INDEX          : in POSITIVE;
             LIST           : in LIST_OF) return ELEMENT_TYPE;

-- This function returns the element value in the list object that has an associated index value equal to the
-- index parameter. The GKS_ERROR 2502 is generated if the index parameter exceeds the current
-- size of the list parameter object.

function LIST (VALUES           : in LIST_VALUES) return LIST_OF;

-- This function returns a valid LIST_OF object. If the VALUES parameter is a null array, an empty
-- LIST_OF object is returned. If the values parameter is not null, this function returns a LIST_OF object
-- containing all the values in the VALUES parameter. The GKS_ERROR 2502 is generated if the number
-- of element values exceeds the maximum size of the LIST_OF object.

-- The generic package specification is:

generic

        type ELEMENT_TYPE is private;
        MAX_LIST_SIZE : POSITIVE := implementation_defined;

package GKS_LIST_UTILITIES is

        subtype LIST_SIZE is NATURAL range 0 .. MAX_LIST_SIZE;
        type LIST_OF  (SIZE  :  LIST_SIZE  := 0) is private;
        type LIST_VALUES is array (POSITIVE range <>) of ELEMENT_TYPE;

function NULL_LIST  return LIST_OF;

function SIZE_OF_LIST (LIST             : in LIST_OF) return NATURAL;

function IS_IN_LIST (ELEMENT            : in ELEMENT_TYPE
                     LIST               : in LIST_OF)     return BOOLEAN;

function LIST_ELEMENT (INDEX            : in POSITIVE;
                       LIST             : in LIST_OF)     return ELEMENT_TYPE;

function LIST (VALUES  : in LIST_VALUES) return LIST_OF;

procedure ADD_TO_LIST (ELEMENT          : in ELEMENT_TYPE;
                       LIST             : in out LIST_OF);

procedure DELETE_FROM_LIST (ELEMENT     : in ELEMENT_TYPE;
                            LIST        : in out LIST_OF);

private

-- The declaration of the LIST_OF type is implementation dependent. However the operations implicitly declared
-- by the LIST_OF declaration, including both assignment and the predefined comparison for equality and inequality,
-- must produce the correct results. This requirement precludes the use of access types for the implementation of the
-- LIST_OF type. The recommended implementation is given below:
--
--      type LIST_OF (SIZE:  LIST_SIZE  :=  0) is
--       record
--         ELEMENTS  :  LIST_VALUES (1 .. SIZE);
--       end_record;
--
```

```
-- Note that declaring unconstrained LIST_OF objects by using the default discriminant value allows dynamic
-- modification of the size of the element array.

end GKS_LIST_UTILITIES;
```

5.2.4 Metafile function utilities

The Metafile item data records are complex, over 55 different formats for these records have been recommended. The application programmer is also allowed to define new formats. The length of these records is variable. The number of data elements in some of these records is also variable. Item data records may contain lists of points, character strings, arrays of colour indices, and GDP and ESC data. Record length depends on the number of data elements. GKS defines that the format is implementation defined.

The item data record type should be private to allow direct manipulation of the record contents in order to have them efficiently processed.

The application programmer must be able to write non-graphical data into the metafile. This can be provided by allowing character strings to be output. Numeric data must be converted to a string by the application programmer prior to calling BUILD_NEW_GKSM_DATA_RECORD. A function is provided as a means to convert item data records into strings.

BUILD NEW GKSM DATA RECORD

```
procedure BUILD_NEW_GKSM_DATA_RECORD
        (TYPE_OF_ITEM                   : in GKSM_ITEM_TYPE;
         ITEM_DATA                      : in STRING;
         ITEM                           : out GKSM_DATA_RECORD);
```

ITEM DATA RECORD STRING

```
function ITEM_DATA_RECORD_STRING
        (ITEM                   : in GKSM_DATA_RECORD)      return STRING;
```

5.3 Conformal Variants

Since no subsets or supersets of the Ada language are allowed, GKS/Ada has no conformal variants. Furthermore, this binding does not require the use of any Ada language feature for which support of that feature is implementation-dependent.

Annex A
Compiled GKS Specification

(This annex does not form an integral part of this standard but provides additional information.)

```
with GKS_LIST_UTILITIES;

package GKS_TYPES is

-- This package contains all the data type definitions used to define the Ada binding to GKS.
-- This compilation was done on a MicroVax II computer using the Vax Ada compiler, Version
-- T1.4-32.  The values for implementation-dependent types or subtypes were chosen to operate in
-- a 32-bit, minicomputer environment with virtual memory.  These values would need to be
-- changed for microcomputer or fixed memory-sized machines.

-- The following constants are implementation-dependent and define maximum implementation
-- limits for GKS/Ada types.

        PRECISION                       : constant      := 6;
        SMALL_NATURAL_MAX               : constant      := 500;
        STRING_SMALL_NATURAL_MAX        : constant      := 100;
        CHOICE_SMALL_NATURAL_MAX        : contstant     := 5;


        subtype SMALL_NATURAL is NATURAL range 0..SMALL_NATURAL_MAX;

        -- This is an implementation-dependent subtype which allows for unconstrained record
        -- objects for various record types defined below without causing the exception
        -- STORAGE_ERROR to be raised.

        subtype STRING_SMALL_NATURAL is NATURAL
                range 0..STRING_SMALL_NATURAL_MAX;

        -- This is an implementation-dependent subtype declaration that allows for unconstrained
        -- record objects for various string record types defined below without causing the
        -- exception STORAGE_ERROR to be raised.

        subtype CHOICE_SMALL_NATURAL is NATURAL
                range 0..CHOICE_SMALL_NATURAL_MAX;

        -- This is an implementation-defined subtype declaration which allows for unconstrained
        -- record objects for CHOICE_PROMPT_STRING_LIST type without causing the
        -- exception STORAGE_ERROR to be raised.

-- The  GKS Coordinate System

generic
        type COORDINATE_COMPONENT_TYPE is digits <>;

package GKS_COORDINATE_SYSTEM is

        type POINT is
                record
                        X : COORDINATE_COMPONENT_TYPE;
                        Y : COORDINATE_COMPONENT_TYPE;
                end record;

        type POINT_ARRAY is array (POSITIVE range <>) of POINT;
```

```
        type POINT_LIST (LENGTH : SMALL_NATURAL := 0) is
                record
                        POINTS : POINT_ARRAY (1..LENGTH);
                end record;

        type VECTOR is new POINT;

        type RECTANGLE_LIMITS is
                record
                        XMIN            : COORDINATE_COMPONENT_TYPE;
                        XMAX            : COORDINATE_COMPONENT_TYPE;
                        YMIN            : COORDINATE_COMPONENT_TYPE;
                        YMAX            : COORDINATE_COMPONENT_TYPE;
        end record;

        type MAGNITUDE_BASE_TYPE is digits PRECISION;

        subtype MAGNITUDE is MAGNITUDE_BASE_TYPE range
                COORDINATE_COMPONENT_TYPE'SAFE_SMALL..
                COORDINATE_COMPONENT_TYPE'SAFE_LARGE;

        type SIZE is
                record
                        XAXIS           : MAGNITUDE;
                        YAXIS           : MAGNITUDE;
                end record;

        type RANGE_OF_MAGNITUDES is
                record
                        MIN     : MAGNITUDE;
                        MAX     : MAGNITUDE;
                end record;

end GKS_COORDINATE_SYSTEM;

        -- ASF                                                  Level 0a

        type ASF is (BUNDLED, INDIVIDUAL);

                -- This type defines an aspect source flag whose value indicates whether
                -- an aspect of a primitive should be set from a bundle table or from an
                -- individual attribute.
```

```
-- ASF_LIST                                                    Level 0a
type ASF_LIST is
        record
            TYPE_OF_LINE_ASF            : ASF;
            WIDTH_ASF                   : ASF;
            LINE_COLOUR_ASF             : ASF;
            TYPE_OF_MARKER_ASF          : ASF;
            SIZE_ASF                    : ASF;
            MARKER_COLOUR_ASF           : ASF;
            FONT_PRECISION_ASF          : ASF;
            EXPANSION_ASF               : ASF;
            SPACING_ASF                 : ASF;
            TEXT_COLOUR_ASF             : ASF;
            INTERIOR_ASF                : ASF;
            STYLE_ASF                   : ASF;
            FILL_AREA_COLOUR_ASF        : ASF;
        end record;

-- A record containing all of the aspect source flags, with components indicating the
-- specific flag.


--ATTRIBUTES_USED_TYPE                                         Level 0a

type ATTRIBUTES_USED_TYPE is
        (POLYLINE_ATTRIBUTES,
         POLYMARKER_ATTRIBUTES,
         TEXT_ATTRIBUTES,
         FILL_AREA_ATTRIBUTES);

-- The types of attributes which may be used in generating output for a GDP and in
-- generating prompt and echo information for certain prompt and echo types of
-- certain classes of input devices.

-- ATTRIBUTES_USED                                             Level 0a

package ATTRIBUTES_USED is
        new GKS_LIST_UTILITIES (ATTRIBUTES_USED_TYPE);

-- Provides for a list of the attributes used.

-- SCALE_FACTOR                                                Level 0a

package SCALE_FACTOR_TYPE is

-- This package is used to encapsulate the derived type SCALE_FACTOR since it is used
-- as the parent of several other derived types.  In Ada, if the parent of a derived type is
-- itself a derived type, then this parent type cannot be declared immediately in the visible
-- part of the same package.

        type SCALE_FACTOR is digits PRECISION;
-- The type used for unitless scaling factors.
end  SCALE_FACTOR_TYPE;
use  SCALE_FACTOR_TYPE;
```

```
-- CHAR_EXPANSION                                          Level 0a

type CHAR_EXPANSION is new SCALE_FACTOR range
     SCALE_FACTOR'SAFE_SMALL..SCALE_FACTOR'LAST;

-- Defines a character expansion factor.  Factors are unitless, and must be greater than
-- zero.


-- CHAR_SPACING                                            Level 0a

type CHAR_SPACING is new SCALE_FACTOR;

-- Defines a character spacing factor.  The factors are unitless.  A positive value indicates
-- the amount of extra space between characters in a text string, and a negative value
-- indicates the amount of overlap between character boxes in a text string.

-- DEVICE_NUMBER                                           Level 0b

package DEVICE_NUMBER_TYPE is
        type DEVICE_NUMBER is new POSITIVE;
        -- Logical input devices are referenced as device numbers.
end DEVICE_NUMBER_TYPE;
use DEVICE_NUMBER_TYPE;

-- CHOICE_DEVICE_NUMBER                                    Level 0b

type CHOICE_DEVICE_NUMBER is new DEVICE_NUMBER;

-- Provides for choice device identifiers.

-- LOCATOR_DEVICE_NUMBER                                   Level 0b

type LOCATOR_DEVICE_NUMBER is new DEVICE_NUMBER;

-- Provides for locator device identifiers.

-- PICK_DEVICE_NUMBER                                      Level 1b

type PICK_DEVICE_NUMBER is new DEVICE_NUMBER;

-- Provides for pick devices.

-- STRING_DEVICE_NUMBER                                    Level 0b

type STRING_DEVICE_NUMBER is new DEVICE_NUMBER;

-- Provides for string device number.

-- STROKE_DEVICE_NUMBER                                    Level 0b

type STROKE_DEVICE_NUMBER is new DEVICE_NUMBER;

-- Provides for stroke device numbers.
```

```
-- VALUATOR_DEVICE_NUMBER                                    Level 0b

type VALUATOR_DEVICE_NUMBER is new DEVICE_NUMBER;

-- Provides for valuator device identifiers.

-- CHOICE_PROMPT                                             Level 0b

type CHOICE_PROMPT is (OFF,ON);

-- Indicates for a choice prompt and echo type whether  a specified prompt is to
-- be displayed or not.


-- CHOICE_PROMPTS                                            Level 0b

package CHOICE_PROMPTS is
        new GKS_LIST_UTILITIES (CHOICE_PROMPT);

 -- Provides for lists of prompts.

-- CHOICE_PROMPT_ECHO_TYPE                                   Level 0b

type CHOICE_PROMPT_ECHO_TYPE is new INTEGER;

-- Defines the choice prompt and echo type.


-- CHOICE_PROMPT_ECHO_TYPES                                  Level 0b

package CHOICE_PROMPT_ECHO_TYPES is
        new GKS_LIST_UTILITIES (CHOICE_PROMPT_ECHO_TYPE);

-- Provides for lists of choice prompt and echo types.


-- CHOICE_PROMPT_STRING                                      Level 0b

type CHOICE_PROMPT_STRING (LENGTH : STRING_SMALL_NATURAL := 0) is
        record
           CONTENTS : STRING (1..LENGTH);
        end record;

-- Provides for a variable length prompt.  Objects of this type should be declared
-- unconstrained to allow for dynamic modification of the length.

-- CHOICE_PROMPT_STRING_ARRAY                                Level 0b

type CHOICE_PROMPT_STRING_ARRAY is array (POSITIVE range <>)
        of CHOICE_PROMPT_STRING;

-- Provides for an array of prompt strings.
```

```
-- CHOICE_PROMPT_STRING_LIST                                    Level 0b

type CHOICE_PROMPT_STRING_LIST(LENGTH:CHOICE_SMALL_NATURAL:= 0)
      is record
         LIST : CHOICE_PROMPT_STRING_ARRAY (1..LENGTH);
      end record;

-- Provides for lists of prompt strings.


-- CHOICE_REQUEST_STATUS                                        Level 0b

type CHOICE_REQUEST_STATUS is (OK, NOCHOICE, NONE);

-- Defines the status of a choice input operation for the request function.


-- CHOICE_STATUS                                                Level 0b

subtype CHOICE_STATUS is CHOICE_REQUEST_STATUS range OK..NOCHOICE;

-- Indicates if a choice was made by the operator for the sample, get, and inquiry
-- functions.

-- CHOICE_VALUE                                                 Level 0b

type CHOICE_VALUE is new POSITIVE;

-- Defines the choice values available on an implementation.

-- CLIPPING_INDICATOR                                           Level 0a

type CLIPPING_INDICATOR is (CLIP, NOCLIP);

-- Indicates whether or not clipping is to be performed.

-- COLOUR_AVAILABLE                                             Level 0a

type COLOUR_AVAILABLE is (COLOUR, MONOCHROME);

-- Indicates whether colour output is available on a workstation.

-- PIXEL_COLOUR_INDEX                                           Level 0a

type PIXEL_COLOUR_INDEX is new INTEGER range -1..INTEGER'LAST;

-- A type for the pixel colour where the value -1 represents an invalid colour index.

-- COLOUR_INDEX                                                 Level 0a

subtype COLOUR_INDEX is PIXEL_COLOUR_INDEX
          range  0..PIXEL_COLOUR_INDEX'LAST;

-- Indices into colour tables are of this type.
```

```
-- COLOUR_INDICES                                          Level 0a

package COLOUR_INDICES is new GKS_LIST_UTILITIES (COLOUR_INDEX);

-- Provides for a set of colour indices which are available on a particular workstation.

-- COLOUR_MATRIX                                           Level 0a

type COLOUR_MATRIX is array (POSITIVE range <>,POSITIVE range <>)
                                of COLOUR_INDEX;

-- Provides for matrices containing colour indices corresponding to a
-- cell array or pattern array.

-- INTENSITY                                               Level 0a

type INTENSITY is digits PRECISION range 0.0..1.0;

-- Defines the range of possible intensities of a colour.

-- COLOUR_REPRESENTATION                                   Level 0a

type COLOUR_REPRESENTATION is
      record
         RED   : INTENSITY;
         GREEN : INTENSITY;
         BLUE  : INTENSITY;
      end record;

-- Defines the representation of a colour as a combination of intensities in
-- an RGB colour system.

-- CONTROL_FLAG                                            Level 0a

type CONTROL_FLAG is (CONDITIONALLY, ALWAYS);

-- The control flag is used to indicate the conditions under which the display
-- surface should be cleared.

-- DC_TYPE                                                 Level 0a

type DC_TYPE is digits PRECISION;

-- The type of a coordinate in the Device Coordinate System.

-- DC                                                      Level 0a

package DC is new GKS_COORDINATE_SYSTEM (DC_TYPE);

-- Defines the Device Coordinate System.

-- DC_UNITS                                                Level 0a

type DC_UNITS is (METRES, OTHER);

-- Device coordinate units for a particular workstation should be in metres unless
-- the device is incapable of producing a precisely scaled image, and appropriate work-
-- station dependent units otherwise.
```

```
-- DEFERRAL_MODE                                          Level 0a

type DEFERRAL_MODE is (ASAP, BNIG, BNIL, ASTI);

-- Defines the four GKS deferral modes.


-- DISPLAY_CLASS                                          Level 0a

type DISPLAY_CLASS is (VECTOR_DISPLAY,
                       RASTER_DISPLAY,
                       OTHER_DISPLAY);

-- The classification of a workstation of category  OUTPUT or OUTIN.

-- DISPLAY_SURFACE_EMPTY                                  Level 0a

type DISPLAY_SURFACE_EMPTY is (EMPTY,  NOTEMPTY);

-- Indicates whether the display surface is empty.

-- DYNAMIC_MODIFICATION                                   Level 1a

type DYNAMIC_MODIFICATION is (IRG, IMM);

-- Indicates whether an update to the state list is performed immediately (IMM)
-- or requires implicit regeneration (IRG).

-- ECHO_SWITCH                                            Level 0b

type ECHO_SWITCH is (ECHO,  NOECHO);

-- Indicates whether or not echoing of the prompt is performed.

-- ERROR_NUMBER                                           Level 0a

type ERROR_NUMBER is new INTEGER;

-- Defines the type for error indicator values.

-- INPUT_CLASS                                            Level 0b

type INPUT_CLASS is     (NONE,
                         LOCATOR_INPUT,
                         STROKE_INPUT,
                         VALUATOR_INPUT,
                         CHOICE_INPUT,
                         PICK_INPUT
                         STRING_INPUT);
```

-- Defines the input device classifications for workstations of category INPUT or OUTIN.

```
-- EVENT_DEVICE_NUMBER                                          Level 0a

type EVENT_DEVICE_NUMBER (CLASS : INPUT_CLASS := NONE) is
     record
      case CLASS is
         when  NONE                => null;
         when LOCATOR_INPUT        => LOCATOR_EVENT_DEVICE
                                        : LOCATOR_ DEVICE_NUMBER;
         when STROKE_INPUT         => STROKE_EVENT_DEVICE
                                        : STROKE_DEVICE_NUMBER;
         when VALUATOR_INPUT       => VALUATOR_EVENT_DEVICE
                                        : VALUATOR_DEVICE_NUMBER;
         when CHOICE_INPUT         => CHOICE_EVENT_DEVICE
                                        : CHOICE_DEVICE_NUMBER;
         when PICK_INPUT           => PICK_EVENT_DEVICE
                                        : PICK_DEVICE_NUMBER;
         when STRING_INPUT         => STRING_EVENT_DEVICE
                                        : STRING_DEVICE_NUMBER;
      end case;
     end record;

-- Provides for returning any class of device number from the event queue.

-- INPUT_QUEUE_CLASS                                            Level 0a

subtype INPUT_QUEUE_CLASS is INPUT_CLASS range
          LOCATOR_INPUT .. STRING_INPUT;

-- Defines the input device classifications for situations in which the
-- NONE classification is impossible.

-- EVENT_OVERFLOW_DEVICE_NUMBER                                 Level 0a

type EVENT_OVERFLOW_DEVICE_NUMBER
     (CLASS : INPUT_QUEUE_CLASS := LOCATOR_INPUT) is
     record
      case CLASS is
        when LOCATOR_INPUT         => LOCATOR_EVENT_DEVICE
                                        : LOCATOR_ DEVICE_NUMBER;
        when STROKE_INPUT          => STROKE_EVENT_DEVICE
                                        : STROKE_DEVICE_NUMBER;
        when VALUATOR_INPUT        => VALUATOR_EVENT_DEVICE
                                        : VALUATOR_DEVICE_NUMBER;
        when CHOICE_INPUT          => CHOICE_EVENT_DEVICE
                                        : CHOICE_DEVICE_NUMBER;
        when PICK_INPUT            => PICK_EVENT_DEVICE
                                        : PICK_DEVICE_NUMBER;
        when STRING_INPUT          => STRING_EVENT_DEVICE
                                        : STRING_DEVICE_NUMBER;
      end case;
     end record;

-- FILL_AREA_INDEX                                              Level 0a

type FILL_AREA_INDEX is new POSITIVE;

-- Defines fill area bundle table indices.
```

```
-- INTERIOR_STYLE                                          Level 0a

type INTERIOR_STYLE is (HOLLOW, SOLID, PATTERN, HATCH);

-- Defines the fill area interior styles.

-- STYLE_INDEX                                             Level 0a

type STYLE_INDEX is new INTEGER;

-- A style index is either a HATCH_STYLE or a  PATTERN_STYLE.

-- FILL_AREA_INDICES                                       Level 0a

package FILL_AREA_INDICES is
            new GKS_LIST_UTILITIES (FILL_AREA_INDEX);

-- Provides for lists of fill area bundle table indices.

-- GDP_ID                                                  Level 0a

type GDP_ID is new INTEGER;

-- Selects among the kinds of Generalized Drawing Primitives.

-- GDP_IDS                                                 Level 0a

package GDP_IDS is new GKS_LIST_UTILITIES (GDP_ID);

-- Provides for lists of Generalized Drawing Primitive ID's.

-- GKS_LEVEL                                               Level 0a

type GKS_Level is (L0a, L0b, L0c, L1a, L1b, L1c, L2a, L2b, L2c);

-- The valid Levels of GKS.

-- GKSM_ITEM_TYPE                                          Level 0a

type GKSM_ITEM_TYPE is new NATURAL;

-- The type of an item contained in a GKSM metafile.

-- HATCH_STYLE                                             Level 0a

subtype HATCH_STYLE is STYLE_INDEX;

-- Defines the fill area hatch styles type.

-- HATCH_STYLES                                            Level 0a

package HATCH_STYLES is new GKS_LIST_UTILITIES (HATCH_STYLE);

-- Provides for lists of hatch styles.
```

```
-- HORIZONTAL_ALIGNMENT                                    Level 0a

type HORIZONTAL_ALIGNMENT is (NORMAL, LEFT, CENTRE, RIGHT);

-- The alignment of the text extent parallelogram with respect to the horizontal
-- positioning of the text.

-- IMPLEMENTATION_DEFINED_ERROR                            Level 0a

subtype IMPLEMENTATION_DEFINED_ERROR is ERROR_NUMBER
        range ERROR_NUMBER'FIRST .. -1;

-- Defines the range of ERROR_NUMBERS to indicate that an implementation
-- defined error has occurred.


-- INPUT_STATUS                                            Level 0b

type INPUT_STATUS is (OK, NONE);

-- Defines the status of a locator, stroke, valuator, or string operation.

-- INPUT_STRING                                            Level 0b

type INPUT_STRING (LENGTH : STRING_SMALL_NATURAL := 0) is
    record
       CONTENTS : STRING (1..LENGTH);
    end record;

-- Provides a variable length string.  Objects of this type should be declared
-- unconstrained to allow for dynamic modification of the length.

-- INTERIOR_STYLES                                         Level 0a

package INTERIOR_STYLES is
        new GKS_LIST_UTILITIES (INTERIOR_STYLE);

-- Provides for lists of interior styles.

-- INVALID_VALUES_INDICATOR                                Level 0a

type INVALID_VALUES_INDICATOR is (ABSENT, PRESENT);

-- Indicates whether the value -1 (i.e. "invalid") is absent from or present
-- in the PIXEL_ARRAY parameter returned by INQ_PIXEL_ARRAY.

-- LANGUAGE_BINDING_ERROR                                  Level 0a

subtype LANGUAGE_BINDING_ERROR is ERROR_NUMBER
        range 2500 .. 2999;

-- Defines the range of ERROR_NUMBERS to indicate that a language binding
-- error has occurred.
```

```
-- POLYLINE_INDEX                                              Level 0a

type POLYLINE_INDEX is new POSITIVE;

-- Defines the range of polyline indices.

-- LINETYPE                                                    Level 0a

type LINETYPE is new INTEGER;

-- Defines the types of line styles provided by GKS.

-- LINEWIDTH                                                   Level 0a

type LINEWIDTH is new SCALE_FACTOR range 0.0..SCALE_FACTOR'LAST;

-- The width of a line is indicated by a scale factor.

-- LINETYPES                                                   Level 0a

package LINETYPES is new GKS_LIST_UTILITIES (LINETYPE);

-- Provides for lists of line types.


-- LOCATOR_PROMPT_ECHO_TYPE                                    Level 0b

type LOCATOR_PROMPT_ECHO_TYPE is new INTEGER;

-- Defines the locator prompt and echo types supported by the implementation.

-- LOCATOR_PROMPT_ECHO_TYPES                                   Level 0b

package LOCATOR_PROMPT_ECHO_TYPES is
        new GKS_LIST_UTILITIES (LOCATOR_PROMPT_ECHO_TYPE);

-- Provides for lists of locator prompt and echo types.

-- POLYMARKER_INDEX                                            Level 0a

type POLYMARKER_INDEX is new POSITIVE;

-- Defines the range of polymarker bundle table indices.

-- MARKER_SIZE                                                 Level 0a

type MARKER_SIZE is new SCALE_FACTOR range 0.0..SCALE_FACTOR'LAST;

-- The size of a marker is indicated by a scale factor.

-- MARKER_TYPE                                                 Level 0a

type MARKER_TYPE is new INTEGER;

-- Defines the type for markers provided by GKS.

-- MARKER_DATA                                                 Level 0b
```

```
-- MARKER_TYPES                                              Level 0a

package MARKER_TYPES is new GKS_LIST_UTILITIES (MARKER_TYPE);

-- Provides for lists of marker types.

-- MORE_EVENTS                                               Level 0a

type MORE_EVENTS is (NOMORE, MORE);

-- Indicates whether more events are contained in the input event queue.

-- NDC_TYPE                                                  LEVE ma

type NDC_TYPE is digits PRECISION;

-- Defines the type of a coordinate in the Normalized Device Coordinate System.

-- NDC                                                       LEVELma

package NDC is new GKS_COORDINATE_SYSTEM (NDC_TYPE);

-- Defines the Normalized Device Coordinate System.

-- NEW_FRAME_NECESSARY                                       Level 0a

type NEW_FRAME_NECESSARY is (NO, YES);

-- Indicates whether a new frame action is necessary at update.

-- OPERATING_MODE                                            Level 0b

type OPERATING_MODE is (REQUEST_MODE,  SAMPLE_MODE, EVENT_MODE);

-- Defines the operating modes of an input device.

-- OPERATING_STATE                                           Level 0a

type OPERATING_STATE is (GKCL, GKOP, WSOP, WSAC, SGOP);

-- Defines the five GKS operating states.

-- PATTERN_INDEX                                             Level 0a

subtype PATTERN_INDEX is STYLE_INDEX range 1..STYLE_INDEX'LAST;

-- Defines the range of pattern table indices.

-- PATTERN_INDICES                                           Level 0a

package PATTERN_INDICES is
        new GKS_LIST_UTILITIES (PATTERN_INDEX);

-- Provides for lists of pattern table indices.
```

```
-- PICK_ID                                                    Level 1b

type PICK_ID is new POSITIVE;

-- Defines the range of pick identifiers available on an implementation.

-- PICK_IDS                                                   Level 1b

package PICK_IDS is new GKS_LIST_UTILITIES (PICK_ID);

-- Provides for lists of pick identifiers.

-- PICK_PROMPT_ECHO_TYPE                                      Level 1b

type PICK_PROMPT_ECHO_TYPE is new INTEGER;

-- Defines the pick prompt and echo type.

-- PICK_PROMPT_ECHO_TYPES                                     Level 1b

package PICK_PROMPT_ECHO_TYPES is new GKS_LIST_UTILITIES
        (PICK_PROMPT_ECHO_TYPE);

-- Provides for lists of pick prompt and echo types.

-- PICK_REQUEST_STATUS                                        Level 1b

type PICK_REQUEST_STATUS is (OK, NOPICK, NONE);

-- Defines the status of a pick input operation  for the request function.

-- PICK_STATUS                                                Level 1b

subtype PICK_STATUS is PICK_REQUEST_STATUS range OK..NOPICK;

-- Defines the status of a pick input operation for the sample, get, and inquiry functions.


-- PIXEL_COLOUR_MATRIX                                        Level 0a

type PIXEL_COLOUR_MATRIX is array (POSITIVE range <>,
     POSITIVE range <>) of PIXEL_COLOUR_INDEX;

-- Provides for matrices of pixel colours.

-- POLYLINE_INDICES                                           Level 0a

package POLYLINE_INDICES is
        new GKS_LIST_UTILITIES (POLYLINE_INDEX);

-- Provides for lists of polyline indices.

-- POLYMARKER_INDICES                                         Level 0a

package POLYMARKER_INDICES is
        new GKS_LIST_UTILITIES (POLYMARKER_INDEX);

-- Provides for lists of polymarker indices.
```

```
-- RADIANS                                                    Level 1a

type RADIANS is digits PRECISION;

-- Values used in performing segment transformations (rotation angle).  Positive indicates
-- an anticlockwise direction.

-- RANGE_OF_EXPANSIONS                                        Level 0a

type RANGE_OF_EXPANSIONS is
        record
           MIN      :CHAR_EXPANSION;
           MAX      :CHAR_EXPANSION;
         end record;

-- Provides a range of character expansion factors.

-- RASTER_UNITS                                               Level 0a

type RASTER_UNITS is new POSITIVE;

-- Defines the range of raster units.

-- RASTER_UNIT_SIZE                                           Level 0a

type RASTER_UNIT_SIZE is
        record
           X : RASTER_UNITS;
           Y : RASTER_UNITS;
        end record;

-- Defines the size of a display screen in raster units on a raster device.

-- REGENERATION_MODE                                          Level 0a

type REGENERATION_MODE is (SUPPRESSED, ALLOWED);

-- Indicates whether implicit regeneration of the display is suppressed or allowed.

-- RELATIVE_PRIORITY                                          Level 0a

type RELATIVE_PRIORITY is (HIGHER,LOWER);

-- Indicates the relative priority between two normalization transformations.

-- RETURN_VALUE_TYPE                                          Level 0a

type RETURN_VALUE_TYPE is (SET,REALIZED);

-- Indicates whether the returned values should be as they were set by the program or as
-- they were actually realized on the device.
```

```
-- SEGMENT_DETECTABILITY                                    Level 1a

type SEGMENT_DETECTABILITY is (UNDETECTABLE, DETECTABLE);

-- Indicates whether a segment is detectable or not.

-- SEGMENT_HIGHLIGHTING                                     Level 1a

type SEGMENT_HIGHLIGHTING is (NORMAL,HIGHLIGHTED);

-- Indicates whether a segment is highlighted or not.

-- SEGMENT_NAME                                             Level 1a

type SEGMENT_NAME is new POSITIVE;

-- Defines the range of segment names.

-- SEGMENT_NAMES                                            Level 1a

package SEGMENT_NAMES is new GKS_LIST_UTILITIES (SEGMENT_NAME);

-- Provides for lists of segment names.

-- SEGMENT_PRIORITY                                         Level 1a

type SEGMENT_PRIORITY is digits PRECISION range 0.0..1.0;

-- Defines the priority of a segment.

-- SEGMENT_VISIBILITY                                       Level 1a

type SEGMENT_VISIBILITY is (VISIBLE, INVISIBLE);

-- Indicates whether a segment is visible or not.

-- STRING_PROMPT_ECHO_TYPE                                  Level 0b

type STRING_PROMPT_ECHO_TYPE is new INTEGER;

-- Defines the string prompt and echo types.

-- STRING_PROMPT_ECHO_TYPES                                 Level 0b

package STRING_PROMPT_ECHO_TYPES is
        new GKS_LIST_UTILITIES (STRING_PROMPT_ECHO_TYPE);

-- Provides for lists of string prompt and echo types.

-- STROKE_PROMPT_ECHO_TYPE                                  Level 0b

type STROKE_PROMPT_ECHO_TYPE is new INTEGER;

-- Defines the stroke prompt and echo types.

-- STROKE_PROMPT_ECHO_TYPES                                 Level 0b
```

```
package STROKE_PROMPT_ECHO_TYPES is
        new GKS_LIST_UTILITIES (STROKE_PROMPT_ECHO_TYPE);

-- Provides for lists of stroke prompt and echo types.

-- VERTICAL_ALIGNMENT                                          Level 0a

type VERTICAL_ALIGNMENT is (NORMAL, TOP, CAP, HALF, BASE, BOTTOM);

-- The alignment of the text extent parallelogram with respect to the vertical positioning of
-- the text.

-- TEXT_ALIGNMENT                                              Level 0a

type TEXT_ALIGNMENT is
      record
         HORIZONTAL          : HORIZONTAL_ALIGNMENT;
         VERTICAL            : VERTICAL_ALIGNMENT;
      end record;

-- The type of the attribute controlling the positioning of the text extent parallelogram
-- in relation to the text position, having horizontal and vertical components as
-- defined above.

-- WC_TYPE                                                     Level 0a

type WC_TYPE is digits PRECISION;

-- Defines the range of accuracy for World Coordinate types.

-- WC                                                          Level 0a

package WC is new GKS_COORDINATE_SYSTEM (WC_TYPE);

-- Defines the World Coordinate System.

-- TEXT_EXTENT_PARALLELOGRAM                                   Level 0a

type TEXT_EXTENT_PARALLELOGRAM is
      record
         LOWER_LEFT          : WC.POINT;
         LOWER_RIGHT         : WC.POINT;
         UPPER_RIGHT         : WC.POINT;
         UPPER_LEFT          : WC.POINT;
      end record;

-- Defines the corner points of the text extent parallelogram with respect to the
-- vertical positioning of the text.

-- TEXT_FONT                                                   Level 0a

type TEXT_FONT is new INTEGER;

-- Defines the types of fonts provided by the implementation.
```

```
-- TEXT_PRECISION                                              Level 0a

type TEXT_PRECISION is  (STRING_PRECISION,
                         CHAR_PRECISION,
                         STROKE_PRECISION);

-- The precision with which text appears.


-- TEXT_FONT_PRECISION                                         Level 0a

type TEXT_FONT_PRECISION is
      record
         FONT              : TEXT_FONT;
         PRECISION         : TEXT_PRECISION;
      end record;

-- This type defines a record describing the text font and precision aspect.

-- TEXT_FONT_PRECISIONS                                        Level 0a

package TEXT_FONT_PRECISIONS is
         new GKS_LIST_UTILITIES (TEXT_FONT_PRECISION);

-- Provides for lists of text font and precision pairs.

-- TEXT_INDEX                                                  Level 0a

type TEXT_INDEX is new POSITIVE;

-- Defincs the range of text bundle table indices.

-- TEXT_INDICES                                                Level 0a

package TEXT_INDICES is new GKS_LIST_UTILITIES (TEXT_INDEX);

-- Provides for lists of text indices.

-- TEXT_PATH                                                   Level 0a

type TEXT_PATH is (RIGHT,LEFT,UP,DOWN);

-- The direction taken by a text string.

-- TRANSFORMATION_FACTOR                                       Level 1a

type TRANSFORMATION_FACTOR is
      record
         X : NDC_TYPE;
         Y : NDC_TYPE;
      end record;

-- Scale factors used in building transformation matrices for performing segment
-- transformations.
```

```
-- TRANSFORMATION_MATRIX                                    Level 1a

type TRANSFORMATION_MATRIX is array (1..2, 1..3) of NDC_TYPE;

-- For segment transformations mapping within NDC space.

-- TRANSFORMATION_NUMBER                                    Level 0a

type TRANSFORMATION_NUMBER is new NATURAL;

-- A normalization transformation number.

subtype POSITIVE_TRANSFORMATION_NUMBER is
        TRANSFORMATION_NUMBER
        range 1 .. TRANSFORMATION_NUMBER'LAST;

-- A normalization transformation number corresponding to a settable transformation.

-- TRANSFORMATION_PRIORITY_ARRAY                            Level 0a

type TRANSFORMATION_PRIORITY_ARRAY is array (POSITIVE range <>)
      of TRANSFORMATION_NUMBER;

-- Type to store transformation numbers.

-- TRANSFORMATION_PRIORITY_LIST                             Level 0a

type TRANSFORMATION_PRIORITY_LIST(LENGTH:SMALL_NATURAL:=0) is
        record
           CONTENTS : TRANSFORMATION_PRIORITY_ARRAY (1..LENGTH);
        end record;

-- Provides for a prioritised list of transformation numbers.

-- UPDATE_REGENERATION_FLAG                                 Level 0a

type UPDATE_REGENERATION_FLAG is (PERFORM, POSTPONE);

-- Flag indicating regeneration action on display.

-- UPDATE_STATE                                             Level 0a

type UPDATE_STATE is (NOTPENDING, PENDING);

-- Indicates whether or not a workstation transformation change has been requested
-- and not yet provided.

-- VALUATOR_INPUT_VALUE                                     Level 0b

type VALUATOR_INPUT_VALUE is digits PRECISION;

-- Defines the range of accuracy of input values on an implementation.

-- VALUATOR_PROMPT_ECHO_TYPE                                Level 0b

type VALUATOR_PROMPT_ECHO_TYPE is new INTEGER;

-- Defines the possible range of valuator prompt and echo types.
```

```
-- VALUATOR_PROMPT_ECHO_TYPES                                    Level 0b

package VALUATOR_PROMPT_ECHO_TYPES is
        new GKS_LIST_UTILITIES (VALUATOR_PROMPT_ECHO_TYPE);

-- Provides for lists of valuator prompt and echo types.

-- VARIABLE_COLOUR_MATRIX                                        Level 0a

type VARIABLE_COLOUR_MATRIX  (DX : SMALL_NATURAL := 0;
                              DY : SMALL_NATURAL := 0) is
      record
        MATRIX : COLOUR_MATRIX (1..DX, 1..DY);
      end record;

-- Provides for variable sized matrices containing colour indices corresponding to
-- a cell array or pattern array.

-- VARIABLE_CONNECTION_ID                                        Level 0a

type VARIABLE_CONNECTION_ID
     (LENGTH : STRING_SMALL_NATURAL := 0) is
       record
         CONNECT : STRING (1..LENGTH);
       end record;

-- Defines a variable length connection identifier for
-- INQ_WS_CONNECTION_AND_TYPE

-- VARIABLE_PIXEL_COLOUR_MATRIX                                  Level 0a

type VARIABLE_PIXEL_COLOUR_MATRIX  (DX : SMALL_NATURAL := 0;
                                    DY : SMALL_NATURAL := 0) is
      record
        MATRIX : PIXEL_COLOUR_MATRIX (1..DX, 1..DY);
      end record;

-- Provides for variable sized matrices of pixel colours.

-- WS_CATEGORY                                                   Level 0a

type WS_CATEGORY is (OUTPUT, INPUT, OUTIN, WISS, MO, MI);

-- Type for GKS workstation categories.

-- WS_ID                                                         Level 0a

type WS_ID is new POSITIVE;

-- Defines the range of workstation identifiers.

-- WS_IDS                                                        Level 0a

package WS_IDS is new GKS_LIST_UTILITIES (WS_ID);

-- Provides for lists of workstation identifiers.
```

```
-- WS_STATES                                                    Level 0a

type WS_STATE is (INACTIVE, ACTIVE);

-- The state of a workstation.

-- WS_TYPES                                                     Level 0a
type WS_TYPE is new POSITIVE;

-- Range of values corresponding to valid workstation types.  Constants specifying names
-- for the various types of workstations should be provided by an implementation.

-- WS_TYPES                                                     Level 0a

package WS_TYPES is new GKS_LIST_UTILITIES (WS_TYPE);

-- Provides for lists of workstation types.

-- INDIVIDUAL_ATTRIBUTE_VALUES                                  Level 0a

type INDIVIDUAL_ATTRIBUTE_VALUES is
      record
         TYPE_OF_LINE          : LINETYPE;
         WIDTH                 : LINEWIDTH;
         LINE_COLOUR           : COLOUR_INDEX;
         TYPE_OF_MARKER        : MARKER_TYPE;
         SIZE                  : MARKER_SIZE;
         MARKER_COLOUR         : COLOUR_INDEX;
         FONT_PRECISION        : TEXT_FONT_PRECISION;
         EXPANSION             : CHAR_EXPANSION;
         SPACING               : CHAR_SPACING;
         TEXT_COLOUR           : COLOUR_INDEX;
         INTERIOR              : INTERIOR_STYLE;
         STYLE                 : STYLE_INDEX;
         FILL_AREA_COLOUR      : COLOUR_INDEX;
         ASF                   : ASF_LIST;
      end record;

-- A record containing all of the current individual attributes for the procedure
-- INQ_CURRENT_INDIVIDUAL_ATTRIBUTE_VALUES.
```

```
-- PRIMITIVE_ATTRIBUTE_VALUES

type PRIMITIVE_ATTRIBUTE_VALUES is
      record
         INDEX_POLYLINE                : POLYLINE_INDEX;
         INDEX_POLYMARKER              : POLYMARKER_INDEX;
         INDEX_TEXT                    : TEXT_INDEX;
         CHAR_HEIGHT                   : WC.MAGNITUDE;
         CHAR_UP_VECTOR                : WC.VECTOR;
         CHAR_WIDTH                    : WC.MAGNITUDE;
         CHAR_BASE_VECTOR              : WC.VECTOR;
         PATH                          : TEXT_PATH;
         ALIGNMENT                     : TEXT_ALIGNMENT;
         INDEX_FILL_AREA               : FILL_AREA_INDEX;
         PATTERN_WIDTH_VECTOR          : WC.VECTOR;
         PATTERN_HEIGHT_VECTOR         : WC.VECTOR;
         PATTERN_REFERENCE_POINT       : WC.POINT;
      end record;

-- A record containing all of the current primitive attributes for the procedure
-- INQ_CURRENT_PRIMITIVE_ATTRIBUTE_VALUES.
```

The following defines the predefined exception GKS_ERROR defined in Section 3.2.3.

```
GKS_ERROR                           : exception;

-- Following are the declarations of implementation dependent constants for defining GKS/Ada
-- types. Some of the constants are used for defining default parameter values for GKS
-- procedures.

-- The following constants define the GKS standard line types

SOLID_LINE                          : constant LINETYPE          := 1;
DASHED_LINE                         : constant LINETYPE          := 2;
DOTTED_LINE                         : constant LINETYPE          := 3;
DASHED_DOTTED_LINE                  : constant LINETYPE          := 4;
```

The following constants define the GKS standard marker types:

```
DOT_MARKER                          : constant MARKER_TYPE       := 1;
PLUS_MARKER                         : constant MARKER_TYPE       := 2;
STAR_MARKER                         : constant MARKER_TYPE       := 3;
ZERO_MARKER                         : constant MARKER_TYPE       := 4;
X_MARKER                            : constant MARKER_TYPE       := 5;

-- The following constants define the prompt and echo types supported by GKS:

DEFAULT_LOCATOR                     : constant LOCATOR_PROMPT_ECHO_TYPE := 1;
CROSS_HAIR_LOCATOR                  : constant LOCATOR_PROMPT_ECHO_TYPE := 2;
TRACKING_CROSS_LOCATOR              : constant LOCATOR_PROMPT_ECHO_TYPE := 3;
RUBBER_BAND_LINE_LOCATOR            : constant LOCATOR_PROMPT_ECHO_TYPE := 4;
RECTANGLE_LOCATOR                   : constant LOCATOR_PROMPT_ECHO_TYPE := 5;
DIGITAL_LOCATOR                     : constant LOCATOR_PROMPT_ECHO_TYPE := 6;

DEFAULT_STROKE                      : constant STROKE_PROMPT_ECHO_TYPE  := 1;
DIGITAL_STROKE                      : constant STROKE_PROMPT_ECHO_TYPE  := 2;
MARKER_STROKE                       : constant STROKE_PROMPT_ECHO_TYPE  := 3;
LINE_STROKE                         : constant STROKE_PROMPT_ECHO_TYPE  := 4;
```

```
DEFAULT_VALUATOR          : constant VALUATOR_PROMPT_ECHO_TYPE  := 1;
GRAPHICAL_VALUATOR        : constant VALUATOR_PROMPT_ECHO_TYPE  := 2;
DIGITAL_VALUATOR          : constant VALUATOR_PROMPT_ECHO_TYPE  := 3;

DEFAULT_CHOICE            : constant CHOICE_PROMPT_ECHO_TYPE    := 1;
PROMPT_ECHO_CHOICE        : constant CHOICE_PROMPT_ECHO_TYPE    := 2;
STRING_PROMPT_CHOICE      : constant CHOICE_PROMPT_ECHO_TYPE    := 3;
STRING_INPUT_CHOICE       : constant CHOICE_PROMPT_ECHO_TYPE    := 4;
SEGMENT_CHOICE            : constant CHOICE_PROMPT_ECHO_TYPE    := 5;

DEFAULT_STRING            : constant STRING_PROMPT_ECHO_TYPE    := 1;

DEFAULT_PICK              : constant PICK_PROMPT_ECHO_TYPE      := 1;
GROUP_HIGHLIGHT_PICK      : constant PICK_PROMPT_ECHO_TYPE      := 2;

SEGMENT_HIGHLIGHT_PICK    : constant PICK_PROMPT_ECHO_TYPE      := 3;

-- The following constants are used for defining default parameter value for GKS procedures.

DEFAULT_MEMORY_UNITS      : constant                            := 0;
DEFAULT_ERROR_FILE        : constant  STRING                    := " ";


end GKS_TYPES;
```

```
-- THE GKS PACKAGE

with GKS_TYPES;
use GKS_TYPES;

package GKS is

-- The package GKS contains all of the procedures that are required to implement GKS level 2c.

-- The following data types are the GKS private types and are included in package GKS for ease
-- of manipulation.

-- CHOICE_DATA_RECORD                Level 0b

      type CHOICE_DATA_RECORD (PROMPT_ECHO_TYPE :
            CHOICE_PROMPT_ECHO_TYPE := DEFAULT_CHOICE) is private;

      -- Defines a record for initialising choice input.

-- GKSM_DATA_RECORD                                                    Level 0b

      type GKSM_DATA_RECORD (TYPE_OF_ITEM        : GKSM_ITEM_TYPE := 0;
                             LENGTH              : NATURAL := 0) is private;

      -- A data record for GKSM metafiles.

-- LOCATOR_DATA_RECORD                                                 Level 0b

      type LOCATOR_DATA_RECORD (PROMPT_ECHO_TYPE :
            LOCATOR_PROMPT_ECHO_TYPE := DEFAULT_LOCATOR)  is private;

      -- Defines a record for initialising locator input.

-- PICK_DATA_RECORD                                                    Level 1b

      type PICK_DATA_RECORD (PROMPT_ECHO_TYPE :
            PICK_PROMPT_ECHO_TYPE := DEFAULT_PICK) is private;

      -- Defines a record for initialising pick input.

-- STRING_DATA_RECORD                Level 0b

      type STRING_DATA_RECORD (PROMPT_ECHO_TYPE :
            STRING_PROMPT_ECHO_TYPE := DEFAULT_STRING) is private;

      -- Defines a record for initialising string input.

-- STROKE_DATA_RECORD                                                  Level 0b

      type STROKE_DATA_RECORD (PROMPT_ECHO_TYPE :
            STROKE_PROMPT_ECHO_TYPE := DEFAULT_STROKE) is private;

      -- Defines a record for initialising stroke input.

-- VALUATOR_DATA_RECORD                                                Level 0b

      type VALUATOR_DATA_RECORD (PROMPT_ECHO_TYPE :
```

```
        VALUATOR_PROMPT_ECHO_TYPE := DEFAULT_VALUATOR) is private;

    -- Defines a record for initialising valuator input.

-- Subprograms for Manipulating Input Data Records

-- The procedures and functions defined below are those necessary for constructing and
-- inquiring the input data records, declared as private types in this package for each of
-- the six classes of input devices defined by the GKS specification the Locator,
-- Stroke, Valuator, Choice, Pick, and String logical devices. The procedures listed here
-- are used to construct the data records for each of the prompt and echo types of a device
-- class to be used for initializing a particular input device. Assorted functions are also
-- provided so that an application of GKS/Ada may examine the parts of the data record
-- which are defined by GKS. Any implementation specific information in the data
-- records is kept private and unavailable. The exception GKS_ERROR (class
-- LANGUAGE_BINDING_ERROR) is raised if any of the below procedures are used
-- incorrectly. That is, if an illegal prompt and echo type is used then error number 2500 is
-- logged onto the error file.

-- Locator Data Record Operations

    procedure BUILD_LOCATOR_DATA_RECORD
            (PROMPT_ECHO_TYPE       : in LOCATOR_PROMPT_ECHO_TYPE;
             DATA_RECORD            : out LOCATOR_DATA_RECORD);

-- Constructs and returns a locator data record.

procedure BUILD_STROKE_DATA_RECORD
      (PROMPT_ECHO_TYPE             : in STROKE_PROMPT_ECHO_TYPE;
       BUFFER_SIZE                  : in POSITIVE;
       DATA_RECORD                  : out STROKE_DATA_RECORD);

-- Constructs and returns a stroke data record.
```

```
function BUFFER_SIZE (DATA_RECORD : in STROKE_DATA_RECORD)
   return  POSITIVE;

-- Valuator Data Record Operations.

procedure BUILD_VALUATOR_DATA_RECORD
      (PROMPT_ECHO_TYPE                 : in VALUATOR_PROMPT_ECHO_TYPE;
       LOW_VALUE                        : in VALUATOR_INPUT_VALUE;
       HIGH_VALUE                       : in VALUATOR_INPUT_VALUE;
       DATA_RECORD                      : out VALUATOR_DATA_RECORD);

-- Constructs and returns a valuator data record.

function HIGH_VALUE (DATA_RECORD : in VALUATOR_DATA_RECORD)
       return VALUATOR_INPUT_VALUE;

-- Returns the high value for the valuator stored in the valuator data record.

function LOW_VALUE (DATA_RECORD : in VALUATOR_DATA_RECORD)
       return VALUATOR_INPUT_VALUE;

-- Returns the low value for the valuator stored in the valuator data record.

-- Choice Data Record Operations.

procedure BUILD_CHOICE_DATA_RECORD
      (PROMPT_ECHO_TYPE                 : in CHOICE_PROMPT_ECHO_TYPE;
       DATA_RECORD                      : out CHOICE_DATA_RECORD);

-- Constructs and returns a choice data record.

-- Pick Data Record Operation.

procedure BUILD_PICK_DATA_RECORD
      (PROMPT_ECHO_TYPE                 : in PICK_PROMPT_ECHO_TYPE;
       DATA_RECORD                      : out PICK_DATA_RECORD);

-- Construct and returns a pick data record.

-- String Data Record Operations.

procedure BUILD_STRING_DATA_RECORD
      (PROMPT_ECHO_TYPE                 : in STRING_PROMPT_ECHO_TYPE;
       INPUT_BUFFER_SIZE                : in POSITIVE;
       INITIAL_CURSOR_POSITION          : in NATURAL;
       DATA_RECORD                      : out STRING_DATA_RECORD);

-- Construct and returns a string data record.

function INPUT_BUFFER_SIZE
      (DATA_RECORD : in STRING_DATA_RECORD) return NATURAL;

-- Returns the size of the buffer used for storing string input stored in the string data record.
function INITIAL_CURSOR_POSITION
      (DATA_RECORD : in STRING_DATA_RECORD) return NATURAL;

-- Returns the initial cursor position for string input stored in the string data record.
```

```
-- GKS procedures

-- CONTROL FUNCTIONS

procedure OPEN_GKS
      (ERROR_FILE              : in  STRING     := DEFAULT_ERROR_FILE;
       AMOUNT_OF_MEMORY        : in NATURAL     := DEFAULT_MEMORY_UNITS);

procedure CLOSE_GKS;

procedure OPEN_WS
      (WS              : in WS_ID;
       CONNECTION      : in STRING;
       TYPE_OF_WS      : in WS_TYPE);

procedure CLOSE_WS
      (WS : in WS_ID);

procedure ACTIVATE_WS
      (WS : in WS_ID);

procedure DEACTIVATE_WS
      (WS : in WS_ID);

procedure CLEAR_WS
      (WS              : in WS_ID;
       FLAG            : in CONTROL_FLAG);

procedure REDRAW_ALL_SEGMENTS_ON_WS
      (WS : in WS_ID);

procedure UPDATE_WS
      (WS                      : in WS_ID;
       REGENERATION            : in UPDATE_REGENERATION_FLAG);

procedure SET_DEFERRAL_STATE
      (WS                      : in WS_ID;
       DEFERRAL                : in DEFERRAL_MODE;
       REGENERATION            : in REGENERATION_MODE);

procedure MESSAGE
      (WS                      : in WS_ID;
       CONTENTS                : in STRING);

-- OUTPUT FUNCTIONS

procedure POLYLINE
      (POINTS                  : in WC.POINT_ARRAY);

procedure POLYMARKER
      (POINTS                  : in WC.POINT_ARRAY);

procedure TEXT
      (POSITION                : in WC.POINT;
       CHAR_STRING             : in STRING);

procedure FILL_AREA
      (POINTS                  : in WC.POINT_ARRAY);
```

```
procedure CELL_ARRAY
      (CORNER_1_1          : in WC.POINT;
       CORNER_DX_DY        : in WC.POINT;
       CELLS               : in COLOUR_MATRIX);

-- OUTPUT ATTRIBUTE FUNCTIONS

procedure SET_POLYLINE_INDEX
      (INDEX               : in POLYLINE_INDEX);

procedure SET_LINETYPE
      (TYPE_OF_LINE : in LINETYPE);

procedure SET_LINEWIDTH_SCALE_FACTOR
      (WIDTH               : in LINEWIDTH);

procedure SET_POLYLINE_COLOUR_INDEX
      (LINE_COLOUR         : in COLOUR_INDEX);

procedure SET_POLYMARKER_INDEX
      (INDEX               : in POLYMARKER_INDEX);

procedure SET_MARKER_TYPE
      (TYPE_OF_MARKER : in MARKER_TYPE);

procedure SET_MARKER_SIZE_SCALE_FACTOR
      (SIZE                : in MARKER_SIZE);

procedure SET_POLYMARKER_COLOUR_INDEX
      (MARKER_COLOUR : in COLOUR_INDEX);

procedure SET_TEXT_INDEX
      (INDEX               : in TEXT_INDEX);

procedure SET_TEXT_FONT_AND_PRECISION
      (FONT_PRECISION : in TEXT_FONT_PRECISION);

procedure SET_CHAR_EXPANSION_FACTOR
      (EXPANSION           : in CHAR_EXPANSION);

procedure SET_CHAR_SPACING
      (SPACING             : in CHAR_SPACING);


procedure SET_TEXT_COLOUR_INDEX
      (TEXT_COLOUR    : in COLOUR_INDEX);

procedure SET_CHAR_HEIGHT
      (HEIGHT                    : in WC.MAGNITUDE);

procedure SET_CHAR_UP_VECTOR
      (CHAR_UP_VECTOR      : in WC.VECTOR);

procedure SET_TEXT_PATH
      (PATH                      : in TEXT_PATH);
```

```
procedure SET_TEXT_ALIGNMENT
      (ALIGNMENT              : in TEXT_ALIGNMENT);

procedure SET_FILL_AREA_INDEX
      (INDEX                  : in FILL_AREA_INDEX);

procedure SET_FILL_AREA_INTERIOR_STYLE
      (INTERIOR               : in INTERIOR_STYLE);

procedure SET_FILL_AREA_STYLE_INDEX
      (STYLE                  : in STYLE_INDEX);

procedure SET_FILL_AREA_COLOUR_INDEX
      (FILL_AREA_COLOUR       : in COLOUR_INDEX);

procedure SET_PATTERN_SIZE
      (SIZE                   : in WC.SIZE);

procedure SET_PATTERN_REFERENCE_POINT
      (POINT                  : in WC.POINT);

procedure SET_ASF
      (ASF                    : in ASF_LIST);

procedure SET_PICK_ID
      (PICK                   : in PICK_ID);

procedure SET_POLYLINE_REPRESENTATION
      (WS                           : in WS_ID;
       INDEX                        : in POLYLINE_INDEX;
       TYPE_OF_LINE                 : in LINETYPE;
       WIDTH                        : in LINEWIDTH;
       LINE_COLOUR                  : in COLOUR_INDEX);

procedure SET_POLYMARKER_REPRESENTATION
      (WS                           : in WS_ID;
       INDEX                        : in POLYMARKER_INDEX;
       TYPE_OF_MARKER               : in MARKER_TYPE;
       SIZE                         : in MARKER_SIZE;
       MARKER_COLOUR                : in COLOUR_INDEX);

procedure SET_TEXT_REPRESENTATION
      (WS                           : in WS_ID;
       INDEX                        : in TEXT_INDEX;
       FONT_PRECISION               : in TEXT_FONT_PRECISION;
       EXPANSION                    : in CHAR_EXPANSION;
       SPACING                      : in CHAR_SPACING;
       TEXT_COLOUR                  : in COLOUR_INDEX);

procedure SET_FILL_AREA_REPRESENTATION
      (WS                           : in WS_ID;
       INDEX                        : in FILL_AREA_INDEX;
       INTERIOR                     : in INTERIOR_STYLE;
       STYLE                        : in STYLE_INDEX;
       FILL_AREA_COLOUR             : in COLOUR_INDEX);
```

```
procedure SET_PATTERN_REPRESENTATION
      (WS                              : in WS_ID;
       INDEX                           : in PATTERN_INDEX;
       PATTERN                         : in COLOUR_MATRIX);

procedure SET_COLOUR_REPRESENTATION
      (WS                              : in WS_ID;
       INDEX                           : in COLOUR_INDEX;
       RGB_COLOUR                      : in COLOUR_REPRESENTATION);

-- TRANSFORMATION FUNCTIONS

procedure SET_WINDOW
      (TRANSFORMATION    : in POSITIVE_TRANSFORMATION_NUMBER;
       WINDOW_LIMITS     : in WC.RECTANGLE_LIMITS);

procedure SET_VIEWPORT
      (TRANSFORMATION    : in POSITIVE_TRANSFORMATION_NUMBER;
       VIEWPORT_LIMITS   : in NDC.RECTANGLE_LIMITS);

procedure SET_VIEWPORT_INPUT_PRIORITY
      (TRANSFORMATION                  : in TRANSFORMATION_NUMBER;
       REFERENCE_TRANSFORMATION        : in TRANSFORMATION_NUMBER;
       PRIORITY                        : in RELATIVE_PRIORITY);

procedure SELECT_NORMALIZATION_TRANSFORMATION
      (TRANSFORMATION    : in TRANSFORMATION_NUMBER);

procedure SET_CLIPPING_INDICATOR
      (CLIPPING                        : in CLIPPING_INDICATOR);

procedure SET_WS_WINDOW
      (WS                              : in WS_ID;
       WS_WINDOW_LIMITS                : in NDC.RECTANGLE_LIMITS);

procedure SET_WS_VIEWPORT
      (WS                              : in WS_ID;
       WS_VIEWPORT_LIMITS              : in DC.RECTANGLE_LIMITS);

-- SEGMENT FUNCTIONS

procedure CREATE_SEGMENT
      (SEGMENT                         : in SEGMENT_NAME);

procedure CLOSE_SEGMENT;

procedure RENAME_SEGMENT
      (OLD_NAME                        : in SEGMENT_NAME;
       NEW_NAME                        : in SEGMENT_NAME);

procedure DELETE_SEGMENT
      (SEGMENT                         : in SEGMENT_NAME);

procedure DELETE_SEGMENT_FROM_WS
      (WS                              : in WS_ID;
       SEGMENT                         : in SEGMENT_NAME);
```

```
procedure ASSOCIATE_SEGMENT_WITH_WS
      (WS                    : in WS_ID;
       SEGMENT               : in SEGMENT_NAME);

procedure COPY_SEGMENT_TO_WS
      (WS                    : in WS_ID;
       SEGMENT               : in SEGMENT_NAME);

procedure INSERT_SEGMENT
      (SEGMENT               : in SEGMENT_NAME;
       TRANSFORMATION        : in TRANSFORMATION_MATRIX);

procedure SET_SEGMENT_TRANSFORMATION
      (SEGMENT               : in SEGMENT_NAME;
       TRANSFORMATION        :  in TRANSFORMATION_MATRIX);

procedure SET_VISIBILITY
      (SEGMENT               : in SEGMENT_NAME;
       VISIBILITY            : in SEGMENT_VISIBILITY);

procedure SET_HIGHLIGHTING
      (SEGMENT               : in SEGMENT_NAME;
       HIGHLIGHTING          : in SEGMENT_HIGHLIGHTING);

procedure SET_SEGMENT_PRIORITY
      (SEGMENT               : in SEGMENT_NAME;
       PRIORITY              : in SEGMENT_PRIORITY);

procedure SET_DETECTABILITY
      (SEGMENT               : in SEGMENT_NAME;
       DETECTABILITY         : in SEGMENT_DETECTABILITY);

-- INPUT FUNCTIONS

procedure INITIALISE_LOCATOR
      (WS                    : in WS_ID;
       DEVICE                : in LOCATOR_DEVICE_NUMBER;
       INITIAL_TRANSFORMATION: in TRANSFORMATION_NUMBER;
       INITIAL_POSITION      : in WC.POINT;
       ECHO_AREA             : in DC.RECTANGLE_LIMITS;
       DATA_RECORD           : in LOCATOR_DATA_RECORD);

procedure INITIALISE_STROKE
      (WS                    : in WS_ID;
       DEVICE                : in STROKE_DEVICE_NUMBER;
       INITIAL_TRANSFORMATION: in TRANSFORMATION_NUMBER;
       INITIAL_STROKE        : in WC.POINT_ARRAY;
       ECHO_AREA             : in DC.RECTANGLE_LIMITS;
       DATA_RECORD           : in STROKE_DATA_RECORD);

procedure INITIALISE_VALUATOR
      (WS                    : in WS_ID;
      DEVICE                 : in VALUATOR_DEVICE_NUMBER;
      INITIAL_VALUE          : in VALUATOR_INPUT_VALUE;
      ECHO_AREA              : in DC.RECTANGLE_LIMITS;
      DATA_RECORD            : in VALUATOR_DATA_RECORD);
```

```
procedure INITIALISE_CHOICE
      (WS                    : in WS_ID;
       DEVICE                : in CHOICE_DEVICE_NUMBER;
       INITIAL_STATUS        : in CHOICE_STATUS;
       INITIAL_CHOICE        : in CHOICE_VALUE;
       ECHO_AREA             : in DC.RECTANGLE_LIMITS;
       DATA_RECORD           : in CHOICE_DATA_RECORD);

procedure INITIALISE_PICK
      (WS                    : in WS_ID;
       DEVICE                : in PICK_DEVICE_NUMBER;
       INITIAL_STATUS        : in PICK_STATUS;
       INITIAL_SEGMENT       : in SEGMENT_NAME;
       INITIAL_PICK          : in PICK_ID;
       ECHO_AREA             : in DC.RECTANGLE_LIMITS;
       DATA_RECORD           : in PICK_DATA_RECORD);

procedure INITIALISE_STRING
      (WS                    : in WS_ID;
       DEVICE                : in STRING_DEVICE_NUMBER;
       INITIAL_STRING        : in INPUT_STRING;
       ECHO_AREA             : in DC.RECTANGLE_LIMITS;
       DATA_RECORD           : in STRING_DATA_RECORD);

procedure SET_LOCATOR_MODE
      (WS                    : in WS_ID;
       DEVICE                : in LOCATOR_DEVICE_NUMBER;
       MODE                  : in OPERATING_MODE;
       SWITCH                : in ECHO_SWITCH);

procedure SET_STROKE_MODE
      (WS                    : in WS_ID;
       DEVICE                : in STROKE_DEVICE_NUMBER;
       MODE                  : in OPERATING_MODE;
       SWITCH                : in ECHO_SWITCH);

procedure SET_VALUATOR_MODE
      (WS                    : in WS_ID;
       DEVICE                : in VALUATOR_DEVICE_NUMBER;
       MODE                  : in OPERATING_MODE;
       SWITCH                : in ECHO_SWITCH);

procedure SET_CHOICE_MODE
      (WS                    : in WS_ID;
       DEVICE                : in CHOICE_DEVICE_NUMBER;
       MODE                  : in OPERATING_MODE;
       SWITCH                : in ECHO_SWITCH);

procedure SET_PICK_MODE
      (WS                    : in WS_ID;
       DEVICE                : in PICK_DEVICE_NUMBER;
       MODE                  : in OPERATING_MODE;
       SWITCH                : in ECHO_SWITCH);
```

```
procedure SET_STRING_MODE
      (WS                    : in WS_ID;
       DEVICE                : in STRING_DEVICE_NUMBER;
       MODE                  : in OPERATING_MODE;
       SWITCH                : in ECHO_SWITCH);

procedure REQUEST_LOCATOR
      (WS                    : in WS_ID;
       DEVICE                : in LOCATOR_DEVICE_NUMBER;
       STATUS                : out INPUT_STATUS;
       TRANSFORMATION        : out TRANSFORMATION_NUMBER;
       POSITION              : out WC.POINT);

procedure REQUEST_STROKE
      (WS                    : in WS_ID;
       DEVICE                : in STROKE_DEVICE_NUMBER;
       STATUS                : out INPUT_STATUS;
       TRANSFORMATION        : out TRANSFORMATION_NUMBER;
       STROKE_POINTS         : out WC.POINT_LIST);

procedure REQUEST_VALUATOR
      (WS                    : in WS_ID;
       DEVICE                : in VALUATOR_DEVICE_NUMBER;
       STATUS                : out INPUT_STATUS;
       VALUE                 : out VALUATOR_INPUT_VALUE);

procedure REQUEST_CHOICE
      (WS                    : in WS_ID;
       DEVICE                : in CHOICE_DEVICE_NUMBER;
       STATUS                : out CHOICE_REQUEST_STATUS;
       CHOICE_NUMBER         : out CHOICE_VALUE);

procedure REQUEST_PICK
      (WS                    : in WS_ID;
       DEVICE                : in PICK_DEVICE_NUMBER;
       STATUS                : out PICK_REQUEST_STATUS;
       SEGMENT               : out SEGMENT_NAME;
       PICK                  : out PICK_ID);

procedure REQUEST_STRING
      (WS                    : in WS_ID;
       DEVICE                : in STRING_DEVICE_NUMBER;
       STATUS                : out INPUT_STATUS;
       CHAR_STRING           : out INPUT_STRING);

procedure SAMPLE_LOCATOR
      (WS                    : in WS_ID;
       DEVICE                : in LOCATOR_DEVICE_NUMBER;
       TRANSFORMATION        : out TRANSFORMATION_NUMBER;
       POSITION              : out WC.POINT);

procedure SAMPLE_STROKE
      (WS                    : in WS_ID;
       DEVICE                : in STROKE_DEVICE_NUMBER;
       TRANSFORMATION        : out TRANSFORMATION_NUMBER;
       STROKE_POINTS         : out WC.POINT_LIST);
```

```
procedure SAMPLE_VALUATOR
      (WS                          : in WS_ID;
       DEVICE                      : in VALUATOR_DEVICE_NUMBER;
       VALUE                       : out VALUATOR_INPUT_VALUE);

procedure SAMPLE_CHOICE
      (WS                          : in WS_ID;
       DEVICE                      : in CHOICE_DEVICE_NUMBER;
       STATUS                      : out CHOICE_STATUS;
       CHOICE_NUMBER               : out CHOICE_VALUE);

procedure SAMPLE_PICK
      (WS                          : in WS_ID;
       DEVICE                      : in PICK_DEVICE_NUMBER;
       STATUS                      : out PICK_STATUS;
       SEGMENT                     : out SEGMENT_NAME;
       PICK                        : out PICK_ID);

procedure SAMPLE_STRING
      (WS                          : in WS_ID;
       DEVICE                      : in STRING_DEVICE_NUMBER;
       CHAR_STRING                 : out INPUT_STRING);

procedure AWAIT_EVENT
      (TIMEOUT                     : in DURATION;
       WS                          : out WS_ID;
       CLASS                       : out INPUT_CLASS;
       DEVICE                      : out EVENT_DEVICE_NUMBER);

procedure FLUSH_DEVICE_EVENTS
      (WS                          : in WS_ID;
       CLASS                       : in INPUT_QUEUE_CLASS;
       DEVICE                      : in EVENT_OVERFLOW_DEVICE_NUMBER);

procedure GET_LOCATOR
      (TRANSFORMATION              : out TRANSFORMATION_NUMBER;
       POSITION                    : out WC.POINT);

procedure GET_STROKE
      (TRANSFORMATION              : out TRANSFORMATION_NUMBER;
       STROKE_POINTS               : out WC.POINT_LIST);

procedure GET_VALUATOR
      (VALUE                       : out VALUATOR_INPUT_VALUE);

procedure GET_CHOICE
      (STATUS                      : out CHOICE_STATUS;
       CHOICE_NUMBER               : out CHOICE_VALUE);

procedure GET_PICK
      (STATUS                      : out PICK_STATUS;
       SEGMENT                     : out SEGMENT_NAME;
       PICK                        : out PICK_ID);

procedure GET_STRING
      (CHAR_STRING                 : out INPUT_STRING);
```

```
-- METAFILE FUNCTIONS

procedure WRITE_ITEM_TO_GKSM
      (WS                              : in WS_ID;
       ITEM                            : in GKSM_DATA_RECORD);

procedure GET_ITEM_TYPE_FROM_GKSM
      (WS                              : in WS_ID;
       TYPE_OF_ITEM                    : out GKSM_ITEM_TYPE;
       LENGTH                          : out NATURAL);

procedure READ_ITEM_FROM_GKSM
      (WS                              : in WS_ID;
       MAX_LENGTH                      : in NATURAL;
       ITEM                            : out GKSM_DATA_RECORD);

procedure INTERPRET_ITEM
      (ITEM                  : in GKSM_DATA_RECORD);

-- INQUIRY FUNCTIONS

procedure INQ_OPERATING_STATE_VALUE
      (VALUE                 : out OPERATING_STATE);

procedure INQ_LEVEL_OF_GKS
      (ERROR_INDICATOR       : out ERROR_NUMBER;
       LEVEL                 : out GKS_LEVEL);

procedure INQ_LIST_OF_AVAILABLE_WS_TYPES
      (ERROR_INDICATOR       : out ERROR_NUMBER;
       TYPES                 : out WS_TYPES.LIST_OF);

procedure INQ_WS_MAX_NUMBERS
      (ERROR_INDICATOR       : out ERROR_NUMBER;
       MAX_OPEN_WS           : out POSITIVE;
       MAX_ACTIVE_WS         : out POSITIVE;
       MAX_SEGMENT_WS        : out POSITIVE);

procedure INQ_MAX_NORMALIZATION_TRANSFORMATION_NUMBER
      (ERROR_INDICATOR       : out ERROR_NUMBER;
       TRANSFORMATION        : out TRANSFORMATION_NUMBER);

procedure INQ_SET_OF_OPEN_WS
      (ERROR_INDICATOR       : out ERROR_NUMBER;
       WS                    : out WS_IDS.LIST_OF);

procedure INQ_SET_OF_ACTIVE_WS
      (ERROR_INDICATOR       : out ERROR_NUMBER;
       WS                    : out WS_IDS.LIST_OF);

procedure INQ_CURRENT_PRIMITIVE_ATTRIBUTE_VALUES
      (ERROR_INDICATOR       : out ERROR_NUMBER;
       ATTRIBUTES            : out PRIMITIVE_ATTRIBUTE_VALUES);

procedure INQ_POLYLINE_INDEX
      (ERROR_INDICATOR       : out ERROR_NUMBER;
       INDEX                 : out POLYLINE_INDEX);
```

```
procedure INQ_POLYMARKER_INDEX
      (ERROR_INDICATOR    : out ERROR_NUMBER;
       INDEX              : out POLYMARKER_INDEX);

procedure INQ_TEXT_INDEX
      (ERROR_INDICATOR    : out ERROR_NUMBER;
       INDEX              : out TEXT_INDEX);

procedure INQ_CHAR_HEIGHT
      (ERROR_INDICATOR    : out ERROR_NUMBER;
       HEIGHT             : out WC.MAGNITUDE);

procedure INQ_CHAR_UP_VECTOR
      (ERROR_INDICATOR    : out ERROR_NUMBER;
       VECTOR             : out WC.VECTOR);

procedure INQ_CHAR_WIDTH
      (ERROR_INDICATOR    : out ERROR_NUMBER;
       WIDTH              : out WC.MAGNITUDE);

procedure INQ_CHAR_BASE_VECTOR
      (ERROR_INDICATOR    : out ERROR_NUMBER;
       VECTOR             : out WC.VECTOR);

procedure INQ_TEXT_PATH
      (ERROR_INDICATOR    : out ERROR_NUMBER;
       PATH               : out TEXT_PATH);

procedure INQ_TEXT_ALIGNMENT
      (ERROR_INDICATOR    : out ERROR_NUMBER;
       ALIGNMENT          : out TEXT_ALIGNMENT);

procedure INQ_FILL_AREA_INDEX
      (ERROR_INDICATOR    : out ERROR_NUMBER;
       INDEX              : out FILL_AREA_INDEX);

procedure INQ_PATTERN_WIDTH_VECTOR
      (ERROR_INDICATOR    : out ERROR_NUMBER;
       WIDTH              : out WC.VECTOR);

procedure INQ_PATTERN_HEIGHT_VECTOR
      (ERROR_INDICATOR    : out ERROR_NUMBER;
       VECTOR             : out WC.VECTOR);

procedure INQ_PATTERN_REFERENCE_POINT
      (ERROR_INDICATOR    : out ERROR_NUMBER;
       REFERENCE_POINT    : out WC.POINT);

procedure INQ_CURRENT_PICK_ID_VALUE
      (ERROR_INDICATOR    : out ERROR_NUMBER;
       PICK               : out PICK_ID);

procedure INQ_CURRENT_INDIVIDUAL_ATTRIBUTE_VALUES
      (ERROR_INDICATOR    : out ERROR_NUMBER;
       ATTRIBUTES         : out INDIVIDUAL_ATTRIBUTE_VALUES);
```

```
procedure INQ_LINETYPE
      (ERROR_INDICATOR    : out ERROR_NUMBER;
       TYPE_OF_LINE       : out LINETYPE);

procedure INQ_LINEWIDTH_SCALE_FACTOR
      (ERROR_INDICATOR    : out ERROR_NUMBER;
       WIDTH              : out LINEWIDTH);

procedure INQ_POLYLINE_COLOUR_INDEX
      (ERROR_INDICATOR    : out ERROR_NUMBER;
       LINE_COLOUR        : out COLOUR_INDEX);

procedure INQ_POLYMARKER_TYPE
      (ERROR_INDICATOR    : out ERROR_NUMBER;
       TYPE_OF_MARKER     : out MARKER_TYPE);

procedure INQ_POLYMARKER_SIZE_SCALE_FACTOR
      (ERROR_INDICATOR    :  out ERROR_NUMBER;
       SIZE               : out MARKER_SIZE);

procedure INQ_POLYMARKER_COLOUR_INDEX
      (ERROR_INDICATOR    : out ERROR_NUMBER;
       MARKER_COLOUR      : out COLOUR_INDEX);

procedure INQ_TEXT_FONT_AND_PRECISION
      (ERROR_INDICATOR    : out ERROR_NUMBER;
       FONT_PRECISION     : out TEXT_FONT_PRECISION);

procedure INQ_CHAR_EXPANSION_FACTOR
      (ERROR_INDICATOR    : out ERROR_NUMBER;
       EXPANSION          : out CHAR_EXPANSION);

procedure INQ_CHAR_SPACING
      (ERROR_INDICATOR    : out ERROR_NUMBER;
       SPACING            : out CHAR_SPACING);

procedure INQ_TEXT_COLOUR_INDEX
      (ERROR_INDICATOR    : out ERROR_NUMBER;
       TEXT_COLOUR        : out COLOUR_INDEX);

procedure INQ_FILL_AREA_INTERIOR_STYLE
      (ERROR_INDICATOR    : out ERROR_NUMBER;
       INTERIOR           : out INTERIOR_STYLE);

procedure INQ_FILL_AREA_STYLE_INDEX
      (ERROR_INDICATOR    : out ERROR_NUMBER;
       STYLE              : out STYLE_INDEX);

procedure INQ_FILL_AREA_COLOUR_INDEX
      (ERROR_INDICATOR    : out ERROR_NUMBER;
       FILL_AREA_COLOUR : out COLOUR_INDEX);

procedure INQ_LIST_OF_ASF
      (ERROR_INDICATOR    : out ERROR_NUMBER;
       LIST               : out ASF_LIST);
```

```
procedure INQ_CURRENT_NORMALIZATION_TRANSFORMATION_NUMBER
      (ERROR_INDICATOR    : out ERROR_NUMBER;
       TRANSFORMATION     : out TRANSFORMATION_NUMBER);

procedure INQ_LIST_OF_NORMALIZATION_TRANSFORMATION_NUMBERS
      (ERROR_INDICATOR    : out ERROR_NUMBER;
       LIST               : out TRANSFORMATION_PRIORITY_LIST);

procedure INQ_NORMALIZATION_TRANSFORMATION
      (TRANSFORMATION     : in TRANSFORMATION_NUMBER;
       ERROR_INDICATOR    : out ERROR_NUMBER;
       WINDOW_LIMITS      : out WC.RECTANGLE_LIMITS;
       VIEWPORT_LIMITS    : out NDC.RECTANGLE_LIMITS);

procedure INQ_CLIPPING
      (ERROR_INDICATOR                 : out ERROR_NUMBER;
       CLIPPING                        : out CLIPPING_INDICATOR;
       CLIPPING_RECTANGLE_LIMITS       : out NDC.RECTANGLE_LIMITS);

procedure INQ_NAME_OF_OPEN_SEGMENT
      (ERROR_INDICATOR    : out ERROR_NUMBER;
       SEGMENT            : out SEGMENT_NAME);

procedure INQ_SET_OF_SEGMENT_NAMES_IN_USE
      (ERROR_INDICATOR    : out ERROR_NUMBER;
       SEGMENTS           : out SEGMENT_NAMES.LIST_OF);

procedure INQ_MORE_SIMULTANEOUS_EVENTS
      (ERROR_INDICATOR    : out ERROR_NUMBER;
       EVENTS             : out MORE_EVENTS);

procedure INQ_WS_CONNECTION_AND_TYPE
      (WS                 : in WS_ID;
       ERROR_INDICATOR    : out ERROR_NUMBER;
       CONNECTION         : out VARIABLE_CONNECTION_ID;
       TYPE_OF_WS         : out WS_TYPE);

procedure INQ_WS_STATE
      (WS                 : in WS_ID;
       ERROR_INDICATOR    : out ERROR_NUMBER;
       STATE              : out WS_STATE);

procedure INQ_WS_DEFERRAL_AND_UPDATE_STATES
      (WS                 : in WS_ID;
       ERROR_INDICATOR    : out ERROR_NUMBER;
       DEFERRAL           : out DEFERRAL_MODE;
       REGENERATION       : out REGENERATION_MODE;
       DISPLAY            : out DISPLAY_SURFACE_EMPTY;
       FRAME_ACTION       : out NEW_FRAME_NECESSARY);

procedure INQ_LIST_OF_POLYLINE_INDICES
      (WS                 : in WS_ID;
       ERROR_INDICATOR    : out ERROR_NUMBER;
       INDICES            : out POLYLINE_INDICES.LIST_OF);
```

```
procedure INQ_POLYLINE_REPRESENTATION
      (WS                  : in WS_ID;
       INDEX               : in POLYLINE_INDEX;
       RETURNED_VALUES     : in RETURN_VALUE_TYPE;
       ERROR_INDICATOR     : out ERROR_NUMBER;
       TYPE_OF_LINE        : out LINETYPE;
       WIDTH               : out LINEWIDTH;
       LINE_COLOUR         : out COLOUR_INDEX);

procedure INQ_LIST_OF_POLYMARKER_INDICES
      (WS                            : in WS_ID;
       ERROR_INDICATOR               : out ERROR_NUMBER;
       INDICES                       : out POLYMARKER_INDICES.LIST_OF);

procedure INQ_POLYMARKER_REPRESENTATION
      (WS                            : in WS_ID;
       INDEX                         : in POLYMARKER_INDEX;
       RETURNED_VALUES               : in RETURN_VALUE_TYPE;
       ERROR_INDICATOR               : out ERROR_NUMBER;
       TYPE_OF_MARKER                : out MARKER_TYPE;
       SIZE                          : out MARKER_SIZE;
       MARKER_COLOUR                 : out COLOUR_INDEX);

procedure INQ_LIST_OF_TEXT_INDICES
      (WS                            : in WS_ID;
       ERROR_INDICATOR               : out ERROR_NUMBER;
       INDICES                       : out TEXT_INDICES.LIST_OF);

procedure INQ_TEXT_REPRESENTATION
      (WS                             : in WS_ID;
       INDEX                          : in TEXT_INDEX;
       RETURNED_VALUES                : in RETURN_VALUE_TYPE;
       ERROR_INDICATOR                : out ERROR_NUMBER;
       FONT_PRECISION                 : out TEXT_FONT_PRECISION;
       EXPANSION                      : out CHAR_EXPANSION;
       SPACING                        : out CHAR_SPACING;
       TEXT_COLOUR                    : out COLOUR_INDEX);

procedure INQ_TEXT_EXTENT
      (WS                            : in WS_ID;
       POSITION                      : in WC.POINT;
       CHAR_STRING                   : in STRING;
       ERROR_INDICATOR               : out ERROR_NUMBER;
       CONCATENATION_POINT           : out WC.POINT;
       TEXT_EXTENT                   : out TEXT_EXTENT_PARALLELOGRAM);

procedure INQ_LIST_OF_FILL_AREA_INDICES
      (WS                            : in WS_ID;
       ERROR_INDICATOR               : out ERROR_NUMBER;
       INDICES                       : out FILL_AREA_INDICES.LIST_OF);
```

```
procedure INQ_FILL_AREA_REPRESENTATION
      (WS                          : in WS_ID;
       INDEX                       : in FILL_AREA_INDEX;
       RETURNED_VALUES             : in RETURN_VALUE_TYPE;
       ERROR_INDICATOR             : out ERROR_NUMBER;
       INTERIOR                    : out INTERIOR_STYLE;
       STYLE                       : out STYLE_INDEX;
       FILL_AREA_COLOUR            : out COLOUR_INDEX);

procedure INQ_LIST_OF_PATTERN_INDICES
      (WS                          : in WS_ID;
       ERROR_INDICATOR             : out ERROR_NUMBER;
       INDICES                     : out PATTERN_INDICES.LIST_OF);

procedure INQ_PATTERN_REPRESENTATION
      (WS                          : in WS_ID;
       INDEX                       : in PATTERN_INDEX;
       RETURNED_VALUES             : in RETURN_VALUE_TYPE;
       ERROR_INDICATOR             : out ERROR_NUMBER;
       PATTERN                     : out VARIABLE_COLOUR_MATRIX);

procedure INQ_LIST_OF_COLOUR_INDICES
      (WS                          : in WS_ID;
       ERROR_INDICATOR             : out ERROR_NUMBER;
       INDICES                     : out COLOUR_INDICES.LIST_OF);

procedure INQ_COLOUR_REPRESENTATION
      (WS                          : in WS_ID;
       INDEX                       : in COLOUR_INDEX;
       RETURNED_VALUES             : in RETURN_VALUE_TYPE;
       ERROR_INDICATOR             : out ERROR_NUMBER;
       RGB_COLOUR                  : out COLOUR_REPRESENTATION);

procedure INQ_WS_TRANSFORMATION
      (WS                          : in WS_ID;
       ERROR_INDICATOR             : out ERROR_NUMBER;
       UPDATE                      : out UPDATE_STATE;
       REQUESTED_WINDOW            : out NDC.RECTANGLE_LIMITS;
       CURRENT_WINDOW              : out NDC.RECTANGLE_LIMITS;
       REQUESTED_VIEWPORT          : out DC.RECTANGLE_LIMITS;
       CURRENT_VIEWPORT            : out DC.RECTANGLE_LIMITS);

procedure INQ_SET_OF_SEGMENT_NAMES_ON_WS
      (WS                          : in WS_ID;
       ERROR_INDICATOR             : out ERROR_NUMBER;
       SEGMENTS                    : out SEGMENT_NAMES.LIST_OF);

procedure INQ_LOCATOR_DEVICE_STATE
      (WS                                : in WS_ID;
       DEVICE                            : in LOCATOR_DEVICE_NUMBER;
       RETURNED_VALUES                   : in RETURN_VALUE_TYPE;
       ERROR_INDICATOR                   : out ERROR_NUMBER;
       MODE                              : out OPERATING_MODE;
       SWITCH                            : out ECHO_SWITCH;
       INITIAL_TRANSFORMATION            : out TRANSFORMATION_NUMBER;
       INITIAL_POSITION                  : out WC.POINT;
       ECHO_AREA                         : out DC.RECTANGLE_LIMITS;
       DATA_RECORD                       : out LOCATOR_DATA_RECORD);
```

```
procedure INQ_STROKE_DEVICE_STATE
      (WS                          : in WS_ID;
       DEVICE                      : in STROKE_DEVICE_NUMBER;
       RETURNED_VALUES             : in RETURN_VALUE_TYPE;
       ERROR_INDICATOR             : out ERROR_NUMBER;
       MODE                        : out OPERATING_MODE;
       SWITCH                      : out ECHO_SWITCH;
       INITIAL_TRANSFORMATION      : out TRANSFORMATION_NUMBER;
       INITIAL_STROKE_POINTS       : out WC.POINT_LIST;
       ECHO_AREA                   : out DC.RECTANGLE_LIMITS;
       DATA_RECORD                 : out STROKE_DATA_RECORD);

procedure INQ_VALUATOR_DEVICE_STATE
      (WS                          : in WS_ID;
       DEVICE                      : in VALUATOR_DEVICE_NUMBER;
       ERROR_INDICATOR             : out ERROR_NUMBER;
       MODE                        : out OPERATING_MODE;
       SWITCH                      : out ECHO_SWITCH;
       INITIAL_VALUE               : out VALUATOR_INPUT_VALUE;
       ECHO_AREA                   : out DC.RECTANGLE_LIMITS;
       DATA_RECORD                 : out VALUATOR_DATA_RECORD);

procedure INQ_CHOICE_DEVICE_STATE
      (WS                          : in WS_ID;
       DEVICE                      : in CHOICE_DEVICE_NUMBER;
       ERROR_INDICATOR             : out ERROR_NUMBER;
       MODE                        : out OPERATING_MODE;
       SWITCH                      : out ECHO_SWITCH;
       INITIAL_STATUS              : out CHOICE_STATUS;
       INITIAL_CHOICE              : out CHOICE_VALUE;
       ECHO_AREA                   : out DC.RECTANGLE_LIMITS;
       DATA_RECORD                 : out CHOICE_DATA_RECORD);

procedure INQ_PICK_DEVICE_STATE
      (WS                          : in WS_ID;
       DEVICE                      : in PICK_DEVICE_NUMBER;
       RETURNED_VALUES             : in RETURN_VALUE_TYPE;
       ERROR_INDICATOR             : out ERROR_NUMBER;
       MODE                        : out OPERATING_MODE;
       SWITCH                      : out ECHO_SWITCH;
       INITIAL_STATUS              : out PICK_STATUS;
       INITIAL_SEGMENT             : out SEGMENT_NAME;
       INITIAL_PICK                : out PICK_ID;
       ECHO_AREA                   : out DC.RECTANGLE_LIMITS;
       DATA_RECORD                 : out PICK_DATA_RECORD);

procedure INQ_STRING_DEVICE_STATE
      (WS                          : in WS_ID;
       DEVICE                      : in STRING_DEVICE_NUMBER;
       ERROR_INDICATOR             : out ERROR_NUMBER;
       MODE                        : out OPERATING_MODE;
       SWITCH                      : out ECHO_SWITCH;
       INITIAL_STRING              : out INPUT_STRING;
       ECHO_AREA                   : out DC.RECTANGLE_LIMITS;
       DATA_RECORD                 : out STRING_DATA_RECORD);
```

```
procedure INQ_WS_CATEGORY
      (TYPE_OF_WS                    : in WS_TYPE;
       ERROR_INDICATOR               : out ERROR_NUMBER;
       CATEGORY                      : out WS_CATEGORY);

procedure INQ_WS_CLASSIFICATION
      (TYPE_OF_WS                    : in WS_TYPE;
       ERROR_INDICATOR               : out ERROR_NUMBER;
       CLASS                         : out DISPLAY_CLASS);

procedure INQ_DISPLAY_SPACE_SIZE
      (TYPE_OF_WS                    : in WS_TYPE;
       ERROR_INDICATOR               : out ERROR_NUMBER;
       UNITS                         : out DC_UNITS;
       MAX_DC_SIZE                   : out DC.SIZE;
       MAX_RASTER_UNIT_SIZE          : out RASTER_UNIT_SIZE);

procedure INQ_DYNAMIC_MODIFICATION_OF_WS_ATTRIBUTES
      (TYPE_OF_WS                              : in WS_TYPE;
       ERROR_INDICATOR                         : out ERROR_NUMBER;
       POLYLINE_REPRESENTATION                 : out DYNAMIC_MODIFICATION;
       POLYMARKER_REPRESENTATION               : out DYNAMIC_MODIFICATION;
       TEXT_REPRESENTATION                     : out DYNAMIC_MODIFICATION;
       FILL_AREA_REPRESENTATION                : out DYNAMIC_MODIFICATION;
       PATTERN_REPRESENTATION                  : out DYNAMIC_MODIFICATION;
       COLOUR_REPRESENTATION                   : out DYNAMIC_MODIFICATION;
       TRANSFORMATION                          : out DYNAMIC_MODIFICATION);

procedure INQ_DEFAULT_DEFERRAL_STATE_VALUES
      (TYPE_OF_WS                    : in WS_TYPE;
       ERROR_INDICATOR               : out ERROR_NUMBER;
       DEFERRAL                      : out DEFERRAL_MODE;
       REGENERATION                  : out REGENERATION_MODE);

procedure INQ_POLYLINE_FACILITIES
      (TYPE_OF_WS                    : in WS_TYPE;
       ERROR_INDICATOR               : out ERROR_NUMBER;
       LIST_OF_TYPES                 : out LINETYPES.LIST_OF;
       NUMBER_OF_WIDTHS              : out NATURAL;
       NOMINAL_WIDTH                 : out DC.MAGNITUDE;
       RANGE_OF_WIDTHS               : out DC.RANGE_OF_MAGNITUDES;
       NUMBER_OF_INDICES             : out NATURAL);

procedure INQ_PREDEFINED_POLYLINE_REPRESENTATION
      (TYPE_OF_WS                    : in WS_TYPE;
       INDEX                         : in POLYLINE_INDEX;
       ERROR_INDICATOR               : out ERROR_NUMBER;
       TYPE_OF_LINE                  : out LINETYPE;
       WIDTH                         : out LINEWIDTH;
       LINE_COLOUR                   : out COLOUR_INDEX);
```

```
procedure INQ_POLYMARKER_FACILITIES
      (TYPE_OF_WS                  : in WS_TYPE;
       ERROR_INDICATOR             : out ERROR_NUMBER;
       LIST_OF_TYPES               : out MARKER_TYPES.LIST_OF;
       NUMBER_OF_SIZES             : out NATURAL;
       NOMINAL_SIZE                : out DC.MAGNITUDE;
       RANGE_OF_SIZES              : out DC.RANGE_OF_MAGNITUDES;
       NUMBER_OF_INDICES           : out NATURAL);

procedure INQ_PREDEFINED_POLYMARKER_REPRESENTATION
      (TYPE_OF_WS                  : in WS_TYPE;
       INDEX                       : in POLYMARKER_INDEX;
       ERROR_INDICATOR             : out ERROR_NUMBER;
       TYPE_OF_MARKER              : out MARKER_TYPE;
       SIZE                        : out MARKER_SIZE;
       MARKER_COLOUR               : out COLOUR_INDEX);

procedure INQ_TEXT_FACILITIES
      (TYPE_OF_WS                  : in WS_TYPE;
       ERROR_INDICATOR             : out ERROR_NUMBER;
       LIST_OF_FONT_PRECISION_PAIRS: out
                                     TEXT_FONT_PRECISIONS.LIST_OF;
       NUMBER_OF_HEIGHTS           : out NATURAL;
       RANGE_OF_HEIGHTS            : out DC.RANGE_OF_MAGNITUDES;
       NUMBER_OF_EXPANSIONS        : out NATURAL;
       EXPANSION_RANGE             : out RANGE_OF_EXPANSIONS;
       NUMBER_OF_INDICES           : out NATURAL);

procedure INQ_PREDEFINED_TEXT_REPRESENTATION
      (TYPE_OF_WS                  : in WS_TYPE;
       INDEX                       : in TEXT_INDEX;
       ERROR_INDICATOR             : out ERROR_NUMBER;
       FONT_PRECISION              : out TEXT_FONT_PRECISION;
       EXPANSION                   : out CHAR_EXPANSION;
       SPACING                     : out CHAR_SPACING;
       TEXT_COLOUR                 : out COLOUR_INDEX);

procedure INQ_FILL_AREA_FACILITIES
      (TYPE_OF_WS                  : in WS_TYPE;
       ERROR_INDICATOR             : out ERROR_NUMBER;
       LIST_OF_INTERIOR_STYLES     : out INTERIOR_STYLES.LIST_OF;
       LIST_OF_HATCH_STYLES        : out HATCH_STYLES.LIST_OF;
       NUMBER_OF_INDICES           : out NATURAL);

procedure INQ_PREDEFINED_FILL_AREA_REPRESENTATION
      (TYPE_OF_WS                  : in WS_TYPE;
       INDEX                       : in FILL_AREA_INDEX;
       ERROR_INDICATOR             : out ERROR_NUMBER;
       INTERIOR                    : out INTERIOR_STYLE;
       STYLE                       : out STYLE_INDEX;
       FILL_AREA_COLOUR            : out COLOUR_INDEX);

procedure INQ_PATTERN_FACILITIES
      (TYPE_OF_WS                  : in WS_TYPE;
       ERROR_INDICATOR             : out ERROR_NUMBER;
       NUMBER_OF_INDICES           : out NATURAL);
```

```
procedure INQ_PREDEFINED_PATTERN_REPRESENTATION
      (TYPE_OF_WS                 : in WS_TYPE;
       INDEX                      : in PATTERN_INDEX;
       ERROR_INDICATOR            : out ERROR_NUMBER;
       PATTERN                    : out VARIABLE_COLOUR_MATRIX);

procedure INQ_COLOUR_FACILITIES
      (TYPE_OF_WS                      : in WS_TYPE;
       ERROR_INDICATOR                 : out ERROR_NUMBER;
       NUMBER_OF_COLOURS               : out NATURAL;
       AVAILABLE_COLOUR                : out COLOUR_AVAILABLE;
       NUMBER_OF_COLOUR_INDICES        : out NATURAL);

procedure INQ_PREDEFINED_COLOUR_REPRESENTATION
      (TYPE_OF_WS                 : in WS_TYPE;
       INDEX                      : in COLOUR_INDEX;
       ERROR_INDICATOR            : out ERROR_NUMBER;
       RGB_COLOUR                 : out COLOUR_REPRESENTATION);

procedure INQ_LIST_OF_AVAILABLE_GDP
      (TYPE_OF_WS                 : in WS_TYPE;
       ERROR_INDICATOR            : out ERROR_NUMBER;
       LIST_OF_GDP                : out GDP_IDS.LIST_OF);

procedure INQ_GDP
      (TYPE_OF_WS                 : in WS_TYPE;
       GDP                        : in GDP_ID;
       ERROR_INDICATOR            : out ERROR_NUMBER;
       LIST_OF_ATTRIBUTES_USED    : out ATTRIBUTES_USED.LIST_OF);

procedure INQ_MAX_LENGTH_OF_WS_STATE_TABLES
      (TYPE_OF_WS                      : in WS_TYPE;
       ERROR_INDICATOR                 : out ERROR_NUMBER;
       MAX_POLYLINE_ENTRIES            : out NATURAL;
       MAX_POLYMARKER_ENTRIES          : out NATURAL;
       MAX_TEXT_ENTRIES                : out NATURAL;
       MAX_FILL_AREA_ENTRIES           : out NATURAL;
       MAX_PATTERN_INDICES             : out NATURAL;
       MAX_COLOUR_INDICES              : out NATURAL);

procedure INQ_NUMBER_OF_SEGMENT_PRIORITIES_SUPPORTED
      (TYPE_OF_WS                 : in WS_TYPE;
       ERROR_INDICATOR            : out ERROR_NUMBER;
       NUMBER_OF_PRIORITIES       : out NATURAL);

procedure INQ_DYNAMIC_MODIFICATION_OF_SEGMENT_ATTRIBUTES
      (TYPE_OF_WS                 : in WS_TYPE;
       ERROR_INDICATOR            : out ERROR_NUMBER;
       TRANSFORMATION             : out DYNAMIC_MODIFICATION;
       VISIBLE_TO_INVISIBLE       : out DYNAMIC_MODIFICATION;
       INVISIBLE_TO_VISIBLE       : out DYNAMIC_MODIFICATION;
       HIGHLIGHTING               : out DYNAMIC_MODIFICATION;
       PRIORITY                   : out DYNAMIC_MODIFICATION;
       ADDING_PRIMITIVES          : out DYNAMIC_MODIFICATION;
       DELETION_VISIBLE           : out DYNAMIC_MODIFICATION);
```

```
procedure INQ_NUMBER_OF_AVAILABLE_LOGICAL_INPUT_DEVICES
      (TYPE_OF_WS                    : in WS_TYPE;
       ERROR_INDICATOR               : out ERROR_NUMBER;
       LOCATOR                       : out NATURAL;
       STROKE                        : out NATURAL;
       VALUATOR                      : out NATURAL;
       CHOICE                        : out NATURAL;
       PICK                          : out NATURAL;
       STRING                        : out NATURAL);

procedure INQ_DEFAULT_LOCATOR_DEVICE_DATA
      (TYPE_OF_WS                    : in WS_TYPE;
       DEVICE                        : in LOCATOR_DEVICE_NUMBER;
       ERROR_INDICATOR               : out ERROR_NUMBER;
       INITIAL_POSITION              : out WC.POINT;
       LIST_OF_PROMPT_ECHO_TYPES :   : out
                                       LOCATOR_PROMPT_ECHO_TYPES.LIST_OF;
      .ECHO_AREA                     : out DC.RECTANGLE_LIMITS;
       DATA_RECORD                   : out LOCATOR_DATA_RECORD);

procedure INQ_DEFAULT_STROKE_DEVICE_DATA
      (TYPE_OF_WS                    : in WS_TYPE;
       DEVICE                        : in STROKE_DEVICE_NUMBER;
       ERROR_INDICATOR               : out ERROR_NUMBER;
       MAX_BUFFER_SIZE               : out NATURAL;
       LIST_OF_PROMPT_ECHO_TYPES     : out
                                       STROKE_PROMPT_ECHO_TYPES.LIST_OF;
       ECHO_AREA                     : out DC.RECTANGLE_LIMITS;
       DATA_RECORD                   : out STROKE_DATA_RECORD);

procedure INQ_DEFAULT_VALUATOR_DEVICE_DATA
      (TYPE_OF_WS                    : in WS_TYPE;
       DEVICE                        : in VALUATOR_DEVICE_NUMBER;
       ERROR_INDICATOR               : out ERROR_NUMBER;
       INITIAL_VALUE                 : out VALUATOR_INPUT_VALUE;
       LIST_OF_PROMPT_ECHO_TYPES     : out
                                       VALUATOR_PROMPT_ECHO_TYPES.LIST_OF;
       ECHO_AREA                     : out DC.RECTANGLE_LIMITS;
       DATA_RECORD                   : out VALUATOR_DATA_RECORD);

procedure INQ_DEFAULT_CHOICE_DEVICE_DATA
      (TYPE_OF_WS                    : in WS_TYPE;
       DEVICE                        : in CHOICE_DEVICE_NUMBER;
       ERROR_INDICATOR               : out ERROR_NUMBER;
       MAX_CHOICES                   : out CHOICE_VALUE;
       LIST_OF_PROMPT_ECHO_TYPES     : out
                                       CHOICE_PROMPT_ECHO_TYPES.LIST_OF;
       ECHO_AREA                     : out DC.RECTANGLE_LIMITS;
       DATA_RECORD                   : out CHOICE_DATA_RECORD);

procedure INQ_DEFAULT_PICK_DEVICE_DATA
      (TYPE_OF_WS                    : in WS_TYPE;
       DEVICE                        : in PICK_DEVICE_NUMBER;
       ERROR_INDICATOR               : out ERROR_NUMBER;
       LIST_OF_PROMPT_ECHO_TYPES     : out PICK_PROMPT_ECHO_TYPES.LIST_OF;
       ECHO_AREA                     : out DC.RECTANGLE_LIMITS;
       DATA_RECORD                   : out PICK_DATA_RECORD);
```

```
procedure INQ_DEFAULT_STRING_DEVICE_DATA
      (TYPE_OF_WS                    : in WS_TYPE;
       DEVICE                        : in STRING_DEVICE_NUMBER;
       ERROR_INDICATOR               : out ERROR_NUMBER;
       MAX_STRING_BUFFER_SIZE        : out NATURAL;
       LIST_OF_PROMPT_ECHO_TYPES     : out
                                       STRING_PROMPT_ECHO_TYPES.LIST_OF;
       ECHO_AREA                     : out DC.RECTANGLE_LIMITS;
       DATA_RECORD                   : out STRING_DATA_RECORD);

procedure INQ_SET_OF_ASSOCIATED_WS
      (SEGMENT                       : in SEGMENT_NAME;
       ERROR_INDICATOR               : out ERROR_NUMBER;
       LIST_OF_WS                    : out WS_IDS.LIST_OF);

procedure INQ_SEGMENT_ATTRIBUTES
      (SEGMENT                       : in SEGMENT_NAME;
       ERROR_INDICATOR               : out ERROR_NUMBER;
       TRANSFORMATION                : out TRANSFORMATION_MATRIX;
       VISIBILITY                    : out SEGMENT_VISIBILITY;
       HIGHLIGHTING                  : out SEGMENT_HIGHLIGHTING;
       PRIORITY                      : out SEGMENT_PRIORITY;
       DETECTABILITY                 : out SEGMENT_DETECTABILITY);

procedure INQ_PIXEL_ARRAY_DIMENSIONS
      (WS                            : in WS_ID;
       CORNER_1_1                    : in WC.POINT;
       CORNER_DX_DY                  : in WC.POINT;
       ERROR_INDICATOR               : out ERROR_NUMBER;
       DIMENSIONS                    : out RASTER_UNIT_SIZE);

procedure INQ_PIXEL_ARRAY
      (WS                            : in WS_ID;
       CORNER                        : in WC.POINT;
       DX                            : in RASTER_UNITS;
       DY                            : in RASTER_UNITS;
       ERROR_INDICATOR               : out ERROR_NUMBER;
       INVALID_VALUES                : out INVALID_VALUES_INDICATOR;
       PIXEL_ARRAY                   : out VARIABLE_PIXEL_COLOUR_MATRIX);

procedure INQ_PIXEL
      (WS                            : in WS_ID;
       POINT                         : in WC.POINT;
       ERROR_INDICATOR               : out ERROR_NUMBER;
       PIXEL_COLOUR                  : out PIXEL_COLOUR_INDEX);

procedure INQ_INPUT_QUEUE_OVERFLOW
      (ERROR_INDICATOR               : out ERROR_NUMBER;
       WS                            : out WS_ID;
       CLASS                         : out INPUT_QUEUE_CLASS;
       DEVICE                        : out EVENT_OVERFLOW_DEVICE_NUMBER);
```

```
   -- UTILITY FUNCTIONS

   procedure EVALUATE_TRANSFORMATION_MATRIX
         (FIXED_POINT                  : in WC.POINT;
          SHIFT_VECTOR                 : in WC.VECTOR;
          ROTATION_ANGLE               : in RADIANS;
          SCALE_FACTORS                : in TRANSFORMATION_FACTOR;
          TRANSFORMATION               : out TRANSFORMATION_MATRIX);

   procedure EVALUATE_TRANSFORMATION_MATRIX
         (FIXED_POINT                  : in NDC.POINT;
          SHIFT_VECTOR                 : in NDC.VECTOR;
          ROTATION_ANGLE               : in RADIANS;
          SCALE_FACTORS                : in TRANSFORMATION_FACTOR;
          TRANSFORMATION               : out TRANSFORMATION_MATRIX);

   procedure ACCUMULATE_TRANSFORMATION_MATRIX
         (SOURCE_TRANSFORMATION        : in TRANSFORMATION_MATRIX;
          FIXED_POINT                  : in WC.POINT;
          SHIFT_VECTOR                 : in WC.VECTOR;
          ROTATION_ANGLE               : in RADIANS;
          SCALE_FACTORS                : in TRANSFORMATION_FACTOR;
          RESULT_TRANSFORMATION        : out TRANSFORMATION_MATRIX);

   procedure ACCUMULATE_TRANSFORMATION_MATRIX
         (SOURCE_TRANSFORMATION        : in TRANSFORMATION_MATRIX;
          FIXED_POINT                  : in NDC.POINT;
          SHIFT_VECTOR                 : in NDC.VECTOR;
          ROTATION_ANGLE               : in RADIANS;
          SCALE_FACTORS                : in TRANSFORMATION_FACTOR;
          RESULT_TRANSFORMATION        : out TRANSFORMATION_MATRIX);

-- ERROR FUNCTIONS

   procedure ERROR_LOGGING
         (ERROR_INDICATOR              : in ERROR_NUMBER;
          GKS_FUNCTION                 : in STRING;
          ERROR_FILE                   : in STRING := DEFAULT_ERROR_FILE);

   procedure EMERGENCY_CLOSE_GKS;

-- Metafile function utilities

   -- Item data records may contain lists of points, character strings, arrays of colour indices,
   -- and GDP and ESC data. Record length depends on the number of data elements. GKS
   -- defines that the format is implementation defined.

   -- The item data record type should be private to allow direct manipulation of the record
   -- contents in order to have them efficiently processed.

   -- The application programmer must be able to write non-graphical data into the metafile.
   -- This can be provided by allowing character strings to be output. Numeric data must be
   -- converted to a string by the application programmer prior to calling
   -- BUILD_NEW_GKSM_DATA_RECORD. A function is provided as a means to
   -- convert item data records into strings.
```

```
   procedure BUILD_NEW_GKSM_DATA_RECORD
         (TYPE_OF_ITEM                : in GKSM_ITEM_TYPE;
          ITEM_DATA                   : in STRING;
          ITEM                        : out GKSM_DATA_RECORD);

   function ITEM_DATA_RECORD_STRING
         (ITEM : in GKSM_DATA_RECORD)  return STRING;
private

   -- The following types define the specifications for the private data records.

   type GKSM_DATA_RECORD  (TYPE_OF_ITEM          : GKSM_ITEM_TYPE := 0;
                           LENGTH                : NATURAL        := 0) is
         record
           null;
         end record;

   type CHOICE_DATA_RECORD (PROMPT_ECHO_TYPE:
                            CHOICE_PROMPT_ECHO_TYPE := DEFAULT_CHOICE is

         record
           null;
         end record;

   type LOCATOR_DATA_RECORD (PROMPT_ECHO_TYPE:
                             LOCATOR_PROMPT_ECHO_TYPE := DEFAULT_LOCATOR) is
         record
           null;
         end record;

   type STRING_DATA_RECORD (PROMPT_ECHO_TYPE:
                            STRING_PROMPT_ECHO_TYPE := DEFAULT_STRING) is
         record
           null;
         end record;

   type STROKE_DATA_RECORD (PROMPT_ECHO_TYPE:
                            STROKE_PROMPT_ECHO_TYPE := DEFAULT_STROKE) is
         record
           null;
         end record;

   type VALUATOR_DATA_RECORD (PROMPT_ECHO_TYPE:
                              VALUATOR_PROMPT_ECHO_TYPE := DEFAULT_VALUATOR) is
         record
           null;
         end record;

   type PICK_DATA_RECORD (PROMPT_ECHO_TYPE:
                          PICK_PROMPT_ECHO_TYPE := DEFAULT_PICK) is
         record
           null;
         end record;

end GKS;
```

```
-- ERROR HANDLING FUNCTION

   -- The ERROR HANDLING FUNCTION is a separate library unit;
   -- and not compiled as a part of package GKS.

   procedure ERROR_HANDLING
         (ERROR_INDICATOR          : in ERROR_NUMBER;
          GKS_FUNCTION             : in STRING;
          ERROR_FILE               : in STRING := DEFAULT_ERROR_FILE);
-------------------------------------------------------------------------------------------

with GKS_TYPES;
use GKS_TYPES;

package GKS_GDP is

-- The GDP package is a separate library unit, and not compiled as a part of GKS.

-- The Generalized Drawing Primitive (GDP) is bound in a one-to-many fashion, with a
--  separate procedure implemented for each GDP, each with its own parameter interface.
-- GDP names and parameters are registered in the ISO International Register of Graphical
-- Items which is maintained by the Registration Authority.

-- Each unregistered GDP procedure, supported by an implementation will be in a separate
-- library package using the following naming convention:

--         package GKS_UGDP_<name of the GDP procedure> is
--         procedure GDP;
--          -- Ada code for UGDP procedure
--         end GKS_UGDP_<name of the GDP procedure>;
--         -- The only procedure name used in the package will be GDP

-- In order to support the ability to write a GDP that is not implemented at a given
-- implementation to a metafile these registered GDPS, may be invoked using the data
-- types and the form of the procedure GENERALIZED_GDP which have the specifications
-- given below:

        type GDP_FLOAT is digits PRECISION;
        type GDP_INTEGER_ARRAY is array (SMALL_NATURAL range <>)
              of INTEGER;
        type GDP_FLOAT_ARRAY is array (SMALL_NATURAL range <>)
              of GDP_FLOAT;
        type GDP_STRING_ARRAY is array (SMALL_NATURAL range <>)
              of STRING (1..80);

        type GDP_DATA_RECORD     (NUM_OF_INTEGERS     : SMALL_NATURAL := 0;
                                  NUM_OF_REALS        : SMALL_NATURAL := 0;
                                  NUM_OF_STRINGS      : SMALL_NATURAL := 0) is
         record
            INTEGER_ARRAY      : GDP_INTEGER_ARRAY (1..NUM_OF_INTEGERS);
            REAL_ARRAY         : GDP_FLOAT_ARRAY    (1..NUM_OF_REALS);
            GDP_STRINGS        : GDP_STRING_ARRAY   (1..NUM_OF_STRINGS);
         end record;

        procedure GENERALIZED_GDP (GDP_NAME          : in GDP_ID;
                                   POINT             : in WC.POINT_LIST;
                                   GDP_DATA          : out GDP_DATA_RECORD);
         end GKS__GDP;
```

```
with GKS_TYPES;
use GKS_TYPES;

package GKS_ESCAPE is

-- The ESCAPE package is a separate library unit, and not compiled as a part of GKS.

-- Escape functions are bound in Ada as separate procedures for each unique type of escape
-- provided by the implementation, each with a formal parameter list appropriate to the
-- procedure implemented. The registered ESCAPE procedures will be in a library package
-- named GKS_ESCAPE. ESCAPE names and parameters are registered in the ISO
-- International Register of Graphical Items which is maintained by the Registration Authority.

-- Each unregistered ESCAPE procedure will be a library package using the following naming
-- convention:

--      package GKS_UESC_<name of the escape procedure> is
--      procedure ESC;
--      -- Ada code for UESC procedure
--      end GKS_UESC_<name of the escape procedure>;
--      -- the only procedure name used in the package will be ESC

--
-- In order to support the ability to write an ESCAPE that is not implemented at a given
-- implementation to a metafile these registered ESCAPES may be invoked using the
-- data types and the form of the procedure GENERALIZED_ESC which have the specifications
-- given below:

        type ESCAPE_ID is new INTEGER;
        type ESCAPE_FLOAT is digits PRECISION;
        type ESC_INTEGER_ARRAY is array (SMALL_NATURAL range <>)
                of INTEGER;
        type ESC_FLOAT_ARRAY is array   (SMALL_NATURAL range <>)
                of ESCAPE_FLOAT;
        type ESC_STRING_ARRAY is array  (SMALL_NATURAL range <>)
                of STRING (1..80);

        type ESC_DATA_RECORD (NUM_OF_INTEGERS       : SMALL_NATURAL := 0;
                              NUM_OF_REALS          : SMALL_NATURAL := 0;
                              NUM_OF_STRINGS        : SMALL_NATURAL := 0) is
        record
            INTEGER_ARRAY     : ESC_INTEGER_ARRAY     (1..NUM_OF_INTEGERS);
            REAL_ARRAY        : ESC_FLOAT_ARRAY       (1..NUM_OF_REALS);
            ESC_STRINGS       : ESC_STRING_ARRAY      (1..NUM_OF_STRINGS);
        end record;

        procedure GENERALIZED_ESC  (ESCAPE_NAM   : in ESCAPE_ID;
                                    ESC_DATA_IN      : in ESC_DATA_RECORD;
                                    ESC_DATA_OUT     : out ESC_DATA_RECORD);
        end GKS_ESCAPE;
```

Annex B

Cross Reference Listing of Implementation Defined Items

(This annex does not form an integral part of this standard but provides additional information.)

ITEM	SECTION
CHOICE_DATA_RECORD	4.2.3
LOCATOR_DATA_RECORD	4.2.3
PICK_DATA_RECORD	4.2.3
STRING_DATA_RECORD	4.2.3
STROKE_DATA_RECORD	4.2.3
VALUATOR_DATA_RECORD	4.2.3
DEFAULT_MEMORY_UNITS	4.2.4
DEFAULT_ERROR_FILE	4.2.4
PRECISION	4.2.4
MAX_LIST_SIZE	5.2.3
SMALL_NATURAL_MAX	4.2.4
CHOICE_SMALL_NATURAL_MAX	4.2.4
STRING_SMALL_NATURAL_MAX	4.2.4