American National Standard

for Information Systems –

Computer Graphics – Graphical Kernel System (GKS) Ada Binding

ADOPTED FOR USE BY THE FEDERAL GOVERNMENT

FIPS

PUB 120-1

SEE NOTICE ON INSIDE

ANSI X3.124.3-1989

ANSI American National Standards Institute
1430 Broadway
New York , New York
10018

This standard has been adopted for Federal Government use.

Details concerning its use within the Federal Government are contained in Federal Information Processing Standards Publication 120-1, Graphical Kernel System (GKS). For a complete list of the publications available in the Federal Information Processing Standards Series, write to the Standards Processing Coordinator (ADP), National Institute of Standards and Technology, Gaithersburg, MD 20899.

ANSI®
X3.124.3-1989

American National Standard
for Information Systems –

Computer Graphics – Graphical Kernel System (GKS) Ada Binding

Secretariat

Computer and Business Equipment Manufacturers Association

Approved June 30, 1989

American National Standards Institute, Inc

Abstract

This standard provides the Ada language syntax for American National Standard for Information Systems – Computer Graphics – Graphical Kernel System (GKS) Functional Description, ANSI X3.124-1985. The Ada language binding of GKS is a syntactic specification, presented as a set of procedures and/or functions that, taken as a whole, provide the semantics of GKS for use by an Ada application program.

For each GKS function, the Ada procedure name, argument list, and argument data types are given. In addition, any special errors associated only with the Ada language binding of GKS are specified and assigned unique error numbers. The data type definitions used to define the Ada binding to GKS are defined in a separate package, package GKS_TYPES; the GKS procedures and/or functions are defined in a separate package, package GKS.

American National Standard

Approval of an American National Standard requires verification by ANSI that the requirements for due process, consensus, and other criteria for approval have been met by the standards developer.

Consensus is established when, in the judgment of the ANSI Board of Standards Review, substantial agreement has been reached by directly and materially affected interests. Substantial agreement means much more than a simple majority, but not necessarily unanimity. Consensus requires that all views and objections be considered, and that a concerted effort be made toward their resolution.

The use of American National Standards is completely voluntary; their existence does not in any respect preclude anyone, whether he has approved the standards or not, from manufacturing, marketing, purchasing, or using products, processes, or procedures not conforming to the standards.

The American National Standards Institute does not develop standards and will in no circumstances give an interpretation of any American National Standard. Moreover, no person shall have the right or authority to issue an interpretation of an American National Standard in the name of the American National Standards Institute. Requests for interpretations should be addressed to the secretariat or sponsor whose name appears on the title page of this standard.

CAUTION NOTICE: This American National Standard may be revised or withdrawn at any time. The procedures of the American National Standards Institute require that action be taken periodically to reaffirm, revise, or withdraw this standard. Purchasers of American National Standards may receive current information on all standards by calling or writing the American National Standards Institute.

Published by

American National Standards Institute
1430 Broadway, New York, New York 10018

Printed in the United States of America

BB1½M1289/35

Foreword (This Foreword is not part of American National Standard X3.124.3-1989.)

This American National Standard provides access to a set of basic functions for computer graphics programming in accordance with American National Standard Reference Manual for the Ada Programming Language, ANSI/MIL-STD 1815A-1983. These graphics functions taken as a whole are called the Ada language binding of the graphical kernel system (GKS).

The graphical kernel system is a set of basic functions for computer graphics programming usable by many graphics-producing applications. This standard (1) allows graphics application programs to be easily transported between installations, (2) aids Ada graphics applications programmers in understanding and using graphics methods, and (3) guides device manufacturers on useful graphics capabilities.

This standard defines an Ada application level programming interface to a graphics system. Hence, it contains functions for (1) outputting graphical primitives; (2) controlling the appearance of graphical primitives with attributes; (3) controlling graphical workstations; (4) controlling transformations and coordinate systems; (5) generating and controlling groups of primitives, called segments; (6) obtaining graphical input; (7) manipulating groups of device-independent instructions, called metafiles; (8) inquiring the capabilities and states of the graphics system; and (9) handling errors.

For each GKS function, the Ada procedure name, argument list, and argument data types are given. In addition, any special errors associated only with the Ada language binding of GKS are specified and assigned unique error numbers. The data type definitions used to define the Ada binding to GKS are defined in a separate Ada package, package GKS_TYPES; the GKS procedures or functions, or both, are defined in a separate Ada package, package GKS.

Twelve upwardly compatible levels of conformance are defined, addressing the most common classes of equipment and applications.

American National Standard for Information Systems — Computer Graphics — Graphical Kernel System (GKS) Functional Description, ANSI X3.124-1985, is supplemented by this derivative standard. ANSI X3.124-1985 corresponds to ISO 7942:1985 in that it represents the functional aspects of GKS. ANSI X3.124-1985 contains specifications not present in ISO 7942:1985, namely, the syntax for using GKS functions and data types from Ada.

The design of this standard is based on the work of many groups. Much of the early design methodology of graphics standards was developed at the Workshop on Graphics Standards Methodology held in May, 1976, in Seillac, France, under IFIP WG5.2 sponsorship. GKS itself was originally developed by Deutsches Institute fur Normung (DIN), the West German standardization institute, in 1978 and was subsequently refined extensively between 1980 and 1982 by Working Group 4 of the Subcommittee on Programming Languages of the Technical committee on Information Processing of the International Organization for Standardization (ISO/IEC JTC1/SC24/WG4). The resulting International Standard (Information Processing — Computer Graphics — Graphical Kernel System (GKS) Functional Description, ISO 7942:1985) was the basis for ANSI X3.124-1985. The development of the GKS was heavily influenced by the work of the Graphics Standards Planning Committee of the Special Interest Group on Computer Graphics of the Association for Computing Machinery (ACM SIGGRAPH GSPC). This work, known as Core System Proposal, was published and widely distributed in 1977 and again (in a revised version) in 1979.

The Ada binding of GKS was started by American participants of Technical

Committee X3H3 of Accredited Standards Committee X3 (Information Processing). After refinement by both Technical Committee X3H3 and ISO/IEC JTC1/SC24/WG4, the document was registered as International Standard for Information Processing Systems — Computer Graphics — Graphical Kernel System (GKS) Language Bindings — Part 3:Ada, ISO 8651-3:1988, in July 1988. ANSI X3.124.3-1989 is identical to ISO 8651-3:1988 in almost all areas of the standard. All functional capabilities of ISO GKS are found in the ANSI GKS and are bound to the Ada programming language identically. The ANSI GKS does, however, differ in the following ways:

(1) A new minimal output level (denoted m) is defined in ANSI X3.124-1985.
(2) A new section defining a conforming program and a conforming implementation replaces a more restrictive conformance statement found in the body of ISO GKS standard document.
(3) Several of the Annexes in the ISO GKS document have been modified. Also, the word "Annex" has been changed to "Appendix."
(4) The default for ASF's is INDIVIDUAL.
(5) The data records for INPUT have been defined.
(6) Appendix G of ANSI X3.124-1985 contains a detailed and exhaustive list of all the differences between ANSI X3.124-1985 and ISO 7942:1985.

All these differences are reflected in this standard.

The FORTRAN language binding already exists as ANSI X3.124.1-1985 and ISO 8651-1:1988, and the Pascal language binding has been published as ANSI X3.124.2-1988 and ISO 8651-2:1988. The C language binding for GKS is currently under development by Technical Committee X3H3. This standard, when approved by X3 and ANSI, will be published as ANSI X3.124.4. Internationally, this language binding of GKS will be published as ISO 8641-4.

This standard was developed by Technical Committee X3H3 of Accredited Standards Committee X3 under two projects authorized by X3, namely, project 268D and project 362D. More specifically, GKS, as a whole, meets the goals of project 268D, while the minimal output level m found in this American National Standard, but not present in ISO 7942:1985, meets the goals of project 362D. Both projects authorized the specification of syntax (as embodied in a programming language binding) as well as semantics.

This standard was approved as an American National Standard by the American National Standards Institute on June 30, 1989.

Suggestions for improvement of this standard will be welcome. They should be sent to the Computer and Business Equipment Manufacturers Association, 311 First Street, NW, Suite 500, Washington, DC 20001.

This standard was processed and approved for submittal to ANSI by the Accredited Standards Committee on Information Processing Systems, X3. Committee approval of the standard does not necessarily imply that all committee members voted for its approval. At the time it approved this standard, the X3 Committee had the following members:

Richard Gibson, Chair
Donald C. Loughry, Vice-Chair
Catherine A. Kachurik, Administrative Secretary

Organization Represented	*Name of Representative*
Allen-Bradley	Ronald H. Reimer
American Library Association	Paul Peters
American Nuclear Society	Geraldine C. Main
AMP, Inc	Edward Kelly
	Ronald Lloyd (Alt)
Apple	Jean Luc LeBrun
	Michael J. Lawler (Alt)
Association of the Institute for Certification of Computer Professionals	Thomas M. Kurihara
AT&T	Thomas F. Frost
	Paul D. Bartoli (Alt)
Boeing Company	Paul W. Mercer
Compaq Computer Corporation	James Barnes
Control Data Corporation	Earnest Fogle
Cooperating Users of Burroughs Equipment	Thomas Easterday
	Donald Miller (Alt)
Dataproducts Corporation	Charles D. Card
Digital Equipment Computer Users Society	James Ebright
Digital Equipment Corporation	Gary S. Robinson
	Delbert L. Shoemaker (Alt)
Eastman Kodak	Gary Haines
	James D. Converse (Alt)
Electronic Data Systems Corporation	Jerrold S. Foley
GUIDE International	Frank Kirshenbaum
	Sandra Swartz Abraham (Alt)
Hewlett-Packard	Donald C. Loughry
Honeywell Bull	David M. Taylor
IBM Corporation	Mary Anne Gray
	Robert H. Follett (Alt)
IEEE Computer Society	Tom Hannon
	Bob Pritchard (Alt)
Lawrence Berkeley Laboratory	David F. Stevens
	Robert L. Fink (Alt)
MAP/TOP	(Representation Vacant)
	Mike Kaminski (Alt)
Moore Business Forms	Delmer H. Oddy
National Communications System	Denis Bodson
	Donald Wilson (Alt)
National Institute of Standards and Technology	Robert E. Rountree
	Mike Hogan (Alt)
NCR Corporation	Tom Kern
	A.R. Daniels (Alt)
OMNICOM	Harold C. Folts
	Cheryl Slobodian (Alt)
Prime Computer Inc	Arthur Norton
Recognition Technology Users Association	Herbert F. Schantz
SHARE, Inc	Thomas B. Steel
	Gary Ainsworth (Alt)
3M Company	Paul D. Jahnke
Unisys	Marvin W. Bass
	Steven Oksala (Alt)
U.S. Department of Defense	William C. Rinehuls
	Thomas H. Kurihara (Alt)
U.S. General Services Administration	Dale O. Christensen
	Larry L. Jackson (Alt)
U.S. WEST	Gary Dempsey
	Sue Caparo (Alt)
VIM	Chris Tanner
	John Ulrich (Alt)

Organization Represented	*Name of Representative*
Wang Corporation	J.J. Cinecoe Sarah Wagner (Alt)
Wintergreen Information Services	John L. Wheeler
Xerox Corporation	Roy Pierce

Technical Committee X3H3 on Computer Graphics, which developed the draft proposals, which held the U.S. Technical Advisory Group responsibilities for ISO/IEC JTC1/SC24/WG4 (formerly ISO TC 97/SC5/WG21), had the following members:

Peter R. Bono, Chair
(Peter R. Bono Associates, Inc.)
John L. McConnell, Vice Chair
(Digital Equipment Corp.)
Randall L. Simons, Secretary
(Sandia Labs)
Janet Chin, International
Representative
(Chin Associates)

Salim Abi-Ezzi
(Rensselaer Polytechnic)
David Ambrose
(Bruning)
Roxann Arey
(Computervision)
David Bailey
(Calcomp)
James E. Bennett
(Convergent Tech)
Bob Benson
(SHARE)
John Blair
(National Semiconductor)
Loren Buchanen
(Computer Sciences)
John Butler
(Microsoft)
Debbie Cahn
(Boeing Comp Services)
George S. Carson
(GSC Associates)
Bruce Coughran
(Computer Intelligence)
Tom Crawford
(Advanced Micro Devices)
Geri Cuthbert
(Software Technology)
Andrew Daniel
(Video Seven)
Frank Dawson
(McDonnell Douglas)
Ann Doice
(Data General)
Sahib A. Dudani
(Advanced Technology)
Richard Ehlers
(Evans & Sutherland)
James Flatten
(DECUS)
Michael Franusich
(Digital Research)
Octavio Garcia
(Tektronix)
Zhana Gelman-Reyf
(Lundy Electronics)
Edward J. Gerety
(Comp Assoc Intel)
Jeffrey Gishen
(Masscomp)
Richard Greco
(Digital Research)

James Grimes
(Intel)
Georges Grinstein
(Univ of Lowell)
Giora Guth
(Prime Computer)
Jennifer J. Haley
(Amoco Production)
Terry Harney
(Hughes Aircraft)
Mike Heck
(Vantage)
Lofton Henderson
(Henderson Software)
Thomas Hodgens
(DEST)
Raymond Hoffman
(AT&T Communications)
Mark Hood
(Applicon Schlumberger)
Joyce Howell
(Intergraph)
John Jeter
(Texas Instruments)
Elizabeth Johnson
(General Electric)
Peter Jones
(Olivetti ATC)
James Kearney
(U.S. Army)
Joseph Lamm
(Tech-Source)
Mark Landguth
(Mark Landguth)
Barbara Lurvey
(Wang Laboratories)
Magsood Mahmud
(Conographic)
Russell McDowell
(Northern Telecom)
Eileen McGinnis
(Sun Microsystems)
James Michener
(Apollo Computer)
Mary Miller
(Landmark Graphics)
Judith Milton
(Computer Image)
Thomas Morrissey
(Hewlett-Packagrd)
T. Motoyama
(Richoh Systems)

David Nation
(Department of Defense)
Christopher Nelson
(Nova Graphics)
Lawrence M. Ockene
(Raster Technologies)
Constantine Pavlakos
(Sandia National Labs)
Burt W. Perry
(Graphics Software Sys)
Niel Pierman
(Precision Visuals)
Brian Plunkett
(Houston Instruments)
Paul Porter
(Systems One Software)
Thomas Powers
(Digital Equipment Corp)
Richard F. Puk
(Puk Consulting)
Mark Quinlan
(Megatek)
Theodore N. Reed
(Los Alamos Natl Lab)
C.B. Rice
(NCR)
Reed Rinn
(Tandem Computer)
Cynthia Rodriguez
(Standard Oil)
Jeffrey H. Rowe
(Lawrence Livermore NL)
Daryl Salmons
(TRW)
George Schaeffer
(Janus Enterprises)
Harold S. Schechter
(MagiCorp)
Shreyas Shah
(EDS)
Barry Shepherd
(IBM)
Donald Singley
(3M)
Mark Skall
(NIST)
Madeleine R. Sparks
(Unisys)
Jonathan Steinhart
(American Info Tech)
James A. Terrel
(Lockheed)
Karla Vecchiet
(Microfield Gaphics)
John Yambor
(SASC Technologies)
Bob Willis
(NCGA)

The document editor for this standard was Geraldine Cuthbert. The camera-ready masters for the body of this standard were provided by Software Technology, Incorporated, Melbourne, FL.

Contents

American National Standard
for Information Systems –

Computer Graphics –
Graphical Kernel System (GKS)
Ada Binding

0 Introduction

The Graphical Kernel System (GKS) is fully described in American National Standard for Information Systems — Computer Graphics — Graphical Kernel System (GKS) Functional Description, ANSI X3.124-1985. As explained in the scope and field of applications of ANSI X3.124-1985, that American National Standard 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 document is to define a standard binding for the Ada computer programming language.

1 Scope and Field of Application

The Graphical Kernel System (GKS), as described in ANSI X3.124-1985, 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 document specifies such a language-dependent layer for the Ada language.

2 References

ANSI X3.124-1985, American National Standard for Information Systems — Computer Graphics — Graphical Kernel System (GKS) Functional Description.

ANSI/MIL-STD-1815A-1983, American National Standard Reference Manual for the Ada Programming Language

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 (ANSI X3.124-1985) 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 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 is 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.

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

3.2.6 Packaging

The GKS standard defines twelve levels of graphic functionality, with level ma 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 twelve 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 American National Standard Reference Manual for the Ada Programming Language ANSI/MIL-STD-1815A-1983, 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 ANSI X3.124-1985 and according to the rules for the designation and operation of registration authorities in ISO Directives, the ISO Council has designated the National Institute of Standards and Technology, A266 Technology Building, Gaithersburg, MD, 20899, USA, to act as registation authority.

4 Tables

4.1 Tables of GKS Functions and Abbreviations

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

Table 2
List of Procedures Using the Abbreviations (Continued)

GKS

- CLOSE_GKS
- EMERGENCY_CLOSE_GKS
- INQ_LEVEL_OF_GKS
- OPEN_GKS

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_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

(Continued on Next Page)

Table 2
List of Procedures Using the Abbreviations (Continued)

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
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_MARKER_SIZE_SCALE_FACTOR
INQ_MARKER_TYPE
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_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

Table 2
List of Procedures Using the Abbreviations (Continued)

	INQ_STRING_DEVICE_STATE
	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

Table 3
GKS Function Names and Ada Names Ordered Alphabetically by Ada Bound Name
(Continued)

Ada Bound Name	GKS Function
INQ_COLOUR_FACILITIES	INQUIRE COLOUR FACILITIES
INQ_COLOUR_REPRESENTATION	INQUIRE COLOUR REPRESENTATION
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

(Continued on Next Page)

Table 3
GKS Function Names and Ada Names Ordered Alphabetically by Ada Bound Name
(Continued)

Ada Bound Name	GKS Function
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
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

Table 3
GKS Function Names and Ada Names Ordered Alphabetically by Ada Bound Name
(Continued)

Ada Bound Name	GKS Function
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
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

(Continued on Next Page)

Table 3
GKS Function Names and Ada Names Ordered Alphabetically by Ada Bound Name
(Continued)

Ada Bound Name	GKS Function
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
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

Table 3
GKS Function Names and Ada Names Ordered Alphabetically by Ada Bound Name
(Continued)

Ada Bound Name	GKS Function
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
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

(Continued on Next Page)

Tables

Table 3
GKS Function Names and Ada Names Ordered Alphabetically by Ada Bound Name
(Continued)

Ada Bound Name	GKS Function
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 ma
- ACTIVATE_WS
- CLEAR_WS
- CLOSE_GKS
- CLOSE_WS
- DEACTIVATE_WS
- ESCAPE
- FILL_AREA
- INQ_CLIPPING
- INQ_COLOUR_FACILITIES
- INQ_COLOUR_REPRESENTATION
- 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

Table 4
Alphabetical List of GKS Functions by Level (Continued)

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_LEVEL_OF_GKS
INQ_LIST_OF_COLOUR_INDICES
INQ_MAX_LENGTH_OF_WS_STATE_TABLES
INQ_NORMALIZATION_TRANSFORMATION
INQ_POLYLINE_FACILITIES
INQ_POLYMARKER_FACILITIES
INQ_TEXT_EXTENT
INQ_TEXT_FACILITIES
INQ_WS_CONNECTION_AND_TYPE
INQ_WS_TRANSFORMATION
OPEN_GKS
OPEN_WS
POLYLINE
POLYMARKER
SELECT_NORMALIZATION _TRANSFORMATION
SET_CHAR_HEIGHT
SET_CHAR_UP_VECTOR
SET_CLIPPING_INDICATOR
SET_COLOUR_REPRESENTATION
SET_FILL_AREA_COLOUR_INDEX
SET_FILL_AREA_INTERIOR_STYLE
SET_LINETYPE

(Continued on Next Page)

Table 4
Alphabetical List of GKS Functions by Level (Continued)

SET_MARKER_TYPE
SET_POLYLINE_COLOUR_INDEX
SET_POLYMARKER_COLOUR_INDEX
SET_TEXT_ALIGNMENT
SET_TEXT_COLOUR_INDEX
SET_VIEWPORT
SET_WINDOW
SET_WS_VIEWPORT
SET_WS_WINDOW

TEXT

UPDATE_WS

LEVEL mb

INITIALISE_CHOICE
INITIALISE_LOCATOR
INITIALISE_STRING
INITIALISE_STROKE
INITIALISE_VALUATOR
INQ_CHOICE_DEVICE_STATE
INQ_DEFAULT_CHOICE_DEVICE_DATA
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 mc

AWAIT_EVENT
FLUSH_DEVICE_EVENTS

Table 4
Alphabetical List of GKS Functions by Level (Continued)

GET_CHOICE
GET_LOCATOR
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 0a

CELL_ARRAY
EMERGENCY_CLOSE_GKS
ERROR_HANDLING
ERROR_LOGGING
GDP
GET_ITEM_TYPE_FROM_GKSM
INQ_GDP
INQ_LIST_OF_AVAILABLE_GDP
INQ_LIST_OF_AVAILABLE_WS_TYPES
INQ_LIST_OF_NORMALIZATION_TRANSFORMATION_NUMBER
INQ_OPERATING_STATE_VALUE
INQ_PATTERN_FACILITIES
INQ_PIXEL
INQ_PIXEL_ARRAY
INQ_PIXEL_ARRAY_DIMENSIONS
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_WS_CATEGORY
INQ_WS_CLASSIFICATION
INQ_WS_DEFFERRAL_AND_UPDATE_STATES
INQ_WS_STATE
INTERPRET_ITEM
READ_ITEM_FROM_GKSM
SET_ASF
SET_CHAR_EXPANSION_FACTOR
SET_CHAR_SPACING
SET_FILL_AREA_INDEX
SET_FILL_AREA_STYLE_INDEX
SET_LINEWIDTH_SCALE_FACTOR

(Continued on Next Page)

Table 4
Alphabetical List of GKS Functions by Level (Continued)

SET_MARKER_SIZE_SCALE_FACTOR
SET_PATTERN_REFERENCE_POINT
SET_PATTERN_SIZE
SET_POLYLINE_INDEX
SET POLYMARKER_INDEX
SET_TEXT_FONT_AND_PRECISION
SET_TEXT_INDEX
SET_TEXT_FONT_AND_PRECISION
SET_TEXT_PATH
WRITE_ITEM_TO_GKSM

LEVEL 0b

NONE

LEVEL 0c

NONE

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

Table 4
Alphabetical List of GKS Functions by Level (Continued)

RENAME_SEGMENT
SET_DEFERRAL_STATE
SET_FILL_AREA_REPRESENTATION
SET_HIGHLIGHTING
SET_PATTERN_REPRESENTATION
SET_POLYLINE_REPRESENTATION
SET_POLYMARKER_REPRESENTATION
SET_SEGMENT_PRIORITY
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 shall 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    := INDIVIDUAL;
       WIDTH_ASF                 : ASF    := INDIVIDUAL;
       LINE_COLOUR_ASF           : ASF    := INDIVIDUAL;
       TYPE_OF_MARKER_ASF        : ASF    := INDIVIDUAL;
       SIZE_ASF                  : ASF    := INDIVIDUAL;
       MARKER_COLOUR_ASF         : ASF    := INDIVIDUAL;
       FONT_PRECISION_ASF        : ASF    := INDIVIDUAL;
       EXPANSION_ASF             : ASF    := INDIVIDUAL;
       SPACING_ASF               : ASF    := INDIVIDUAL;
       TEXT_COLOUR_ASF           : ASF    := INDIVIDUAL;
       INTERIOR_ASF              : ASF    := INDIVIDUAL;
       STYLE_ASF                 : ASF    := INDIVIDUAL;
       FILL_AREA_COLOUR_ASF      : ASF    := INDIVIDUAL;
    end record;

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

ATTRIBUTES_FLAG LEVEL 0a

```
type ATTRIBUTES_FLAG is (CURRENT, SPECIFIED);

-- Indicates whether output attributes that are to be used for prompting and
-- echoing are to be as currently set, or as explicitly specified.
```

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 ma

```
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 ma

```
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 mb

```
type CHOICE_DEVICE_NUMBER is new DEVICE_NUMBER;

-- Provides for choice device identifiers.
```

CHOICE_PROMPT LEVEL mb

```
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 mb

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

 -- Provides for lists of prompts.
```

CHOICE_PROMPT_ECHO_TYPE LEVEL mb

```
type CHOICE_PROMPT_ECHO_TYPE is new INTEGER;

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

CHOICE_PROMPT_ECHO_TYPES LEVEL mb

```
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 mb

```
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 mb

```
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 mb

```
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 mb

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

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

CHOICE_SMALL_NATURAL LEVEL mb

```
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 mb

```
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 mb

```
type CHOICE_VALUE is new POSITIVE;

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

CLIPPING_INDICATOR LEVEL ma

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

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

COLOUR_AVAILABLE LEVEL ma

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

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

COLOUR_INDEX LEVEL ma

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

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

COLOUR_INDICES LEVEL ma

```
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 ma

```
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 ma

```
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 ma

```
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 ma

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

-- Defines the Device Coordinate System.
```

DC_TYPE LEVEL ma

```
type DC_TYPE is digits PRECISION;

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

DC_UNITS LEVEL ma

```
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 ma

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

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

DEVICE_NUMBER LEVEL mb

```
package DEVICE_NUMBER_TYPE is
        type DEVICE_NUMBER 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 mb

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

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

ERROR_NUMBER LEVEL ma

```
type ERROR_NUMBER is new INTEGER;

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

EVENT_DEVICE_NUMBER LEVEL mc

```
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 mc

```
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_DATA LEVEL mb

```
type FILL_AREA_DATA (ATTRIBUTES : ATTRIBUTES_FLAG := CURRENT) is
    record
      case ATTRIBUTES is
        when SPECIFIED =>
            STYLE_ASF                : ASF;
            STYLE_INDEX_ASF          : ASF;
            COLOUR_ASF               : ASF;
            INDEX                    : FILL_AREA_INDEX;
            INTERIOR                 : INTERIOR_STYLE;
            STYLE                    : STYLE_INDEX;
            FILL_AREA_COLOUR         : COLOUR_INDEX;
        when CURRENT => NULL;
      end case;
    end record;

-- A record containing information needed for input data records to specify the
-- appearance of a filled area. It is also used for results of inquiry about the contents
-- of data records. The information stored in this record is accessible through the
-- use of the subprograms for manipulating data records.
```

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 ID's.
```

GKS_LEVEL LEVEL ma

```
type GKS_LEVEL is (Lma, Lmb, Lmc, 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 ma

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

HATCH_STYLES LEVEL ma

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

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

HORIZONTAL_ALIGNMENT LEVEL ma

```
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 ma

```
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 ma

```
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 mb

```
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 mc

```
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 mb

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

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

INPUT_STRING LEVEL mb

```
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 ma

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

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

INTERIOR_STYLE LEVEL ma

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

-- Defines the fill area interior styles.
```

INTERIOR_STYLES LEVEL ma

```
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.
```

LINE_DATA LEVEL mb

```
type LINE_DATA (ATTRIBUTES : ATTRIBUTES_FLAG := CURRENT) is
   record
      case ATTRIBUTES is
         when SPECIFIED =>
               LINE_ASF          : ASF;
               WIDTH_ASF         : ASF;
               COLOUR_ASF        : ASF;
               INDEX             : POLYLINE_INDEX;
               LINE              : LINETYPE;
               WIDTH             : LINEWIDTH;
               LINE_COLOUR       : COLOUR_INDEX;
         when CURRENT => NULL;
      end case;
   end record;

-- A record containing information needed for input data records to specify the
-- appearance of prompting and echo types. It is also used for results of inquiry about
-- the contents of data records. The information stored in this record is accessible through
-- the use of the subprograms for manipulating data records.
```

LINETYPE LEVEL ma

```
type LINETYPE is new INTEGER;

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

LINETYPES LEVEL ma

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

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

LINEWIDTH LEVEL ma

```
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 mb

```
type LOCATOR_DEVICE_NUMBER is new DEVICE_NUMBER;

-- Provides for locator device identifiers.
```

LOCATOR_PROMPT_ECHO_TYPE LEVEL mb

```
type LOCATOR_PROMPT_ECHO_TYPE is new INTEGER;

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

LOCATOR_PROMPT_ECHO_TYPES LEVEL mb

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

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

MARKER_DATA LEVEL mb

```
type MARKER_DATA (ATTRIBUTES : ATTRIBUTES_FLAG := CURRENT) is
    record
      case ATTRIBUTES is
        when SPECIFIED =>
            MARKER_ASF          : ASF;
            SIZE_ASF            : ASF;
            COLOUR_ASF          : ASF;
            INDEX               : POLYMARKER_INDEX;
            MARKER              : MARKER_TYPE;
            SIZE                : MARKER_SIZE;
            MARKER_COLOUR       : COLOUR_INDEX;
        when CURRENT => NULL;
      end case;
    end record;

-- A record containing information needed for input data records to specify the
-- appearance of prompting and echo types. It is also used for results of inquiry about
-- the contents of data records. The information stored in this record is accessible through
-- the use of the subprograms for manipulating data records.
```

MARKER_SIZE LEVEL ma

```
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 ma

```
type MARKER_TYPE is new INTEGER;

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

MARKER_TYPES LEVEL ma

```
package MARKER_TYPES is new GKS_LIST_UTILITIES (MARKER_TYPE);

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

MORE_EVENTS LEVEL mc

```
type MORE_EVENTS is (NOMORE, MORE);

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

NDC LEVEL ma

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

NDC_TYPE LEVEL ma

```
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 mb

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

OPERATING_STATE LEVEL ma

```
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 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_INDEX LEVEL ma

```
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 ma

```
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.
```

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 ma

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

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

RASTER_UNITS LEVEL ma

```
type RASTER_UNITS is new POSITIVE;

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

RASTER_UNIT_SIZE LEVEL ma

```
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 ma

```
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 ma

```
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 ma

```
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 mb

```
type STRING_DEVICE_NUMBER is new DEVICE_NUMBER;

-- Provides for string device number.
```

STRING_PROMPT_ECHO_TYPE LEVEL mb

```
type STRING_PROMPT_ECHO_TYPE is new INTEGER;

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

STRING_PROMPT_ECHO_TYPES LEVEL mb

```
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 ma

```
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 mb

```
type STROKE_DEVICE_NUMBER is new DEVICE_NUMBER;

-- Provides for stroke device numbers.
```

STROKE_PROMPT_ECHO_TYPE LEVEL mb

```
type STROKE_PROMPT_ECHO_TYPE is new INTEGER;

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

STROKE_PROMPT_ECHO_TYPES LEVEL mb

```
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 ma

```
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 ma

```
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 ma

```
type TEXT_FONT is new INTEGER;

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

TEXT_FONT_PRECISION LEVEL ma

```
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 ma

```
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 ma

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

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

TEXT_PRECISION LEVEL ma

```
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 ma

```
type TRANSFORMATION_NUMBER is new NATURAL;

-- A normalization transformation number.
```

TRANSFORMATION_PRIORITY_ARRAY LEVEL ma

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

-- Type to store transformation numbers.
```

TRANSFORMATION_PRIORITY_LIST LEVEL ma

```
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 mb

```
type VALUATOR_DEVICE_NUMBER is new DEVICE_NUMBER;

-- Provides for valuator device identifiers.
```

VALUATOR_INPUT_VALUE LEVEL mb

```
type VALUATOR_INPUT_VALUE is digits PRECISION;

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

VALUATOR_PROMPT_ECHO_TYPE LEVEL mb

```
type VALUATOR_PROMPT_ECHO_TYPE is new INTEGER;

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

VALUATOR_PROMPT_ECHO_TYPES LEVEL mb

```
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 ma

```
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 ma

```
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 ma

```
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 ma

```
package WC is new GKS_COORDINATE_SYSTEM (WC_TYP`

-- Defines the World Coordinate System.
```

WC_TYPE LEVEL ma

```
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 ma

```
type WS_ID is new POSITIVE;

-- Defines the range of workstation identifiers.
```

WS_IDS LEVEL ma

```
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 ma

```
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 ma

```
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 shall 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 mb

```
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 metafile data records (5.2.4).
```

LOCATOR_DATA_RECORD LEVEL mb

```
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 1b

```
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 mb

```
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 mb

```
type STROKE_DATA_RECORD (PROMPT_ECHO_TYPE :
     STROKE_PROMPT_ECHO_TYPE := DEFAULT_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 mb

```
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 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 5.0.

```
DEFAULT_MEMORY_UNITS        : constant                  := implementation defined;
PRECISION                   : constant                  := implementation_defined;
DEFAULT_ERROR_FILE          : constant  STRING          := implementation_defined;
```

The following defines the predefined exception GKS_ERROR defined in 3.2.3.

```
GKS_ERROR                   : exception;
```

The following constants define maximum implementation limits for GKS/Ada types.

```
SMALL_NATURAL_MAX           : constant := implementation_defined
STRING_SMALL_NATURAL_MAX    : constant := implementation_defined
CHOICE_SMALL_NATURAL_MAX    : constant := implementation_defined.
```

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 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 Appendix 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

ANSI X3.124-1985 provides a mapping of error numbers for each GKS function. Certain 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 .

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 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 errors occur, the predefined Ada exception that caused the error is raised automatically.

2501 Unknown error occurred during processing
2502 Usage error is 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 ma

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

CLOSE GKS LEVEL ma

```
procedure CLOSE_GKS;
```

OPEN WORKSTATION LEVEL ma

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

CLOSE WORKSTATION LEVEL ma

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

ACTIVATE WORKSTATION LEVEL ma

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

DEACTIVATE WORKSTATION LEVEL ma

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

CLEAR WORKSTATION LEVEL ma

```
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 ma

```
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 ma

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 ma

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

POLYMARKER LEVEL ma

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

TEXT LEVEL ma

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

FILL AREA LEVEL ma

```
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 GDP's 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_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;
                           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 ma

```
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 ma

```
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 ma

```
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 ma

```
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 ma

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

SET CHARACTER HEIGHT LEVEL ma

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

SET CHARACTER UP VECTOR LEVEL ma

```
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 ma

```
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 ma

```
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 ma

```
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 ma

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

SET WINDOW LEVEL ma

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

SET VIEWPORT LEVEL ma

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

SET VIEWPORT INPUT PRIORITY LEVEL mb

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

SELECT NORMALIZATION TRANSFORMATION LEVEL ma

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

SET CLIPPING INDICATOR LEVEL ma

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

SET WORKSTATION WINDOW LEVEL ma

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

SET WORKSTATION VIEWPORT LEVEL ma

```
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 mb

```
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 mb

```
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 mb

```
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 mb

```
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 mb

```
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 mb

```
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 mb

```
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 mb

```
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 mb

```
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 mb

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

REQUEST LOCATOR LEVEL mb

```
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 mb

```
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 mb

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

REQUEST CHOICE LEVEL mb

```
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 mb

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

SAMPLE LOCATOR LEVEL mc

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

SAMPLE STROKE LEVEL mc

```
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 mc

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

SAMPLE CHOICE LEVEL mc

```
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 mc

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

AWAIT EVENT LEVEL mc

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

FLUSH DEVICE EVENTS LEVEL mc

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

GET LOCATOR LEVEL mc

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

GET STROKE LEVEL mc

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

GET VALUATOR LEVEL mc

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

GET CHOICE LEVEL mc

```
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 mc

```
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

```
procedure INQ_OPERATING_STATE_VALUE
  (VALUE  : out OPERATING_STATE);
```

INQUIRE LEVEL OF GKS LEVEL ma

```
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 ma

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

-- The following procedures support inquiry of the primitive values individually.

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 ma

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

-- The following procedures support inquiry of the individual attributes individually.

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);
```

INQUIRE CURRENT NORMALIZATION TRANSFORMATION NUMBER LEVEL ma

```
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 ma

```
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 ma

```
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 mc

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

INQUIRE WORKSTATION CONNECTION AND TYPE LEVEL ma

```
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 ma

```
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 ma

```
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 ma

```
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 ma

```
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 mb

```
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);
```

INQUIRE STROKE DEVICE STATE LEVEL mb

```
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 mb

```
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 mb

```
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 mb

```
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 ma

```
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 ma

```
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 ma

```
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 ma

```
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              : outCHAR_SPACING;
   TEXT_COLOUR          : out COLOUR_INDEX);
```

INQUIRE FILL AREA FACILITIES LEVEL ma

```
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 ma

```
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 ma

```
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 mb

```
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 mb

```
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 mb

```
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 mb

```
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 mb

```
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 mb

```
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);
```

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 mc

```
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 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 mb.

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 for the locator prompt and echo
        -- types 1,2,3, and 6.

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

        -- Constructs and returns a locator data record for the locator prompt and echo
        -- type 5 when its attributes are specified by Polyline attributes.

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

        -- Constructs and returns a locator data record for the locator prompt and echo
        -- type 5 when its attributes are specified by Fill Area attributes.

        function ATTRIBUTE_FLAG
                (DATA_RECORD  :  in LOCATOR_DATA_RECORD)
          return ATTRIBUTES_FLAG;

        -- Returns the attribute flag CURRENT  or SPECIFIED stored in the data record
        -- for the prompt and echo types 4 and 5.
```

```
        function LOCATOR_ATTRIBUTES_USED
                (DATA_RECORD : in LOCATOR_DATA_RECORD)
          return ATTRIBUTES_USED_TYPE;

        -- Returns which attribute set, either Polyline or Fill Area, stored in the data record
        -- for prompt and echo types 4 and 5.

        function LINE_ATTRIBUTES
                (DATA_RECORD : in LOCATOR_DATA_RECORD)
          return LINE_DATA;

        -- Returns the Polyline attribute information stored in the data record for
        -- prompt and echo types 4 or 5.

        function FILL_AREA_ATTRIBUTES
                (DATA_RECORD : in LOCATOR_DATA_RECORD)
          return FILL_AREA_DATA;

        -- Returns the Fill Area attribute information sotred in the data record for
        -- prompt and echo type 5.

-- Stroke Data Record Operations.

        procedure BUILD_STROKE_DATA_RECORD
                (PROMPT_ECHO_TYPE       : in STROKE_PROMPT_ECHO_TYPE;
                BUFFER_SIZE             : in POSITIVE;
                POSITION                : in POSITIVE;
                INTERVAL                : in WC.SIZE;
                TIME                    : in DURATION;
                DATA_RECORD             : out STROKE_DATA_RECORD);

        -- Constructs and returns a stroke data record for stroke prompt and echo types 1 and 2.

        procedure BUILD_STROKE_DATA_RECORD
                (PROMPT_ECHO_TYPE       : in STROKE_PROMPT_ECHO_TYPE;
                BUFFER_SIZE             : in POSITIVE;
                POSITION                : in POSITIVE;
                INTERVAL                : in WC.SIZE;
                TIME                    : in DURATION;
                CONTENTS                : in MARKER_DATA;
                DATA_RECORD             : out STROKE_DATA_RECORD);

        -- Constructs and returns a stroke data record for stroke prompt and echo type 3.

        procedure BUILD_STROKE_DATA_RECORD
                (PROMPT_ECHO_TYPE       : in STROKE_PROMPT_ECHO_TYPE;
                BUFFER_SIZE             : in POSITIVE;
                POSITION                : in POSITIVE;
                INTERVAL                : in WC.SIZE;
                TIME                    : in DURATION;
                CONTENTS                : in LINE_DATA;
                DATA_RECORD             : out STROKE_DATA_RECORD);

        -- Constructs and returns a stroke data record for stroke prompt and echo type 4.
```

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

-- Returns the size of the input stroke buffer stored in the data record for prompt and
-- echo types 1, 2, 3 and 4.

function POSITION
        (DATA_RECORD  :  in STROKE_DATA_RECORD)
    return  POSITIVE;

-- Returns the editing position within the stroke input buffer stored in the data record
-- for prompt and echo types 1, 2, 3, and 4.

function INTERVAL
        (DATA_RECORD  :  in STROKE_DATA_RECORD)
   return WC.SIZE;

-- Returns the interval value stored in the stroke data record for the prompt and echo
-- types 1, 2, 3, and 4.

function TIME
        (DATA_RECORD  :  in  STROKE_DATA_RECORD)
   return  DURATION;

-- Returns the measuring time for sampling stroke input stored in the stroke data record
-- for prompt and echo types 1, 2, 3 and 4.

function MARKER_ATTRIBUTES
        (DATA_RECORD  :  in  STROKE_DATA_RECORD)
   return MARKER_DATA;

-- Returns the Polymarker attributes used to echo the stroke input stored in the data
-- record for prompt and echo type 3.

function LINE_ATTRIBUTES
        (DATA_RECORD  : in STROKE_DATA_RECORD)
   return LINE_DATA;

-- Returns the Polyline attributes used to echo the stroke input stored in the data record
-- for prompt and echo type 4.

-- 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 for the valuator prompt and echo
-- types 1, 2, and 3.
```

```
        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 for
        -- prompt and echo types 1, 2, and 3.

        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 for prompt and
        -- types 1, 2, and 3.

-- 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 for choice prompt and echo type 1.

        procedure BUILD_CHOICE_DATA_RECORD
                (PROMPT_ECHO_TYPE       : in CHOICE_PROMPT_ECHO_TYPE;
                 ARRAY_OF_PROMPTS       : in CHOICE_PROMPTS.LIST_OF;
                 DATA_RECORD            : out CHOICE_DATA_RECORD);

        -- Constructs and returns a choice data record for choice prompt and echo type 2.

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

        -- Constructs and returns a choice data record for choice prompt and echo types 3 and 4.

        procedure BUILD_CHOICE_DATA_RECORD
                (PROMPT_ECHO_TYPE       : in CHOICE_PROMPT_ECHO_TYPE;
                 SEGMENT                : in SEGMENT_NAME;
                 LIST_OF_PICK_IDS       : in PICK_IDS.LIST_OF;
                 DATA_RECORD            : out CHOICE_DATA_RECORD);

        -- Constructs and returns a choice data record for choice prompt and echo type 5.

        function ARRAY_OF_PROMPTS
                (DATA_RECORD : in CHOICE_DATA_RECORD)
          return CHOICE_PROMPTS.LIST_OF;

        -- Returns the array of prompts stored in the choice data record for prompt and echo
        -- type 2.

        function ARRAY_OF_STRINGS
                (DATA_RECORD : in CHOICE_DATA_RECORD)
          return CHOICE_PROMPT_STRING_LIST;

        -- Returns the array of prompt strings stored in the choice data record for prompt
        -- and echo types 3 and 4.
```

```
        function SEGMENT
                (DATA_RECORD : in CHOICE_DATA_RECORD)
          return SEGMENT_NAME;

        -- Returns the segment name stored in the choice data record for prompt and
        -- echo type 5.

        function LIST_OF_PICK_IDS
                (DATA_RECORD : in CHOICE_DATA_RECORD)
          return PICK_IDS_LIST.OF;

        -- Returns the list of pick ids stored in the choice data record for prompt and
        -- echo type 5.

-- 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 NATURAL;
                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.
```

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 for data types specified in 4.2.2 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 Utilities 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 accomplsihed by calling DELETE_FROM_LIST repeatedly with a particular element
-- value while the function IS_IN_LIST returns TRUE for that 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

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

The U.S. Department of Defense (DoD) enforces the single Ada language definition of ANSI/MIL-STD-1815A-1983. 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.

Appendix A
Compiled GKS Specification

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

```
-- The GKS_LIST_UTILITIES 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;
```

```
with GKS_LIST_UTILITIES;

-- The GKS_TYPES Package.

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 declaration that 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 that 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  := INDIVIDUAL;
         WIDTH_ASF                 : ASF  := INDIVIDUAL;
         LINE_COLOUR_ASF           : ASF  := INDIVIDUAL;
         TYPE_OF_MARKER_ASF        : ASF  := INDIVIDUAL;
         SIZE_ASF                  : ASF  := INDIVIDUAL;
         MARKER_COLOUR_ASF         : ASF  := INDIVIDUAL;
         FONT_PRECISION_ASF        : ASF  := INDIVIDUAL;
         EXPANSION_ASF             : ASF  := INDIVIDUAL;
         SPACING_ASF               : ASF  := INDIVIDUAL;
         TEXT_COLOUR_ASF           : ASF  := INDIVIDUAL;
         INTERIOR_ASF              : ASF  := INDIVIDUAL;
         STYLE_ASF                 : ASF  := INDIVIDUAL;
         FILL_AREA_COLOUR_ASF      : ASF  := INDIVIDUAL;
      end record;

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

-- ATTRIBUTES_FLAG                                        LEVEL 0a

type ATTRIBUTES_FLAG is (CURRENT, SPECIFIED);

-- Indicates whether output attributes that are to be used for prompting and
-- echoing are to be as currently set, or as explicitly specified.

--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 ma

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 ma

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 ma

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 mb

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 mb

type CHOICE_DEVICE_NUMBER is new DEVICE_NUMBER;

-- Provides for choice device identifiers.

-- LOCATOR_DEVICE_NUMBER                                           LEVEL mb

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 mb

type STRING_DEVICE_NUMBER is new DEVICE_NUMBER;

-- Provides for string device number.

-- STROKE_DEVICE_NUMBER                                    LEVEL mb

type STROKE_DEVICE_NUMBER is new DEVICE_NUMBER;

-- Provides for stroke device numbers.

-- VALUATOR_DEVICE_NUMBER                                  LEVEL mb

type VALUATOR_DEVICE_NUMBER is new DEVICE_NUMBER;

-- Provides for valuator device identifiers.

-- CHOICE_PROMPT                                           LEVEL mb

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 mb

package CHOICE_PROMPTS is
        new GKS_LIST_UTILITIES (CHOICE_PROMPT);

-- Provides for lists of prompts.

-- CHOICE_PROMPT_ECHO_TYPE                                 LEVEL mb

type CHOICE_PROMPT_ECHO_TYPE is new INTEGER;

-- Defines the choice prompt and echo type.

-- CHOICE_PROMPT_ECHO_TYPES                                LEVEL mb

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 mb

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 mb

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 mb

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 mb

type CHOICE_REQUEST_STATUS is (OK, NOCHOICE, NONE);

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

-- CHOICE_STATUS                                                 LEVEL mb

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 mb

type CHOICE_VALUE is new POSITIVE;

-- Defines the choice values available on an implementation.

-- CLIPPING_INDICATOR                                            LEVEL ma

type CLIPPING_INDICATOR is (CLIP, NOCLIP);

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

-- COLOUR_AVAILABLE                                              LEVEL ma

type COLOUR_AVAILABLE is (COLOUR, MONOCHROME);

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

-- PIXEL_COLOUR_INDEX                                            LEVEL ma

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 ma

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

-- Indices into colour tables are of this type.

-- COLOUR_INDICES                                                  LEVEL ma

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 ma

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 ma

type INTENSITY is digits PRECISION range 0.0..1.0;

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

-- COLOUR_REPRESENTATION                                           LEVEL ma

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 ma

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 ma

type DC_TYPE is digits PRECISION;

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

-- DC                                                              LEVEL ma

package DC is new GKS_COORDINATE_SYSTEM (DC_TYPE);

-- Defines the Device Coordinate System.
```

```
-- DC_UNITS                                                    LEVEL ma

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 workstation dependent units
-- otherwise.

-- DEFERRAL_MODE                                               LEVEL ma

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 mb

type ECHO_SWITCH is (ECHO,  NOECHO);

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

-- ERROR_NUMBER                                                LEVEL ma

type ERROR_NUMBER is new INTEGER;

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

```
-- INPUT_CLASS                                                    LEVEL mb

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 mc

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 mc

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 mc

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 ma

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_DATA                                                    LEVEL mb

type FILL_AREA_DATA (ATTRIBUTES : ATTRIBUTES_FLAG := CURRENT) is
    record
      case ATTRIBUTES is
         when SPECIFIED =>
              STYLE_ASF              : ASF;
              STYLE_INDEX_ASF        : ASF;
              COLOUR_ASF             : ASF;
              INDEX                  : FILL_AREA_INDEX;
              INTERIOR               : INTERIOR_STYLE;
              STYLE                  : STYLE_INDEX;
              FILL_AREA_COLOUR       : COLOUR_INDEX;
         when CURRENT => NULL;
      end case;
    end record;

-- A record containing information needed for input data records to specify the appearance of a
-- filled area. It is also used for results of inquiry about the contents of data records. The
-- information stored in this record is accessible through the use of the subprograms for
-- manipulating data records.

-- 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 ma

type GKS_LEVEL is (Lma, Lmb, Lmc, L0a, L0b, L0c, L1a, L1b, L1c, L2a,L 2b, 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 ma

subtype HATCH_STYLE is STYLE_INDEX;

-- Defines the fill area hatch styles type.

-- HATCH_STYLES                                                   LEVEL ma

package HATCH_STYLES is new GKS_LIST_UTILITIES (HATCH_STYLE);

-- Provides for lists of hatch styles.

-- HORIZONTAL_ALIGNMENT                                           LEVEL ma

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 ma

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 mb

type INPUT_STATUS is (OK, NONE);

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

-- INPUT_STRING                                                   LEVEL mb

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 ma

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 ma

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 ma

type LINETYPE is new INTEGER;

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

-- LINEWIDTH                                                 LEVEL ma

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

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

-- LINE_DATA                                                 LEVEL mb

type LINE_DATA (ATTRIBUTES : ATTRIBUTES_FLAG := CURRENT) is
   record
     case ATTRIBUTES is
        when SPECIFIED =>
             LINE_ASF       : ASF;
             WIDTH_ASF      : ASF;
             COLOUR_ASF     : ASF;
             INDEX          : POLYLINE_INDEX;
             LINE           : LINETYPE;
             WIDTH          : LINEWIDTH;
             LINE_COLOUR    : COLOUR_INDEX;
        when CURRENT => NULL;
     end case;
   end record;

-- A record containing information needed for input data records to specify the appearance of
-- prompting and echo types. It is also used for results of inquiry about the contents of data
-- records. The information stored in this record is accessible through the use of the
-- subprograms for manipulating data records.

-- LINETYPES                                                 LEVEL ma

package LINETYPES is new GKS_LIST_UTILITIES (LINETYPE);

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

```
-- LOCATOR_PROMPT_ECHO_TYPE                                    LEVEL mb

type LOCATOR_PROMPT_ECHO_TYPE is new INTEGER;

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

-- LOCATOR_PROMPT_ECHO_TYPES                                   LEVEL mb

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 ma

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 ma

type MARKER_TYPE is new INTEGER;

-- Defines the type for markers provided by GKS.

-- MARKER_DATA                                                 LEVEL mb

type MARKER_DATA (ATTRIBUTES : ATTRIBUTES_FLAG := CURRENT) is
    record
      case ATTRIBUTES is
        when SPECIFIED =>
            MARKER_ASF          : ASF;
            SIZE_ASF            : ASF;
            COLOUR_ASF          : ASF;
            INDEX               : POLYMARKER_INDEX;
            MARKER              : MARKER_TYPE;
            SIZE                : MARKER_SIZE;
            MARKER_COLOUR       : COLOUR_INDEX
        when CURRENT => NULL;
      end case;
    end record;

-- A record containing information needed for input data records to specify the
-- appearance of prompting and echo types. It is also used for results of inquiry about
-- the contents of data records. The information stored in this record is accessible through
-- the use of the subprograms for manipulating data records.
```

```
-- MARKER_TYPES                                          LEVEL ma

package MARKER_TYPES is new GKS_LIST_UTILITIES (MARKER_TYPE);

-- Provides for lists of marker types.

-- MORE_EVENTS                                           LEVEL mc

type MORE_EVENTS is (NOMORE, MORE);

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

-- NDC_TYPE                                              LEVEL ma

type NDC_TYPE is digits PRECISION;

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

-- NDC                                                   LEVEL ma

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 mb

type OPERATING_MODE is (REQUEST_MODE, SAMPLE_MODE, EVENT_MODE);

-- Defines the operating modes of an input device.

-- OPERATING_STATE                                       LEVEL ma

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 ma

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

-- Provides a range of character expansion factors.

-- RASTER_UNITS                                              LEVEL ma

type RASTER_UNITS is new POSITIVE;

-- Defines the range of raster units.

-- RASTER_UNIT_SIZE                                          LEVEL ma

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 ma

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 mb

type STRING_PROMPT_ECHO_TYPE is new INTEGER;

-- Defines the string prompt and echo types.

-- STRING_PROMPT_ECHO_TYPES                                 LEVEL mb

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 mb

type STROKE_PROMPT_ECHO_TYPE is new INTEGER;

-- Defines the stroke prompt and echo types.

-- STROKE_PROMPT_ECHO_TYPES                                   LEVEL mb

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 ma

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 ma

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 m

type WC_TYPE is digits PRECISION;

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

-- WC                                                         LEVEL ma

package WC is new GKS_COORDINATE_SYSTEM (WC_TYPE);

-- Defines the World Coordinate System.

-- TEXT_EXTENT_PARALLELOGRAM                                  LEVEL ma

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 ma

type TEXT_FONT is new INTEGER;

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

-- TEXT_PRECISION                                               LEVEL ma

type TEXT_PRECISION is  (STRING_PRECISION,
                         CHAR_PRECISION,
                         STROKE_PRECISION);

-- The precision with which text appears.

-- TEXT_FONT_PRECISION                                          LEVEL ma

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 ma

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 ma

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 ma

type TRANSFORMATION_NUMBER is new NATURAL;

-- A normalization transformation number.

-- POSITIVE_TRANSFORMATION_NUMBER                           LEVEL ma

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

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

-- TRANSFORMATION_PRIORITY_ARRAY                            LEVEL ma

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

-- Type to store transformation numbers.

-- TRANSFORMATION_PRIORITY_LIST                             LEVEL ma

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 mb

type VALUATOR_INPUT_VALUE is digits PRECISION;

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

-- VALUATOR_PROMPT_ECHO_TYPE                                       LEVEL mb

type VALUATOR_PROMPT_ECHO_TYPE is new INTEGER;

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

-- VALUATOR_PROMPT_ECHO_TYPES                                      LEVEL mb

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 ma

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 ma

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 ma

type WS_ID is new POSITIVE;

-- Defines the range of workstation identifiers.

-- WS_IDS                                                          LEVEL ma

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 ma

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 ma

package WS_TYPES is new GKS_LIST_UTILITIES (WS_TYPE);

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

```
-- INDIVIDUAL_ATTRIBUTE_VALUES                                    LEVEL ma

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                                     LEVEL ma

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 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 mb

    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 for locator prompt and echo
--  types 1, 2, 3, and 6.

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

-- Constructs and returns a locator data record for locator prompt and echo
--  type 5 when its attributes are specified by Polyline attributes.

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

-- Constructs and returns a locator data record for locator prompt and echo
--  type 5 when its attributes are specified by Fill Area attributes.

function ATTRIBUTE_FLAG
          (DATA_RECORD  : in LOCATOR_DATA_RECORD)
 return ATTRIBUTES_FLAG;

-- Returns the attribute flag CURRENT or SPECIFIED stored in the
-- data record for the prompt and echo types 4 and 5.
```

```
function LOCATOR_ATTRIBUTES_USED
        (DATA_RECORD : in LOCATOR_DATA_RECORD)
 return ATTRIBUTES_USED_TYPE;

-- Returns which attribute set, either Polyline or Fill Area, stored in the data record
-- for prompt and echo types 4 and 5.

function LINE_ATTRIBUTES
         (DATA_RECORD : in LOCATOR_DATA_RECORD)
  return LINE_DATA;

-- Returns the Polyline attribute information stored in the data record for prompt and echo types 4 or 5.

function FILL_AREA_ATTRIBUTES
         (DATA_RECORD : in LOCATOR_DATA_RECORD)
  return FILL_AREA_DATA;

-- Returns the Fill Area attribute information stored in the data record for prompt and echo type 5.

-- Stroke Data Record Operations.

procedure BUILD_STROKE_DATA_RECORD
     (PROMPT_ECHO_TYPE  : in STROKE_PROMPT_ECHO_TYPE;
      BUFFER_SIZE       : in POSITIVE;
      POSITION          : in POSITIVE;
      INTERVAL          : in WC.SIZE;
      TIME              : in DURATION;
      DATA_RECORD       : out STROKE_DATA_RECORD);

-- Constructs and returns a stroke data record for stroke prompt and echo types 1 and 2.

procedure BUILD_STROKE_DATA_RECORD
     (PROMPT_ECHO_TYPE  : in STROKE_PROMPT_ECHO_TYPE;
      BUFFER_SIZE       : in POSITIVE;
      POSITION          : in POSITIVE;
      INTERVAL          : in WC.SIZE;
      TIME              : in DURATION;
      CONTENTS          : in MARKER_DATA;
      DATA_RECORD       : out STROKE_DATA_RECORD);

-- Constructs and returns a stroke data record for stroke prompt and echo type 3.

procedure BUILD_STROKE_DATA_RECORD
     (PROMPT_ECHO_TYPE  : in STROKE_PROMPT_ECHO_TYPE;
      BUFFER_SIZE       : in POSITIVE;
      POSITION          : in POSITIVE;
      INTERVAL          : in WC.SIZE;
      TIME              : in DURATION;
      CONTENTS          : in LINE_DATA;
      DATA_RECORD       : out STROKE_DATA_RECORD);

-- Constructs and returns a stroke data record for stroke prompt and echo type 4.
```

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

-- Returns the size of the input stroke buffer stored in the data record for prompt and
-- echo types 1, 2, 3 and 4.

function POSITION
            (DATA_RECORD : in STROKE_DATA_RECORD)
 return  POSITIVE;

-- Returns the editing position within the input stroke buffer stored in the data record
-- for prompt and echo types 1, 2, 3 and 4.

function INTERVAL
        (DATA_RECORD : in STROKE_DATA_RECORD)
 return  WC.SIZE;

-- Returns the interval value stored the stroke buffer stored  data record for prompt and
-- echo types 1, 2, 3 and 4.

function TIME
        (DATA_RECORD : in STROKE_DATA_RECORD)
 return  DURATION;

-- Returns the measuring time for sampling stroke input stored in the data record for prompt and
-- echo types 1, 2, 3 and 4.

function MARKER_ATTRIBUTES
        (DATA_RECORD : in STROKE_DATA_RECORD)
 return  MARKER_DATA;

-- Returns the Polymarker attributes used to echo the stroke input stored in the data record
-- for prompt and echo type 3.

function LINE_ATTRIBUTES
        (DATA_RECORD : in STROKE_DATA_RECORD)
 return  LINE_DATA;

-- Returns the Polyline attributes used to echo the stroke input stored in the data record for
-- prompt and echo type 4.

-- 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 for the valuator prompt and echo  types 1, 2, and 3.
```

```
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 for
-- prompt and echo types 1, 2, and 3.

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 for
-- prompt and echo types 1, 2, and 3.

-- 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 for choice prompt and echo type 1.

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

-- Constructs and returns a choice data record for choice prompt and echo type 2.

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

-- Constructs and returns a choice data record for choice prompt and echo types 3 and 4.

procedure BUILD_CHOICE_DATA_RECORD
      (PROMPT_ECHO_TYPE  : in CHOICE_PROMPT_ECHO_TYPE;
       SEGMENT           : in SEGMENT_NAME;
       LIST_OF_PICK_IDS  : in PICK_IDS.LIST_OF;
       DATA_RECORD       : out CHOICE_DATA_RECORD);

-- Constructs and returns a choice data record for choice prompt and echo types 5.

function ARRAY_OF_PROMPTS
        (DATA_RECORD : in CHOICE_DATA_RECORD)
 return  CHOICE_PROMPTS.LIST_OF;

-- Returns the array of prompts stored in the choice data record for
-- prompt and echo type 2.

function ARRAY_OF_STRINGS
        (DATA_RECORD : in CHOICE_DATA_RECORD)
 return  CHOICE_PROMPT_STRING_LIST;

-- Returns the array of prompt strings stored in the choice data record for  prompt and echo types 3 and 4.
```

```
function SEGMENT
        (DATA_RECORD : in CHOICE_DATA_RECORD)
 return  SEGMENT_NAME;

-- Returns the segment name stored in the choice data record for  prompt and echo type 5.

function LIST_OF_PICK_IDS
        (DATA_RECORD : in CHOICE_DATA_RECORD)
 return  PICK_IDS.LIST_OF;

-- Returns the list of pick id's  stored in the choice data record for  prompt and echo type 5.

-- 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 POSTIVE_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 REQUEST_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;
       MAXIMUM_LENGTH           : in MAX_LENGTH : 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 EROR_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;
       PIC                 : 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 EROR_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_NAME     : in ESCAPE_ID;
                                     ESC_DATA_IN     : in ESC_DATA_RECORD;
                                     ESC_DATA_OUT    : out ESC_DATA_RECORD);
end GKS_ESCAPE;
```

Appendix B
Cross Reference Listing of Implementation Defined Items

(This Appendix 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

Appendix C
Example Programs

(This Appendix is not an integral part of the standard, but provides additional information.)

This Appendix gives complete programs using the language binding defined in this standard.

C.1 Example Program 1 : STAR

```
--  PROGRAM STAR

-- DESCRIPTION:
--       This program draws a yellow star on a blue background and writes the
--       title 'STAR' in green under the star.
-- CONFORMANCE:
--       GKS Level:  0a
--       The implementation must support at least one workstation of category  output or outin.

with GKS;
with GKS_TYPES;

use GKS;
use GKS_TYPES;

procedure STAR is

        -- Define the Workstation variables and error logging file.

        MY_WS_ID              : constant WS_ID          := 1;
        SOME_CONNECTION       : constant STRING         := "UNIT_1";
        SOME_OUTPUT_TYPE      : constant WS_TYPE        := 1;
        ERROR_FILE            : constant STRING         := "MY_ERROR_FILE";

        -- Define the points of the Star.

        STAR_POINTS           : constant WC.POINT_ARRAY :=
                                          (( 0.951057, 0.309017),
                                          (-0.951057, 0.309017),
                                          ( 0.587785,-0.951057),
                                          ( 0.0      , 1.0),
                                          (-0.587785,-0.951057));

        -- Define World Coordinate Window and miscellaneous attributes.

        WINDOW                : WC.RECTANGLE_LIMITS :=
                                           (XMIN => -1.25, XMAX => 1.25,
                                           YMIN => -1.25,  YMAX => 1.25);
        TEXT_POSITION         : WC.POINT := (0.0,-1.0);

        begin

        -- Open GKS and activate a workstation.

        OPEN_GKS      (ERROR_FILE);
        OPEN_WS       (MY_WS_ID, SOME_CONNECTION, SOME_OUTPUT_TYPE);
        ACTIVATE_WS   (MY_WS_ID);
```

```
        -- Center the window around the origin.

        SET_WINDOW    (1, WINDOW);
        SELECT_NORMALIZATION_TRANSFORMATION (1);

        -- Define the colours.

        SET_COLOUR_REPRESENTATION    (WS => MY_WS_ID,
                                      INDEX => 0,
                                      RGB_COLOUR => (0.0,0.0,1.0));

        SET_COLOUR_REPRESENTATION    (WS => MY_WS_ID,
                                      INDEX => 1,
                                      RGB_COLOUR => (1.0,1.0,0.0));

        SET_COLOUR_REPRESENTATION    (WS => MY_WS_ID,
                                      INDEX => 2,
                                      RGB_COLOUR => (1.0,1.0,1.0));

        -- Set Fill Area attributes.

        SET_FILL_AREA_INTERIOR_STYLE (SOLID);
        SET_FILL_AREA_COLOUR_INDEX (1);

        -- Draw the star.

        FILL_AREA  (STAR_POINTS);

        -- Select large characters centered under the star.

        SET_CHAR_HEIGHT              (HEIGHT => 0.15);
        SET_TEXT_ALIGNMENT           (ALIGNMENT => (CENTRE, HALF));
        SET_TEXT_COLOUR_INDEX        (TEXT_COLOUR => 2);

        -- Draw the title.

        TEXT (TEXT_POSITION, "STAR");

        -- Close the workstation and shut down GKS.

        DEACTIVATE_WS    (MY_WS_ID);
        CLOSE_WS         (MY_WS_ID);
        CLOSE_GKS;

end STAR;
```

C.2 Example Program 2 : IRON

```
-- PROGRAM IRON

-- DESCRIPTION:
--      This program draws a horizontal bar chart illustrating costs within the iron industry.
--      The user can select the data to be displayed using a GKS choice device. The plot is
--      adapted from "Scientific American' May 1984, page 139.
-- CONFORMANCE:
--      GKS Level: 2b

with GKS;
with GKS_TYPES;

use GKS;
use GKS_TYPES;

procedure IRON is

    -- Define the Workstation variables and error logging file.

    MY_WS_ID                : constant WS_ID                    := 1;
    SOME_CONNECTION         : constant STRING                   := "TTY";
    SOME_OUTIN_TYPE         : constant WS_TYPE                  := 2;
    ERROR_FILE              : constant STRING                   :=
                                    "MY_ERROR_FILE";

    -- Declare and initialise aspect source flags (use BUNDLES-- for fill interior;
    -- others set to INDIVIDUAL).

    ASF_SETTINGS            : ASF_LIST :=
        (TYPE_OF_LINE_ASF           => INDIVIDUAL,
        WIDTH_ASF                   => INDIVIDUAL,
        LINE_COLOUR_ASF             => INDIVIDUAL,
        TYPE_OF_MARKER_ASF          => INDIVIDUAL,
        SIZE_ASF                    => INDIVIDUAL,
        MARKER_COLOUR_ASF           => INDIVIDUAL,
        FONT_PRECISION_ASF          => INDIVIDUAL,
        EXPANSION_ASF               => INDIVIDUAL,
        SPACING_ASF                 => INDIVIDUAL,
        TEXT_COLOUR_ASF             => INDIVIDUAL,
        INTERIOR_ASF                => BUNDLED,
        STYLE_ASF                   => BUNDLED,
        FILL_AREA_COLOUR_ASF        => INDIVIDUAL);

    -- Declare and initialise objects for choice device.

    CHOICE_STRING_COUNT     : constant                          := 3;
    CHOICE_DEVICE           : constant CHOICE_DEVICE_NUMBER     := 1;
    CHOICE_ERROR            : ERROR_NUMBER;
    CHOICE_MODE             : OPERATING_MODE;
    CHOICE_ECHO_SWITCH      : ECHO_SWITCH;
    INITIAL_CHOICE          : CHOICE_VALUE;
    CHOICE_INPUT_RECORD     : CHOICE_DATA_RECORD;
    CHOICE_ECHO_AREA        : DC.RECTANGLE_LIMITS;
```

```
   PROMPT_ECHO_TYPE          : constant CHOICE_PROMPT_ECHO_TYPE      := 3;
   CHOICE_STRINGS            : constant CHOICE_PROMPT_STRING_LIST    :=
                                        (LENGTH=> 3, LIST =>
                                        ((4,"U.S."), (9,"W.GERMANY"),
                                        (5,"JAPAN")));
   CHOICE_RECORD             : CHOICE_DATA_RECORD;
   CHOICE                    : CHOICE_VALUE;
   CHOICE_REQUEST            : CHOICE_REQUEST_STATUS;
   INITIAL_STATUS            : CHOICE_STATUS;

   -- Declare colour objects.

   RGB_COLOUR                : COLOUR_REPRESENTATION;
   WHITE                     : constant COLOUR_INDEX                 := 0;
   BLACK                     : constant COLOUR_INDEX                 := 1;
   RED                       : constant COLOUR_INDEX                 := 2;


   -- Declare and initialise window objects.

   WINDOW_1                  : TRANSFORMATION_NUMBER                 := 1;
   WINDOW_LIMITS             : WC.RECTANGLE_LIMITS                   :=
                                        (XMIN => -100.0, XMAX => 175.0,
                                         YMIN => -2.0,   YMAX =>  13.0);

   -- Declare and initialise object for bar data to plot.

   MAX_DATA                  : constant                              := 6;
   type IRON_DATA is array (1 .. MAX_DATA) of WC_TYPE;
   US_DATA_1                 : IRON_DATA   := (69.0, 50.0, 15.0, 53.0, 57.0, 150.0);
   US_DATA_2                 : IRON_DATA   := (72.0, 50.0, 103.0, 0.0, 0.0, 56.0);
   GERMANY_DATA_1            : IRON_DATA   := (65.0, 42.0, 3.0, 89.0, 52.0, 93.0);
   GERMANY_DATA_2            : IRON_DATA   := (70.0, 53.0, 102.0, 0.0, 0.0, 49.0);
   JAPAN_DATA_1              : IRON_DATA   := (65.0, 47.0, 2.0, 60.0, 52.0, 55.0);
   JAPAN_DATA_2              : IRON_DATA   := (70.0, 57.0, 105.0, 0.0, 0.0, 41.0);

procedure BARS   (LENGTH     : in WC_TYPE;
                  POSITION   : in WC.POINT) is

LEFT_HALF     : TEXT_ALIGNMENT := (LEFT,HALF);
BAR_POINTS    : WC.POINT_ARRAY (1..4);

begin

  if LENGTH = 0.0 then
    SET_TEXT_ALIGNMENT (LEFT_HALF);
    TEXT (POSITION,"0");
  else
      BAR_POINTS  :=(
            (X => 0.0,                          Y => POSITION.Y + 0.4),
            (X => LENGTH,                       Y => POSITION.Y + 0.4),
            (X => LENGTH,                       Y => POSITION.Y - 0.4),
            (X => 0.0,                          Y => POSITION.Y - 0.4));
    FILL_AREA (BAR_POINTS);
  end if;

end BARS;
```

```
procedure TICKS (TICK_MARK_POSITION : in out WC.POINT_ARRAY) is

TICK_MARK_LABEL_POSITION : WC.POINT;

begin

for I in 1..4 loop
  POLYLINE (TICK_MARK_POSITION);
  TICK_MARK_POSITION (1).X := TICK_MARK_POSITION(1).X + 50.0;
  TICK_MARK_POSITION (2).X := TICK_MARK_POSITION(2).X + 50.0;
end loop;

-- Draw tick mark labels.

TICK_MARK_LABEL_POSITION.X := 0.0;
TICK_MARK_LABEL_POSITION.Y := WC_TYPE (TICK_MARK_POSITION(1).Y);
TEXT (TICK_MARK_LABEL_POSITION, "0");

TICK_MARK_LABEL_POSITION.X := 50.0;
TEXT (TICK_MARK_LABEL_POSITION, "50");

TICK_MARK_LABEL_POSITION.X := 100.0;
TEXT (TICK_MARK_LABEL_POSITION, "100");

TICK_MARK_LABEL_POSITION.X := 150.0;
TEXT (TICK_MARK_LABEL_POSITION, "150");

 end TICKS;

procedure BORDER is

-- Draws the border surrounding the data.

LABEL_POSITION                : WC.POINT;
TITLE_POSITION                : WC.POINT := (37.5, -2.0);
HEIGHT                        : WC.MAGNITUDE := 0.5;
LEFT_HALF                     : TEXT_ALIGNMENT := (LEFT, HALF);
CENTRE_BOTTOM                 : TEXT_ALIGNMENT := (CENTRE, BOTTOM);
CENTRE_CAP                    : TEXT_ALIGNMENT := (CENTRE, CAP);
ONLY_IF_NOT_EMPTY             : CONTROL_FLAG := CONDITIONALLY;
BOX_POINTS                    : constant WC.POINT_ARRAY :=(
                                  (0.0,0.0), (150.0,0.0), (150.0,12.0),
                                  (0.0,12.0), (0.0, 0.0));
type LABELS is array (1..6) of INPUT_STRING;
BAR_LABELS                    : constant LABELS := (
                                  (5,"LABOR"), (8,"IRON ORE"),
                                  (12,"COKE OR COAL"),
                                  (15,"PURCHASED SCRAP"),  (11, "OTHER COSTS"),
                                  (12,"OTHER ENERGY"));
TOP_TICK_MARK_START           : WC.POINT_ARRAY(1..2) := (
                                  (0.0,12.0), (0.0,11.9));
BOTTOM_TICK_MARK_START        : WC.POINT_ARRAY(1..2) := (
                                  (0.0,0.0), (0.0,0.1));
```

```
begin  -- Procedure BORDER.

CLEAR_WS (MY_WS_ID,ONLY_IF_NOT_EMPTY);

--  Draw the box surrounding the chart area.

POLYLINE (BOX_POINTS);

--  Draw the bar labels centered on the bar and flush left.

SET_TEXT_ALIGNMENT (LEFT_HALF);
SET_CHAR_HEIGHT (HEIGHT);
SET_TEXT_COLOUR_INDEX (BLACK);

LABEL_POSITION.X := -99.0;
for I in 1..6 loop
   LABEL_POSITION.Y := WC_TYPE(2.0 * (FLOAT(I)-1.0) + 1.2);
   TEXT (LABEL_POSITION, BAR_LABELS(INTEGER(I)).CONTENTS);
end loop;

--  Draw the top and bottom tick marks (bottom in red).

SET_TEXT_ALIGNMENT (CENTRE_BOTTOM);

--  Call procedures to draw ticks.

TICKS (TOP_TICK_MARK_START);

SET_TEXT_ALIGNMENT (CENTRE_CAP);
SET_TEXT_COLOUR_INDEX (RED);
TICKS (BOTTOM_TICK_MARK_START);

--  Draw the title.

SET_TEXT_COLOUR_INDEX (BLACK);
SET_TEXT_ALIGNMENT (CENTRE_BOTTOM);
TEXT (TITLE_POSITION, "PRODUCTION COST");

end BORDER;

procedure DRAW     (DATA1        : in out IRON_DATA;
                    DATA2        : in out IRON_DATA) is

FILL_INDEX      : FILL_AREA_INDEX :=1;
POSITION        : WC.POINT;

begin

--  Draw the border.

BORDER;

--  Draw the black bars.

SET_FILL_AREA_COLOUR_INDEX (BLACK);
SET_TEXT_COLOUR_INDEX (BLACK);
SET_FILL_AREA_INDEX (FILL_INDEX);
```

```
        for I in 1..6 loop
          POSITION.Y := 2.0 * (WC_TYPE(I)-1.0) + 1.6;
          -- Call the procedure that draws the bars
          BARS (DATA1(INTEGER(I)), POSITION);
        end loop;
        -- Draw the red bars.
        SET_FILL_AREA_COLOUR_INDEX (RED);
        SET_TEXT_COLOUR_INDEX (RED);
        FILL_INDEX := 2;
        SET_FILL_AREA_INDEX (FILL_INDEX);
        for I in 1..6 loop
          POSITION.Y := 2.0 * (WC_TYPE(I)-1.0) + 1.6;
          -- Call the procedure that draws the bars.
          BARS (DATA2(INTEGER(I)), POSITION);
        end loop;

        end DRAW;


begin  -- Procedure IRON.

        -- Open GKS and activate a workstation.

        OPEN_GKS (ERROR_FILE);
        OPEN_WS (MY_WS_ID,SOME_CONNECTION,SOME_OUTIN_TYPE);
        ACTIVATE_WS (MY_WS_ID);

        -- Specify the window onto chart.

        SET_WINDOW (WINDOW_1, WINDOW_LIMITS);
        SELECT_NORMALIZATION_TRANSFORMATION (WINDOW_1);

        -- Define the colours we'll be using.

          SET_COLOUR_REPRESENTATION      (MY_WS_ID,
                                          INDEX => WHITE,
                                          RGB_COLOUR =>(1.0,1.0,1.0));
          SET_COLOUR_REPRESENTATION      (MY_WS_ID,
                                          INDEX => BLACK,
                                          RGB_COLOUR =>(0.0,0.0,0.0));
          SET_COLOUR_REPRESENTATION      (MY_WS_ID,
                                          INDEX => RED,
                                          RGB_COLOUR =>(1.0,0.0,0.0));

        -- Use bundled attributes except for colour.

        SET_ASF (ASF_SETTINGS);
```

```
        -- Initialise the choice device.

        INQ_CHOICE_DEVICE_STATE     (MY_WS_ID, CHOICE_DEVICE, CHOICE_ERROR,
                                     CHOICE_MODE, CHOICE_ECHO_SWITCH,
                                     INITIAL_STATUS, INITIAL_CHOICE,
                                     CHOICE_ECHO_AREA, CHOICE_RECORD);

        BUILD_CHOICE_DATA_RECORD    (PROMPT_ECHO_TYPE, CHOICE_STRINGS,
                                     CHOICE_RECORD);

        INITIALISE_CHOICE    (MY_WS_ID, CHOICE_DEVICE, INITIAL_STATUS,
                              INITIAL_CHOICE, CHOICE_ECHO_AREA, CHOICE_RECORD);

        -- Get the user's choice (U.S. , W. GERMANY, or JAPAN).

        loop
          REQUEST_CHOICE   (MY_WS_ID, CHOICE_DEVICE,
                            INITIAL_STATUS, CHOICE);
          if INITIAL_STATUS = OK then
            case CHOICE is
                when 1     =>  DRAW (US_DATA_1,US_DATA_2);
                when 2     =>  DRAW (GERMANY_DATA_1,GERMANY_DATA_2);
                when 3     =>  DRAW (JAPAN_DATA_1,JAPAN_DATA_2);
                when others =>  exit;
            end case;
          else
            exit;
          end if;
        end loop;

        -- Close workstation and shut down GKS.

        DEACTIVATE_WS   (MY_WS_ID);
        CLOSE_WS        (MY_WS_ID);
        CLOSE_GKS;

end IRON;
```

C.3 Example Program 3 : MAP

```
-- PROGRAM MAP

-- DESCRIPTION:
--      This program reads a GKS metafile to draw a map. The primitives in each region are in a
--      separate segment. The user can use a pick device to select the various regions. A sampled
--      choice device determines the action taken with the selected region.
-- CONFORMANCE:
--      GKS Level: 1c
--      The implementation must support at least on workstation of the category OUTIN and one of the
--      category MI (metafile input). The default choice device must support at least five choices.


with GKS;
with GKS_TYPES;

use GKS;
use GKS_TYPES;

procedure METAFILE is

        -- Define the Metafile Workstation

        METAFILE_WS_ID          : constant WS_ID                    := 1;
        METAFILE_CONNECTION     : constant STRING                   :=
                                        "METAFILE_INPUT_FILE";
        METAFILE_TYPE           : constant WS_TYPE                  := 2;
        METAFILE_ITEM_TYPE      : GKSM_ITEM_TYPE;
        METAFILE_DATA_RECORD    : GKSM_DATA_RECORD;
        LENGTH, MAX_LENGTH      : NATURAL                           := 500;

        -- Define the OUTIN Workstation

        MY_WS_ID                : constant WS_ID                    := 2;
        SOME_CONNECTION         : constant STRING                   := "UNIT_2";
        SOME_OUTIN_TYPE         : constant WS_TYPE                  := 1;
        SOME_CHOICE_DEVICE      : constant CHOICE_DEVICE_NUMBER     := 1;
        CSTATUS                 : CHOICE_STATUS;
        CHOICE_NUMBER           : CHOICE_VALUE;
        SOME_PICK_DEVICE        : constant PICK_DEVICE_NUMBER       := 1;
        PSTATUS                 : PICK_REQUEST_STATUS;
        PICK                    : PICK_ID;
        SEGMENT                 : SEGMENT_NAME;

        -- Define the Error Logging file.

        ERROR_FILE              : constant STRING  :=  "MY_ERROR_FILE";

        begin

        -- Open GKS and activate workstations.

        OPEN_GKS        (ERROR_FILE);
        OPEN_WS         (METAFILE_WS_ID,METAFILE_CONNECTION,METAFILE_TYPE);
        OPEN_WS         (MY_WS_ID,SOME_CONNECTION,SOME_OUTIN_TYPE);
        ACTIVATE_WS     (MY_WS_ID);
```

```
        -- Set the choice device to sample mode.

        SET_CHOICE_MODE (MY_WS_ID, SOME_CHOICE_DEVICE, SAMPLE_MODE, NOECHO);

        -- Interpret metafile items until end of metafile is read.

        loop
          GET_ITEM_TYPE_FROM_GKSM   (METAFILE_WS_ID, METAFILE_ITEM_TYPE,
                                     LENGTH);

          if METAFILE_ITEM_TYPE = 0 then
            exit;
          end if;
          READ_ITEM_FROM_GKSM   (METAFILE_WS_ID, MAX_LENGTH,
                                 METAFILE_DATA_RECORD);

          INTERPRET_ITEM (METAFILE_DATA_RECORD);
        end loop;

        -- Close the Metafile Workstation.
        CLOSE_WS (METAFILE_WS_ID);

        -- Allow the user to select states until the 'exit' choice.

        loop
          REQUEST_PICK (MY_WS_ID,SOME_PICK_DEVICE,PSTATUS,SEGMENT,PICK);
          if PSTATUS = OK then
            SAMPLE_CHOICE   (MY_WS_ID, SOME_CHOICE_DEVICE,
                             CSTATUS, CHOICE_NUMBER);
            if CSTATUS = OK then
                case CHOICE_NUMBER is
                  when 1 => SET_HIGHLIGHTING     (SEGMENT,HIGHLIGHTED);
                  when 2 => SET_HIGHLIGHTING     (SEGMENT,NORMAL);
                  when 3 => SET_VISIBILITY       (SEGMENT,INVISIBLE);
                  when 4 => SET_VISIBILITY       (SEGMENT,VISIBLE);
                  when others => null;
                end case;
            end if;
          end if;
        end loop;

        -- Close Workstations and GKS.

        DEACTIVATE_WS   (MY_WS_ID);
        CLOSE_WS        (MY_WS_ID);
        CLOSE_GKS;

end METAFILE;
```

C.4 Example Program 4 : MANIPULATE

```
-- PROGRAM MANIPULATE

-- DESCRIPTION:
--      This program allows the user to create an object and then manipulate the
--      object by changing the segment transformation.
-- CONFORMANCE:
--      GKS Level 2b

with GKS;
with GKS_TYPES;

use GKS;
use GKS_TYPES;

procedure POLYGON is

        POINTS                  : WC.POINT_ARRAY(I..500);
        POINT_1                 : WC.POINT                  := (0.6,0.4);
        POINT_2                 : WC.POINT                  := (0.4,0.3);
        POLYGON_SEGMENT         : SEGMENT_NAME              := I;
        SHIFT                   : constant CHOICE_VALUE     := I;
        ZOOM                    : constant CHOICE_VALUE     := 2;
        ROTATE                  : constant CHOICE_VALUE     := 3;
        NEXT                    : POSITIVE      := I;
        TRANSFORMATION          : TRANSFORMATION_NUMBER;
        TRANSFORMATION_1        : TRANSFORMATION_NUMBER;
        TRANSFORMATION_2        : TRANSFORMATION_NUMBER;
        RED                     : constant COLOUR_INDEX     := 2;
        AXIS_CHARACTER_HEIGHT   : constant WC.MAGNITUDE     := 0.02;
        CHOICE                  : CHOICE_VALUE;
        CHOICE_STATUS           : CHOICE_REQUEST_STATUS;
        LOCATOR_STATUS          : INPUT_STATUS;
        MATRIX                  : TRANSFORMATION_MATRIX;
        MATRIX_RESULT           : TRANSFORMATION_MATRIX;

        -- Define the workstation variables and error logging file.

        DISPLAY                 : constant WS_ID            := 1;
        DISPLAY_CONNECTION      : constant STRING           := "DDDIS";
        DISPLAY_TYPE            : constant WS_TYPE          := 3;
        ERROR_FILE              : constant STRING           := "MY_ERROR_FILE";

        -- Define the Segment workstation variables.

        SEGSTORE                : constant WS_ID            := 2;
        SEG_CONNECTION          : constant STRING           := "DDSEG";
        SEG_TYPE                : constant WS_TYPE          := 4;

        -- Define the Plotter workstation variable.

        PLOTTER                 : constant WS_ID            := 6;
        PLOT_CONNECTION         : constant STRING           := "PLOT";
        PLOT_TYPE               : constant WS_TYPE          := 5;
```

```
-- Define World Coordinate Window and other attributes.

WINDOW_BOUNDS      : WC.RECTANGLE_LIMITS :=
                          (XMIN => 0.0, XMAX => 1.0,
                          YMIN => 0.0, YMAX => 1.0);
VIEWPORT_BOUNDS    : NDC.RECTANGLE_LIMITS :=
                          (XMIN => 0.0, XMAX => 1.0,
                          YMIN => 0.0, YMAX => 1.0);
TEXT_POSITION      : WC.POINT := (0.5,0.5);

begin

-- Open GKS and activate a workstation.

OPEN_GKS                          (ERROR_FILE);
OPEN_WS                           (DISPLAY, DISPLAY_CONNECTION,
                                  DISPLAY_TYPE);
ACTIVATE_WS                       (DISPLAY);
OPEN_WS                           (SEGSTORE, SEG_CONNECTION, SEG_TYPE);
ACTIVATE_WS                       (SEGSTORE);
SET_WINDOW                        (1,WINDOW_BOUNDS);
SET_VIEWPORT                      (1,VIEWPORT_BOUNDS);
SET_VIEWPORT_INPUT_PRIORITY       (1,0,HIGHER);

--Construction of segment POLYGON_SEGMENT.

CREATE_SEGMENT        (POLYGON_SEGMENT);
SET_POLYLINE_INDEX    (3);
REQUEST_LOCATOR       (DISPLAY,1,LOCATOR_STATUS,
                      TRANSFORMATION,POINTS(NEXT));

SELECT_NORMALIZATION_TRANSFORMATION (TRANSFORMATION_1);
loop
  NEXT:=NEXT+1;
  REQUEST_LOCATOR(DISPLAY,1,LOCATOR_STATUS,
  TRANSFORMATION,POINTS(NEXT));
  exit when LOCATOR_STATUS = NONE or
    TRANSFORMATION /= TRANSFORMATION_1 or
    NEXT = 500;
end loop;
POINTS(NEXT) := POINTS(1);
POLYLINE (POINTS);
CLOSE_SEGMENT;

EVALUATE_TRANSFORMATION_MATRIX    (WC.POINT'((0.0,0.0)),
                                  WC.VECTOR'((0.0,0.0)), 0.0, (1.0,1.0),
                                  MATRIX);
```

```
        -- Initialise transformation matrix.

        loop
          REQUEST_CHOICE (DISPLAY,1,CHOICE_STATUS,CHOICE);
          exit when CHOICE_STATUS = NONE or CHOICE_STATUS = NOCHOICE;
          case CHOICE is
            -- Shift the polygon to a given position.
            when SHIFT =>
                REQUEST_LOCATOR (DISPLAY,1,LOCATOR_STATUS,
                    TRANSFORMATION_2,POINT_1);
            exit when LOCATOR_STATUS = NONE;
                REQUEST_LOCATOR (DISPLAY,1,LOCATOR_STATUS,
                    TRANSFORMATION,POINT_2);
            exit when LOCATOR_STATUS=NONE or
              TRANSFORMATION /= TRANSFORMATION_2;
                SELECT_NORMALIZATION_TRANSFORMATION(TRANSFORMATION_2);
                ACCUMULATE_TRANSFORMATION_MATRIX (
                    SOURCE_TRANSFORMATION          => MATRIX,
                    FIXED_POINT                    => WC.POINT'((0.0,0.0)),
                    SHIFT_VECTOR                   =>
                      WC.VECTOR'( (POINT_1.X - POINT_2.X,POINT_1.Y - POINT_2.Y)),
                    ROTATION_ANGLE                 => 0.0,
                    SCALE_FACTORS                  => (1.0,1.0),
                    RESULT_TRANSFORMATION          => MATRIX_RESULT);
                SET_SEGMENT_TRANSFORMATION (
                    POLYGON_SEGMENT,MATRIX_RESULT);
            when ZOOM          => null;
            when ROTATE        => null;
            when others        => exit;
          end case;
          UPDATE_WS (DISPLAY,PERFORM);
        end loop;

        -- Now the polygon is plotted.

        DEACTIVATE_WS (DISPLAY);
        DEACTIVATE_WS (SEGSTORE);
        OPEN_WS (PLOTTER, PLOT_CONNECTION, PLOT_TYPE);
        ACTIVATE_WS (PLOTTER);

        -- Set up representations for this workstation.

        SET_COLOUR_REPRESENTATION (PLOTTER,RED,(1.0,0.0,0.0));
        SET_POLYLINE_REPRESENTATION (PLOTTER,3,1,1.5,RED);
        SET_TEXT_REPRESENTATION (PLOTTER,2, (0,STRING_PRECISION), 1.0,0.0,RED);
        SET_WS_VIEWPORT (PLOTTER, (0.0, 0.5, 0.0, 0.5));
        COPY_SEGMENT_TO_WS (PLOTTER,POLYGON_SEGMENT);
        SET_TEXT_INDEX (2);
        SET_CHAR_HEIGHT (AXIS_CHARACTER_HEIGHT);
        TEXT ((0.5,0.5), "This is a polygon");

        DEACTIVATE_WS (PLOTTER);
        CLOSE_WS (PLOTTER);
        CLOSE_WS (DISPLAY);
        CLOSE_WS (SEGSTORE);
        CLOSE_GKS;
end POLYGON;
```

C.5 Example Program 5:

```
-- PROGRAM SHOWLN

-- DESCRIPTION:
--      This program illustrates the available linetypes on a user selected workstation.
--      It contains a typical GKS initialization routine and demonstrates how to program
--      subprograms which do not change any state list entries.
-- CONFORMANCE:
--      GKS Level 0a

with GKS_TYPES;
with GKS;
with TEXT_IO;
use GKS_TYPES;
use GKS;
use TEXT_IO;
procedure SHOWLN is

TYPE_OF_WS      : WS_TYPE;
ERROR_IND       : ERROR_NUMBER    := 0;
WORKSTATION     : WS_ID           := 1;
OP_STATE        : OPERATING_STATE;
package WS_TYPE_IO is new INTEGER_IO (WS_TYPE);

procedure INIT_GKS  (WTYPE    : in out WS_TYPE;
                     ERRIND   : in out ERROR_NUMBER) is

    -- GKS initialisation sequence.

    ERROR_FILE      : constant STRING  := 'SHOWLN_ERR_FILE';
    GKS_WS_TYPES    : WS_TYPES.LIST_OF;
    CATEGORY        : WS_CATEGORY;
    CONNECTION      : STRING (1..20);
    CONN_LENGTH     : NATURAL;

begin
    OPEN_GKS (ERROR_FILE);

    -- Inquire available workstation types and print them.
    INQ_LIST_OF_AVAILABLE_WS_TYPES (ERRIND, GKS_WS_TYPES);
    if ERRIND /= 0 then
      return;
    end if;

    PUT_LINE ("The available output and outin workstation types are:");
    for I in 1..WS_TYPES.SIZE_OF_LIST (GKS_WS_TYPES) loop
      INQ_WS_CATEGORY
        (WS_TYPES.LIST_ELEMENT (I, GKS_WS_TYPES), ERRIND, CATEGORY);
      if (CATEGORY = OUTPUT or CATEGORY = OUTIN) then
       WS_TYPE_IO.PUT (WS_TYPES.LIST_ELEMENT (I, GKS_WS_TYPES));
       PUT ( " " );
      end if;
    end loop;
    NEW_LINE;
```

```
    -- Choose one workstation to open and activate.

    PUT_LINE   ("Please enter connection identifier and workstation type");
    GET_LINE   (CONNECTION, CONN_LENGTH);
    WS_TYPE_IO.GET  (WTYPE);

    OPEN_WS          (WORKSTATION, CONNECTION (1..CONN_LENGTH), WTYPE);
    ACTIVATE_WS      (WORKSTATION);

    -- Check the operating state to ensure successful opening and activation
    INQ_OPERATING_STATE_VALUE (OP_STATE);
    if OP_STATE /= WSAC then
      ERRIND := 3;
      return;
    end if;

    ERRIND := 0;
end INIT_GKS;

procedure LINE_DEMO   (WTYPE    : in out  WS_TYPE;
                       ERRIND   : in out  ERROR_NUMER) is

        STATUS                  : UPDATE_STATE;
        REQ_WINDOW              : NDC.RECTANGLE_LIMITS;
        CUR_WINDOW              : NDC.RECTANGLE_LIMITS;
        REQ_VIEWPORT            : DC.RECTANGLE_LIMITS;
        CUR_VIEWPORT            : DC.RECTANGLE_LIMITS;
        LINETYPE_LIST           : LINETYPES.LIST _OF;
        NUM_WIDTHS              : NATURAL;
        NOMINAL_WIDTH           : DC.MAGNITUDE;
        RANGE_OF_WIDTHS         : DC.RANGE_OF_MAGNITUTES;
        NUM_INDICES             : NATURAL;
        LIST_OF_ASF             : ASF_LIST  := (others => INDIVIDUAL);
        SAVED_XFORM_NUM         : TRANSFORMATION_NUMBER;
        SAVED_PRIM_ATTR         : PRIMITIVE_ATTRIBUTE_VALUES;
        SAVED_INDV_ATTR         : INDIVIDUAL_ATTRIBUTE_VALUES
        DISTANCE                : NDC_TYPE;
        PTS                     : WC.POINT_ARRAY  (1..2);

begin

    -- Check the operating state.

    INQ_OPERATING_STATE_VALUE  (OP_STATE);
    if  (OP-STATE /=  WSAC  and OP_STATE  /=  SGOP) then
      ERRIND := 5;
      return;
    end if;

    -- Inquire workstation transformation.

    INQ_WS_TRANSFORMATION   (WORKSTATION, ERRIND, STATUS,
                             REQ_WINDOW, CUR_WINDOW,
                             REQ_VIEWPORT, CUR_VIEWPORT);
```

```
if ERRIND /= 0 then
  return;
end if;

-- Inquire polyline facilities.

INQ_POLYLINE_FACITIIES (WS_TYPE, ERRIND, LINETYPE_LIST,
                        NUM_WIDTHS, NOMINAL_WIDTH,
                        NOMINAL_WIDTH, RANGE_OF_WIDTHS,
                        NUM_INDICES);

if ERRIND /= 0 then
  return;
end if;

INQ_CURRENT_NORMALIZATION_TRANSFORMATION_NUMBER
                        (ERRIND, SAVED_PRIM_ATTR);
INQ_CURRENT_INDIVIDUAL_ATTRIBUTE_VALUES
                        (ERRIND, SAVE_INDV_ATTR);

-- Set unity normalization transformation number, individual aspect
-- source flags, linewidth scale factor (1.0), polyline colour index (1)
-- and reasonable text attributes.

SELECT_NORMALIZATION_TRANSFORMATION (0);
SET_ASF (LIST_OF_ASF);
SET_LINEWIDTH_SCALE_FACTOR (1.0);
SET_POLYLINE_COLOUR_INDEX (1);
SET_CHAR_UP_VECTOR ( (0.0,1.0) );
SET_TEXT_PATH (RIGHT);
SET_TEXT_ALIGNMENT( (LEFT, HALF) );
SET_TEXT_FONT_AND_PRECISION ((1, STRING_PRECISION) );
SET_CHAR_EXPANSION_FACTOR (1.0);
SET_CHAR_SPACING (0.0);
SET_TEXT_COLOUR_INDEX (1);

-- Compute the distance between lines.

DISTANCE := (CUR_WINDOW.YMAX - CUR_WINDOW.YMIN) /
            NDC_TYPE (LINETYPES.SIZE_OF_LIST (LINETYPE_LIST) );

-- Set the character height to half of the distance between the lines, but not
-- more than 1/20th of the height of the current workstation window.

if (DISTANCE/2.0) < ( (CUR_WINDOW.YMAX - CUR_WINDOW.YMIN) / 20.0
  then SET_CHAR_HEIGHT (WC.MAGNITUDE (DISTANCE/2.0) );
else
  SET_CHAR_HEIGHT
    (WC.MAGNITUDE ( (CUR_WINDOW.YMAX - CUR_WINDOW.YMIN) / 20.0) );
end if;

-- Lines stretch from the left bound to the middle of the current workstation window.

PTS (1).X := WC_TYPE (CUR_WINDOW.XMIN);
PTS (1).Y := WC_TYPE (CUR_WINDOW.YMAX - DISTANCE/2.0);
PTS (2).X := WC_TYPE (CUR_WINDOW.XMIN + CUR_WINDOW.XMAX / 2.0);
```

```
      -- Loop over the available linetypes.

      for I in 1..LINETYPES.SIZE_OF_LIST  (LINETYPE_LIST) loop
        SET_LINETYPE (LINETYPES.LIST_ELEMENT  (I, LINETYPE_LIST) );
        PTS (2).Y  := PTS (1).Y;
        POLYLINE (PTS);
        PTS (I).Y  := PTS (I).Y  -  WC_TYPE  (DISTANCE);

       -- Annotate the linetype.

       TEXT (PTS(2),  INTEGER'IMAGE (INTEGER (
          LINETYPES.LIST_ELEMENT (I, LINETYPE.LIST )  )  )  );
      end loop;

      -- Restore normalization transformation number and attributes.

      SELECT_NORMALIZATION_TRANSFORMATION (0);
      SET_ASF (SAVED_INDV_ATTR.ASF);
      SET_LINEWIDTH_SCALE_FACTOR (SAVED_INDV_ATTR.WIDTH);
      SET_POLYLINE_COLOUR_INDEX (SAVED_INDV_ATTR.LINE_COLOUR);
      SET_CHAR_UP_VECTOR (SAVED_PRIM_ATTR.CHAR_UP_VECTOR);
      SET_TEXT_PATH (SAVED_PRIM_ATTR.PATH);
      SET_TEXT_ALIGNMENT (SAVED_PRIM_ATTR.ALIGHMENT);
      SET_TEXT_FONT_AND_PRECISION  (SAVED_INDV_ATTR.FONT_PRECISION);
      SET_CHAR_EXPANSION_FACTOR (SAVED_INDV_ATTR.EXPANSION);
      SET_CHAR_SPACING (SAVED_INDV_ATTR.SPACING);
      SET_TEXT_COLOUR_INDEX (SAVED_INDV_ATTR.TEXT_COLOUR);

      ERRIND  :=  0;
end LINE_DEMO;

-- Main Procedure SHOWLN.

begin

      -- Call the initialization routine.
      INIT_GKS  (TYPE_OF_WS, ERROR_IND)
      if  ERROR_IND  =  0 then
        -- Call the demonstration subporgram for linetype capabilities
        LINE_DEMO  (TYPE_OF_WS,  ERROR_IND);
      end if;

      -- Close everything.
      EMERGENCY_CLOSE_GKS;
end SHOWLN;
```

Appendix D
GKS Multi-Tasking

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

The binding of GKS functions as subprograms in an Ada package has a straight-forward implementation: GKS "state" data are declared as variables local to the package body, and they are directly accessed and updated by the bodies of the GKS subprograms. This approach will work when application programs use only sequential control structures, the problem is that concurrent calls on GKS subprograms may cause a state variable to be corrupted; e.g., by simultaneous attempts to write to it. This problem exists whether the concurrency is actual (with multiple processors) or simulated (via multiplexed execution on a single processor).

There is an implementation technique that overcomes this problem without changing the GKS interface as seen by the Ada application program. In short, the idea is to protect the package body data (i.e., the state variables) by localizing them to a task contained in the package body. For each subprogram that accesses the data, there will be a corresponding entry declared in the task, The same name can be used for the entry and the subprogram, taking advantage of Ada's overloading facility. The task body takes the form of a "monitor" -- i.e., a loop containing a selective wait with an accept branch for each entry. The accept statement performs the actual reading or writing of the state information as required by the corresponding GKS subprogram. The body of each GKS subprogram comprises simply a call of the identically named task entry. Thus, even if two tasks from a user application program concurrently call subprograms that update or access state variables, these will result in entry calls that are queued in first come / first served fashion. There is no danger of corrupting the state variables.

To illustrate this technique, the following example shows how a skeletal version of the GKS package might be written.

```
with GKS_TYPES;
use GKS_TYPES;
package GKS is

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

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

        procedure CLOSE_GKS;

        ...

end GKS;


-- Version for sequential application programs:

with ERROR_HANDLING;
package body GKS is

        -- State variables:
        CURRENT_OPERATING_STATE    : OPERATING_STATE := GKCL;
        SET_OF_OPEN_WORKSTATIONS   : WS_IDS.LIST_OF  := WS_IDS.NULL_LIST;
```

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


        if CURRENT_OPERATING_STATE /= GKCL then
            ERROR_HANDLING (1, "OPEN_GKS");
        else
            CURRENT_OPERATING_STATE := GKOP;
        end if;
        ...

        end OPEN_GKS;

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

        if CURRENT_OPERATING_STATE not in GKOP .. SGOP then
            ERROR_HANDLING (8, "OPEN_WS");
        else
            CURRENT_OPERATING_STATE := WSOP;
            WS_IDS.ADD_TO_LIST (WS, SET_OF_OPEN_WORKSTATIONS);
        end if;
        ...

        end OPEN_WS;

        procedure CLOSE_GKS is
        begin
        ...

        if CURRENT_OPERATING_STATE /= GKOP then
            ERROR_HANDLING (2, "CLOSE_GKS");
        else
            CURRENT_OPERATING_STATE := GKCL;
        end if;
        ...

        end CLOSE_GKS;

        ...

end GKS;
```

```
-- Version for application programs that use tasking:

with ERROR_HANDLING;
package body GKS is

    task MONITOR is
    entry OPEN_GKS
        (ERROR_FILE             : in STRING     := DEFAULT_ERROR_FILE;
        AMOUNT_OF_MEMORY        : in NATURAL    := DEFAULT_MEMORY_UNITS);

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

    entry CLOSE_GKS;
    ...

    end MONITOR;

    task body MONITOR is

    -- State variables:
    CURRENT_OPERATING_STATE         : OPERATING_STATE       := GKCL;
    SET_OF_OPEN_WORKSTATIONS        : WS_IDS.LIST_OF        := WS_IDS.NULL_LIST;

    begin
        loop
            begin
                select
                    accept OPEN_GKS
                      (ERROR_FILE           : in STRING     := DEFAULT_ERROR_FILE;
                      AMOUNT_OF_MEMORY      : in NATURAL    := DEFAULT_MEMORY_UNITS) do
                        ...

                        if CURRENT_OPERATING_STATE /= GKCL then
                            ERROR_HANDLING (1, "OPEN_GKS");
                        else
                            CURRENT_OPERATING_STATE := GKOP;
                        end if;
                        ...

                    end OPEN_GKS;
                or
                    accept OPEN_WS
                      (WS                   : in WS_ID;
                      CONNECTION            : in STRING;
                      TYPE_OF_WS            : in WS_TYPE) do
                        ...

                        if CURRENT_OPERATING_STATE not in GKOP .. SGOP then
                          ERROR_HANDLING (8, "OPEN_WS");
                        else
                        CURRENT_OPERATING_STATE := WSOP;
                        WS_IDS.ADD_TO_LIST (WS, SET_OF_OPEN_WORKSTATIONS);
                        end if;
                        ...
```

```
                    end OPEN_WS;
                or
                    accept CLOSE_GKS do
                        ...

                        if CURRENT_OPERATING_STATE /= GKOP then
                          ERROR_HANDLING (2, "CLOSE_GKS");
                        else
                          CURRENT_OPERATING_STATE := GKCL;
                        end if;
                        ...

                    end CLOSE_GKS;
                    ...

                or
                    terminate;
                end select;
            exception
                when others => null;
            end;
        end loop;
    end MONITOR;

    procedure OPEN_GKS
        (ERROR_FILE                 : in STRING     := DEFAULT_ERROR_FILE;
        AMOUNT_OF_MEMORY    : in NATURAL := DEFAULT_MEMORY_UNITS) is
    begin
        MONITOR.OPEN_GKS (ERROR_FILE, AMOUNT_OF_MEMORY);
    end OPEN_GKS;

    procedure OPEN_WS
        (WS             : in WS_ID;
        CONNECTION      : in STRING;
        TYPE_OF_WS      : in WS_TYPE) is
    begin
        MONITOR.OPEN_WS (WS, CONNECION, TYPE_OF_WS);
    end OPEN_WS;

    procedure CLOSE_GKS is
    begin
        MONITOR.CLOSE_GKS;
    end CLOSE_GKS;

    ...

end GKS;
```

Some comments on the interactions with exception handling in the case where the ERROR_HANDLING procedure raises GKS_ERROR: note that in both versions of the package body, the ERROR_HANDLING procedure is called. Suppose that the application program calls OPEN_GKS when GKS is already open. In the sequential version, calling ERROR_HANDLING from the body of OPEN_GKS procedure will cause GKS_ERROR to be propagated back to the application program that called OPEN_GKS. In the tasking version, the same effect is achieved, in the following fashion. During execution of the MONITOR task's accept statement for the OPEN_GKS entry, ERROR_HANDLING will be called and GKS_ERROR will be raised. By Ada semantics, since this exception is

not handled by a local handler in the accept, it is propagated both (1) to the point following the accept, and (2) to the point of the entry call. For (1), it is handled in the block enclosing the select statement; thus the MONITOR task can continue the next iteration of the loop without disruption. For (2), note that this point is in the body of the OPEN_GKS procedure. Since there is no exception handler here, GKS_ERROR is propagated (as desired) back to the call in the application program.

Termination of the MONITOR task will be accomplished by selection of the terminate alternative when the application program's tasks have completed.

There are a number of variations on the technique discussed and illustrated above that a GKS implementor may consider in the interest of increasing the potential for parallelism in GKS application programs that use tasking. With the method just outlined, the entire set of state variables is protected by one task. If the state information can be partitioned into independent sets, with one monitor task per set, then an application program task that reads / writes a variable in one set can do so concurrently with a task that reading/writing a variable in another set. The GKS implementor should consider the trade-offs.

Another variation on the protection of state variables is to distinguish between GKS functions that simply read data values and those that write them. The technique outlined above treats readers and writers in the same way; thus it prohibits two tasks in an application program from simultaneously reading state information. It is possible for the GKS implementor to program the monitor task in such a way that concurrent reading by application program tasks is permitted, but that neither concurrent writing nor concurrent reading and writing can occur. There are many different approaches to this problem, depending on whether the implementor takes into account such factors as:

- o the possibility of an application task being aborted, and
- o the possibility of an application task being "starved" for service because of the way that the entry accepts are programmed.

An implementation of GKS that supports multitasking Ada programs can still be used for programs that are purely sequential, though efficiency may be impaired. An implementor may wish to provide two bodies for the GKS package -- one for use with sequential applications, and the other for tasking. The practicality of this scheme depends on the sophistication of the Ada library manager and perhaps also the binder.

Appendix E
Unsupported Generalized Drawing Primitives and Escapes

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

This Appendix provides clarification of the relationship between a GKSM metafile and the GKS/Ada subprograms for Generalized Drawing Primitives (GDP) and ESCAPE (ESC) functions. Each GDP and ESC function registered is available to the application program as an individual procedure with its own formal parameters and subprogram name as described in 5.1 in this binding.

The GKS/Ada implementation should provide the ability to write or read a registered GDP or ESCAPE to a metafile even though that GKS/Ada implementation does not support the GDP or ESCAPE function. The data record format for registered GDPs and ESCAPEs must therefore be available between implementations in order for this ability to be supported.

For example, consider that a metafile "A" is generated on a GKS/Ada implementation which supports the GDP for circle. Metafile "A" now has a GKSM_DATA_RECORD containing the GDP circle identifier, center point and radius. Then metafile "A" is sent to another site having a GKS/Ada implementation which does not support circle GDP's. At the new site a new metafile "B" will be generated containing all the contents of metafile "A" as well as additional graphics data. It is critical that the circle GDP from metafile "A" be included into metafile "B" even though none of the workstations at the site will be able to generate the circle for display.

To illustrate the technique, the following example shows an application code fragment and a GKS/Ada implementation of the INTERPRET_ITEM metafile function.

```
with GKS;
use  GKS;
with GKS_TYPES;
use  GKS_TYPES;

-- This application program transfers data from metafile "A" to metafile "B"

procedure TRANSFER_METAFILE is

    --  Declare variables here.

    INPUT_METAFILE                  : constant WS_ID       := 1;
    OUTPUT_METAFILE                 : constant WS_ID       := 2;

    INPUT_METAFILE_TYPE             : constant WS_TYPE     := 2;
    OUTPUT_METAFILE_TYPE            : constant WS_TYPE     := 3;

    INPUT_METAFILE_CONNECTION_ID    : constant STRING      := "METAFILE_A";
    OUTPUT_METAFILE_CONNECTION_ID   : constant STRING      := "METAFILE_B";

    METAFILE_DATA_RECORD            :GKSM_DATA_RECORD;
    METAFILE_ITEM_TYPE              : GKSM_ITEM_TYPE;
    LENGTH, MAX_LENGTH              : NATURAL              := 500;

    ERROR_FILE                      : constant STRING      := "MY_ERROR_FILE";
```

```
begin

    -- Open GKS.

    OPEN_GKS (ERROR_FILE);

    -- Open both input and output metafiles.

    OPEN_WS     (INPUT_METAFILE,
                 INPUT_METAFILE_CONNECTION_ID,
                 INPUT_METAFILE_TYPE);

    OPEN_WS     (OUTPUT_METAFILE,
                 OUTPUT_METAFILE_CONNECTION_ID,
                 OUTPUT_METAFILE_TYPE);

    -- Only output metafiles are activated.

    ACTIVATE_WS (OUTPUT_METAFILE);

    -- In this loop every element of Metafile "A" is read, and passed to GKS through the
    -- INTERPRET_ITEM function call. Remember that metafile "A" contains a circle GDP
    -- which will be handled by the INTERPRET_ITEM example which will follow shortly.

    loop

    GET_ITEM_TYPE_FROM_GKSM     (INPUT_METAFILE, METAFILE_ITEM_TYPE,
                                 LENGTH);

    if METAFILE_ITEM_TYPE = 0 then
      -- exit the loop, the metafile is empty.
      exit;
    end if;

    READ_ITEM_FROM_GKSM     (INPUT_METAFILE, MAX_LENGTH,
                             METAFILE_DATA_RECORD);
    INTERPRET_ITEM (METAFILE_DATA_RECORD);
    end loop;

    -- Only output metafiles are deactivated.

    DEACTIVATE_WS (OUTPUT_METAFILE);

    -- Close both the input and output metafiles.

    CLOSE_WS (INPUT_METAFILE);
    CLOSE_WS (OUTPUT_METAFILE);

     - Close GKS.

    CLOSE_GKS;

end TRANSFER_METAFILE;
```

The example application program relies on the GKS/Ada implementation of the INTERPRET_ITEM function to recognize the circle GDP and pass the GDP to the output metafile. This could easily be done in the following code segment.

Let us assume that the private type GKSM_DATA_RECORD is declared as a discriminant record type with different components based on the type of item. When the GKSM_DATA_RECORD contains a GDP, several fields exist containing all the available information about the GDP.

```
  type GKSM_DATA_RECORD (TYPE_OF_ITEM      : GKSM_ITEM_TYPE := 0;
                         LENGTH            : NATURAL := 0) is
  record
    case TYPE_OF_ITEM is
      when OPEN_GKS  => ...
      when POLYLINE  => ...
      when GDP       =>
        ID                                 : GDP_ID;
        NUM_PTS                            : POSITIVE;
        INTEGER_DATA_LENGTH                : NATURAL;
        REAL_DATA_LENGTH                   : NATURAL;
        LIST_OF_POINTS                     : WC.POINT_ARRAY (1..NUM_PTS);
        INTEGER_DATA                       : INTEGER_ARRAY
                                             (1..INTEGER_DATA_LENGTH);
        REAL_DATA                          : REAL_ARRAY (1..REAL_DATA_LENGTH);
      when ...
    end case;
  end record;

-- Example of how an implementation could handle the transfer of an unsupported GDP
-- through the INTERPRET_ITEM function.

procedure INTERPRET_ITEM      ( ITEM : in GKSM_DATA_RECORD) is
  REGISTERED_GDP_CIRCLE  : constant GDP_ID := 1;
begin
  case ITEM.TYPE_OF_ITEM is
    when OPEN_GKS      => ...
    when POLYLINE      => ...
    when GDP           =>
      case ITEM.ID is
        when REGISTERED_GDP_SPLINE    => ...
        when REGISTERED_GDP_ELLIPSE   => ...
        when REGISTERED_GDP_CIRCLE    =>
          -- call the metafile generator here with the ITEM data record as the parameter to be
          -- written to  all open and active metafiles
        when ...
      end case;
  end case;
end INTERPRET_ITEM;
```

Appendix F
Metafile Item Types

(This Appendix does not form an integral part of the standard, but provides additional information.)

The GET ITEM TYPE FROM GKSM function returns the type of the next metafile item; however, the value of this type may vary depending on the metafile implementation. In order to allow application programs to be written in a manner which is independent of the metafile implementation, the following Ada names are suggested. The implementation should define these names with values which match the values returned by the GET ITEM TYPE FROM GKSM procedure.

GKSM Item Type	Ada Name
ASPECT SOURCE FLAGS	GKSM_ASF
CELL ARRAY	GKSM_CELL_ARRAY
CHARACTER EXPANSION FACTOR	GKSM_CHAR_EXPANSION_FACTOR
CHARACTER SPACING	GKSM_CHAR_SPACING
CHARACTER VECTORS	GKSM_CHAR_VECTORS
CLEAR WORKSTATION	GKSM_CLEAR_WS
CLIPPING RECTANGLE	GKSM_CLIPPING_RECTANGLE
CLOSE SEGMENT	GKSM_CLOSE_SEGMENT
COLOUR REPRESENTATION	GKSM_COLOUR_REPRESENTATION
CREATE SEGMENT	GKSM_CREATE_SEGMENT
DEFERRAL STATE	GKSM_DEFERRAL_STATE
DELETE SEGMENT	GKSM_DELETE_SEGMENT
END ITEM	GKSM_END_ITEM
ESCAPE	GKSM_ESCAPE
FILL AREA	GKSM_FILL_AREA
FILL AREA COLOUR INDEX	GKSM_FILL_AREA_COLOUR_INDEX
FILL AREA INDEX	GKSM_FILL_AREA_INDEX
FILL AREA INTERIOR STYLE	GKSM_FILL_AREA_INTERIOR_STYLE
FILL AREA REPRESENTATION	GKSM_FILL_AREA_REPRESENTATION
FILL AREA STYLE INDEX	GKSM_FILL_AREA_STYLE_INDEX
GENERALIZED DRAWING PRIMITIVE	GKSM_GDP
LINETYPE	GKSM_LINETYPE
LINEWIDTH SCALE FACTOR	GKSM_LINEWIDTH_SCALE_FACTOR
MARKER SIZE SCALE FACTOR	GKSM_MARKER_SIZE_SCALE_FACTOR
MARKER TYPE	GKSM_MARKER_TYPE
MESSAGE	GKSM_MESSAGE
PATTERN REFERENCE POINT	GKSM_PATTERN_REFERENCE_POINT
PATTERN REPRESENTATION	GKSM_PATTERN_REPRESENTATION
PATTERN VECTORS	GKSM_PATTERN_VECTORS
PICK IDENTIFIER	GKSM_PICK_ID
POLYLINE	GKSM_POLYLINE
POLYLINE COLOUR INDEX	GKSM_POLYLINE_COLOUR_INDEX
POLYLINE INDEX	GKSM_POLYLINE_INDEX

(Continued on Next Page)

GKSM Item Type	Ada Name
POLYLINE REPRESENTATION	GKSM_POLYLINE_REPRESENTATION
POLYMARKER	GKSM_POLYMARKER
POLYMARKER REPRESENTATION	GKSM_POLYMARKER_REPRESENTATION
POLYMARKER COLOUR INDEX	GKSM_POLYMARKER_COLOUR_INDEX
REDRAW ALL SEGMENTS ON WORKSTATION	GKSM_REDRAW_ALL_SEGMENTS_WS
RENAME SEGMENT	GKSM_RENAME_SEGMENT
SET DETECTABILITY	GKSM_SET_DETECTABILITY
SET HIGHLIGHTING	GKSM_SET_HIGHLIGHTING
SET SEGMENT PRIORITY	GKSM_SET_SEGMENT_PRIORITY
SET SEGMENT TRANSFORMATION	GKSM_SET_SEGMENT_TRANSFORMATION
SET VISBILITY	GKSM_SET_VISIBILITY
TEXT	GKSM_TEXT
TEXT ALIGNMENT	GKSM_TEXT_ALIGNMENT
TEXT COLOUR INDEX	GKSM_TEXT_COLOUR_INDEX
TEXT FONT AND PRECISION	GKSM_TEXT_FONT_AND_PRECISION
TEXT INDEX	GKSM_TEXT_INDEX
TEXT PATH	GKSM_TEXT_PATH
TEXT REPRESENTATION	GKSM_TEXT_REPRESENTATION
UPDATE WORKSTATION	GKSM_UPDATE_WS
USER ITEM	GKSM_USER_ITEM
WORKSTATION VIEWPORT	GKSM_WS_VIEWPORT
WORKSTATION WINDOW	GKSM_WS_WINDOW

Appendix G
Index of GKS Functions

(This Appendix does not form an integral part of this standard.)