INTERNATIONAL STANDARD

ISO 8651-3

First edition
1988-09-15

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

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

Part 3 : Ada

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

Partie 3 : Ada

Reference number
ISO 8651-3 : 1988 (E)

Foreword

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

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

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

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

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

— *Part 1 : FORTRAN*

— *Part 2 : PASCAL*

— *Part 3 : Ada*

Annexes A to G are for information only.

Contents

This page intentionally left blank

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

Part 3 :
Ada

0 Introduction

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

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

1 Scope and field of application

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

2 References

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

ISO 8652, *Programming Languages — Ada.*

3 The Ada language binding of GKS

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

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

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

3.1 Conformance

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

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

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

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

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

3.2 Implications of the language

3.2.1 Functional mapping

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

3.2.2 Implementation and host dependencies

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

3.2.3 Error handling

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

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

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

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

The procedure ERROR_HANDLING is defined as a library subprogram:

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


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

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

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

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

3.2.4 Data mapping

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

GKS integers are mapped to Ada integer types.

GKS reals are mapped to Ada floating-point types.

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

GKS points are mapped to Ada record types.

GKS names are mapped to Ada discrete types.

GKS enumeration types are mapped to Ada enumeration types.

GKS vectors are mapped to Ada record types.

GKS matrices are mapped to Ada array types.

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

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

GKS ordered pairs are mapped to Ada record types.

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

3.2.5 Multi-tasking

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

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

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

3.2.6 Packaging

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

package GKS is ... end GKS;

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

package GKS_TYPES is ... end GKS_TYPES;

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

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

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

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

- o generic package GKS_COORDINATE_SYSTEM
- o generic package GKS_LIST_UTILITIES

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

3.2.7 Application program environment

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

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

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

3.2.8 Registration [1)]

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

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

4 Tables

4.1 Procedures

Table 1 - Abbreviations used in Procedure names

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

Table 2 - List of procedures using the abbreviations

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

Table 2 - List of procedures using the abbreviations

GKSM
- GET_ITEM_TYPE_FROM_GKSM
- READ_ITEM_FROM_GKSM
- WRITE_ITEM_TO_GKSM

ID
- INQ_CURRENT_PICK_ID_VALUE
- SET_PICK_ID

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

Table 2 - List of procedures using the abbreviations

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

Table 2 - List of procedures using the abbreviations

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