INTERNATIONAL STANDARD

ISO/IEC 9593-4

First edition
1991-12-15

Information technology — Computer graphics — Programmer's Hierarchical Interactive Graphics System (PHIGS) language bindings —

Part 4: C

Technologies de l'information — Infographie — Interfaces langage entre un programme d'application et son support graphique — Partie 4: C

Reference number
ISO/IEC 9593-4:1991(E)

Contents

Foreword

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

International Standard ISO/IEC 9593-4 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology.*

ISO/IEC 9593 consists of the following parts, under the general title *Information technology — Computer graphics — Programmer's Hierarchical Interactive Graphics System (PHIGS) language bindings* :

— *Part 1 : FORTRAN 77*
— *Part 3 : ADA*
— *Part 4 : C*

Annexes A, B, C, D, and E of this part of ISO/IEC 9593 are for information only.

Introduction

The Programmer's Hierarchical Interactive Graphics System (PHIGS), the functional description of which is given in ISO/IEC 9592-1, 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/IEC 9593 is to define a standard binding for the C computer programming language.

Information technology – Computer graphics – Programmer's Hierarchical Interactive Graphics System (PHIGS) language bindings –

Part 4: C

1 Scope

The Programmer's Hierarchical Interactive Graphics System (PHIGS), ISO/IEC 9592-1, specifies a language independent nucleus of a graphics system. For integration into a programming language, PHIGS is embedded in a language dependent layer obeying the particular conventions of that language. This part of ISO/IEC 9593 specifies such a language dependent layer for the C language.

2 Normative references

The following standards contain provisions which, through reference in this text, constitute provisions of this part of ISO/IEC 9593. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this part of ISO/IEC 9593 are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. Members of IEC and ISO maintain registers of currently valid International Standards.

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

ISO/IEC 9899 : 1990, *Programming Languages – C.*

ISO/IEC TR 9973 : 1988, *Information processing – Procedures for Registration of Graphical Items.*

3 The C language binding of PHIGS

3.1 Conformance

This binding incorporates the rules of conformance defined in the PHIGS Standard (ISO/IEC 9592) for PHIGS implementations, with those additional requirements specifically defined for C language implementations of PHIGS.

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

> In order to conform, an implementation of the C binding of PHIGS shall implement ISO/IEC 9592-1. It shall make visible all of the declarations in the C binding specified in this part of ISO/IEC 9593.
>
> Thus, for example, the syntax of the function names shall be precisely as specified in this part of ISO/IEC 9593 and the parameters shall be of the data types stated in this part of ISO/IEC 9593.

3.2 Functions versus macros

An implementation may substitute macros for functions. However, the macros must be designed so that side-effects work properly.

3.3 Character strings

The C language represents character strings as an array of characters terminated by the null character (*i.e.* '\0'). This means that the null character is not usable as a printable character.

3.4 Function identifiers

The C standard (ISO/IEC 9899) requires that compilers recognize internal identifiers which are distinct in at least 31 characters. That standard also requires that external identifiers (*i.e.* those seen by the linker) be recognized to a minimum of six characters, independent of case.

In order to produce more readable C code, this part of ISO/IEC 9593 binds the PHIGS function names to identifiers which can be longer than six characters but are less than 31 characters. Implementations which must run in environments which honour less than 31 characters in external identifiers must include a set of `#defines` in the header file which equate the long identifiers defined in this part of ISO/IEC 9593 to a set of shorter identifiers which conform to the requirements of the implementation. An example of these shorter names can be found in annex C.

3.5 Registration

ISO/IEC 9592-1 reserves certain value ranges for registration[1] as graphical items. The registered graphical items will be bound to the C programming language (and other programming languages). The binding of registered items will be consistent with the bindings presented in this part of ISO/IEC 9593.

3.6 Identifiers for graphics items

ISO/IEC 9592-1 provides a mechanism to extend its functionality by the use of the graphical items GENERALIZED DRAWING PRIMITIVE, GENERALIZED DRAWING PRIMITIVE 3, GENERALIZED STRUCTURE ELEMENT and ESCAPE. These items are referenced via identifiers formed from the registration number of the item. This binding specifies the format of the identifiers but it does not specify the registration of the identifiers. The identifiers are used as arguments to the functions `pgdp`, `pgdp3`, `pgse`, and `pescape` and returned to the application by the functions `pinq_elem_content` and `pinq_cur_elem_content`.

[1] For the purpose of this International Standard and according to the rules for the designation and operation of registration authorities in the ISO/IEC JTC1 procedures, the ISO and IEC Councils have designated the National Institute of Standards and Technology (National Computer Systems Laboratory), A-266 Technology Building, Gaithersburg, MD 20899, USA, to act as registration authority.

An implementation may also represent these items as separate functions, but this is not required.

The format of the identifiers for graphical items is

"PFFF_Tn"

where:

FFF	is the abbreviation of the PHIGS function name.
T	is the type of the item, "R" for registered and "U" for unregistered.
n	is the registration number (for unregistered items, the absolute value of the item number shall be used).

The formats for registered GENERALIZED DRAWING PRIMITIVE, GENERALIZED DRAWING PRIMITIVE 3, GENERALIZED STRUCTURE ELEMENT and ESCAPE graphical items are:

```
#define PGDP_Rn               (n)
#define PGDP3_Rn              (n)
#define PGSE_Rn               (n)
#define PESCAPE_Rn            (n)
```

For example, the identifier for registered GENERALIZED DRAWING PRIMITIVE item 12 would be:

```
#define PGDP_R12              (12)
```

The formats for unregistered GENERALIZED DRAWING PRIMITIVE, GENERALIZED DRAWING PRIMITIVE 3, GENERALIZED STRUCTURE ELEMENT and ESCAPE graphical items are:

```
#define PGDP_Un               (-n)
#define PGDP3_Un              (-n)
#define PGSE_Un               (-n)
#define PESCAPE_Un            (-n)
```

For example, the identifier for unregistered GENERALIZED DRAWING PRIMITIVE item -1 would be:

```
#define PGDP_U1               (-1)
```

3.7 Return values

All PHIGS C functions return `void`.

3.8 Header files

C provides a mechanism to allow external files to be included in a compilation. The file `phigs.h` shall be included in any application program that intends to use PHIGS via the C binding.

Clause 5 of this binding describes the data types that shall be defined in the file `phigs.h`; clause 6 defines the macros that shall be in the file `phigs.h`. Additional implementation-dependent items may be placed in this file if needed.

The type `size_t` is used in the binding. It is defined in the standard header `<stddef.h>`. Therefore, the file `phigs.h` shall include `<stddef.h>` in order to define `size_t`.

The file `phigs.h` shall also contain external prototype declarations for all PHIGS C functions because they return `void`. For example, the declaration for the function `popen_phigs` is:

```
extern   void   popen_phigs(const char *err_file, size_t mem_units);
```

3.9 Memory management

The application shall allocate the memory needed for the data returned by the implementation. In general, the application will allocate a C structure and pass a pointer to that structure to an inquiry routine which will then place information into the structure. However, a number of inquiry functions return variable length data, the length of which is not known *a priori* by the application.

These functions fall into two classes. One class of functions returns a simple, homogeneous list of elements. For example, the function INQUIRE STRUCTURE IDENTIFIERS returns a list of the structure identifiers in use. The other class returns complex, heterogeneous data structures. For example, the function INQUIRE LOCATOR DEVICE STATE returns the device state which includes a locator data record; the data record can contain arbitrarily complex implementation-defined data structures. The binding of these two classes of functions is described in detail below.

Additional binding-specific errors which relate to memory management are described in 3.11.3.

3.9.1 Inquiry functions which return simple lists

Inquiry functions which return a list of elements are bound such that the application can inquire about a portion of the list. This list portion is a subset of the implementation's internal list and is called the application's list. This allows the application to process the implementation's list in a piecewise manner rather than all at once.

The application allocates the memory for its list and passes that list to the implementation. The implementation places the results of the inquiry into the list. In order to support this policy of memory management, three additional parameters have been added to functions which return simple lists:

a) `num_elem_appl_list`: An integer input parameter which is the size of the application's list. The value of `num_elem_appl_list` indicates the number of list elements which will fit into the application's list. A value of 0 is valid and allows the application to determine the size of the implementation's list (which is returned via `num_elem_impl_list`) without having the implementation return any of the elements of its list.

b) `start_ind`: An integer input parameter which is the starting index into the implementation's list. (Index 0 is the first element of both the implementation's and application's list.) `start_ind` indicates the element in the implementation's list that is copied into index 0 of the application's list. Elements are copied sequentially from the implementation's list into the application's list until the application's list is full or there are no more elements in the implementation's list.

   If `start_ind` is out of range, error number 2200 (`PE_START_IND_INVAL`) is returned as the value of the error indicator parameter.

c) `num_elem_impl_list`: An output parameter which is a pointer to an integer. The implementation stores into this parameter the number of elements that are in the implementation's list.

Each function which returns a simple list has an output parameter *list*, which is a pointer to a structure with fields for the number of elements in the list and a pointer to the elements in the list. The type of *list* depends upon the function in which it is used. The implementation places the elements in the memory pointed to by *list->elems* and assigns the number of elements in *list->elems* to *list->num_elems*. If the implementation's list is of zero length, no data is placed in *list->elems* and *list->num_elems* is set to zero.

3.9.2 Inquiry functions which return complex data structures

The data returned by the element content functions and the functions which return input device data records can be complex in structure; they cannot be represented by a simple list of elements. It would be an onerous task for the application to allocate and to prepare data structures for these routines. In order to facilitate the task of using these inquiry functions, the binding defines a new resource, called a *Store*, to manage the memory for these functions.

A *Store* is used by the implementation to manage the memory needed by the functions which return complex data structures. The *Store* resource is opaque to the application. The application does not know the structure of the *Store* or how it is implemented. The *Store* is defined as a `void *`. The binding defines two new functions which create

(CREATE STORE, bound as `pcreate_store`) and destroy (DELETE STORE, bound as `pdel_store`) a *Store*.

The semantics of the *Store* resource provide two levels of memory management: The implementation is responsible for managing the memory at a low level because it uses, re-uses, allocates and deallocates memory from the system in order to return information to the application. But the application is ultimately responsible for managing the memory at a high level because it creates and destroys *Store*s.

A *Store* is passed as a parameter to a function returning complex data. Another parameter to this function is a pointer to a pointer to a structure which defines the format of the returned data. The *Store* contains memory for the structure and any additional memory referenced by fields within the structure. The application accesses the returned data through its pointer to the structure; it does not use the *Store* to access the data.

For some functions, a *Store* is used to manage the memory for two or more distinct complex data structures. For example, in the function INQUIRE PICK DEVICE STATE, the *Store* manages the memory for the pick filter, the initial pick path, and the pick data record, all of which are returned to the application.

A *Store* continues to hold the information returned from the function until the *Store* is destroyed by the `pdel_store` function, or until the *Store* is used as an argument to a subsequent function which returns complex data. At that time, the old information is replaced with the new. Thus multiple calls to functions overwrite the contents of a *Store*. A *Store* only contains the results of the last function. An application may create more than one *Store*.

This binding defines two new errors that can occur when using or creating a *Store*; these errors are described in 3.11.3. For most functions using a *Store*, these and other errors are returned via the "error indicator" parameter. However, the functions RETRIEVE PATHS TO ANCESTORS, RETRIEVE PATHS TO DESCENDANTS and ESCAPE do not have an error indicator parameter. For these functions, the error reporting mechanism is used when an error is encountered. For these functions, the implementation shall, in addition to reporting the error, set the pointer to the returned data to NULL when an error occurs.

The definitions for the functions CREATE STORE and DELETE STORE follow:

CREATE STORE (PHOP, *, *, *)

Parameters:

OUT	error indicator	I
OUT	store	STORE

Effect: Creates a Store and returns a handle to it in the parameter *store*. If the Store cannot be created, the *store* parameter is set to NULL and the error indicator is set to one of the following error numbers:

002 Ignoring function, function requires state (PHOP, *, *, *).

2203 Ignoring function, error allocating Store.

Errors: none

DELETE STORE (PHOP, *, *, *)

Parameters:

OUT	error indicator	I
IN/OUT	store	STORE

Effect: Deletes the Store and all internal resources associated with it. The parameter *store* is set to NULL to signify that it is no longer valid. If an error is detected, the error indicator is set to the following error number:

002 Ignoring function, function requires state (PHOP, *, *, *).

Errors: none

3.9.3 Meaning of the size of an element

The functions INQUIRE CURRENT ELEMENT TYPE AND SIZE and INQUIRE ELEMENT TYPE AND SIZE return the size of an element. The size of an element is the size of a buffer, in bytes, that the application would have to allocate in order to contain the element. If the application would not have to allocate any memory, then 0 is returned.

3.10 Inquiries returning structure elements

PHIGS provides the ability for the application to inquire the contents of a structure element (through INQUIRE CURRENT ELEMENT CONTENT and INQUIRE ELEMENT CONTENT). PHIGS also allows the application to read in an archive file that can contain GDPs (both GENERALIZED DRAWING PRIMITIVEs and GENERALIZED DRAWING PRIMITIVE 3's) and GSEs (GENERALIZED STRUCTURE ELEMENTs) which are not supported by the implementation. Thus it is possible for the application to inquire the contents of a structure element that contains a GDP or GSE data record that the implementation does not support.

In order to allow the inquiry of unsupported data records, the binding has introduced a field, called `unsupp`, to the GDP and GSE data record structures. The type of this field is `Pdata` which is a structure containing a field for the size of a block of data and another field for a pointer to the data. The `unsupp` field is used if the implementation does not support a GDP or a GSE.

3.11 Error handling

3.11.1 Application defined error handlers

An application can define the error handling function for the PHIGS implementation via the utility function SET ERROR HANDLER, which is bound as `pset_err_hand`. The definition for SET ERROR HANDLER is:

SET ERROR HANDLER (*, *, *, *)

Parameters:

IN	new error handling function	Function
OUT	old error handling function	Function

Effect: Sets the PHIGS error handling function to *new error handling function* and returns the previous function in *old error handling function*.

Errors: none

Application defined error handling functions accept the same arguments as the standard error handler. It may invoke the standard error logging function `perr_log`.

ISO/IEC 9592-1 defines the initial error handling function to be `perr_hand`; that is, the value of the parameter `old_err_hand` points to `perr_hand` when SET ERROR HANDLER is invoked for the first time.

When the application changes the error handling function, the implementation will invoke the new function when an error is detected. If the application calls the default error handling function `perr_hand`, `perr_hand` will always call the default error logging function `perr_log`; `perr_hand` does not call the error handling function specified in SET ERROR HANDLER.

3.11.2 Error codes

This binding defines, in 6.2, a set of constants for the PHIGS error numbers. Each error constant begins with the characters PE.

3.11.3 C specific PHIGS errors

This binding defines the following additional errors, beyond the ones described in ISO/IEC 9592-1.

Error	Message
2200	`Ignoring function, start index is out of range` Is issued when the start index is less than zero or larger than the last element in the implementation list.
2201	`Ignoring function, length of application's list is negative` Is issued when the length of the application's list is less than zero.
2202	`Ignoring function, enumeration type out of range` Is issued when a parameter value whose type is an enumeration is out range of the enumeration.
2203	`Ignoring function, error while allocating a Store` Is issued when an error is detected during CREATE STORE.
2204	`Ignoring function, error while allocating memory for a Store` Is issued when a function using a Store is unable to allocate memory for the store.

3.12 Storage of two-dimensional data

3.12.1 Storage of matrices

ISO/IEC 9592-1 represents transformations as 3×3 and 4×4 matrices. This part of ISO/IEC 9593 binds a PHIGS 3×3 matrix to the type `Pmatrix` and a PHIGS 4×4 matrix to the type `Pmatrix3`.

The elements of a PHIGS 3×3 matrix are:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

and these elements are stored such that:

```
m[0][0] = a;   m[0][1] = b;   m[0][2] = c;
m[1][0] = d;   m[1][1] = e;   m[1][2] = f;
m[2][0] = g;   m[2][1] = h;   m[2][2] = i;
```

where `m` is of type `Pmatrix`.

The elements of a PHIGS 4×4 matrix are:

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix}$$

and these elements are stored such that:

```
m[0][0] = a;   m[0][1] = b;   m[0][2] = c;   m[0][3] = d;
m[1][0] = e;   m[1][1] = f;   m[1][2] = g;   m[1][3] = h;
m[2][0] = i;   m[2][1] = j;   m[2][2] = k;   m[2][3] = l;
m[3][0] = m;   m[3][1] = n;   m[3][2] = o;   m[3][3] = p;
```

where m is of type Pmatrix3.

For example, in order to store the projection transformation defined by:

$$x \rightarrow x'/w'$$
$$y \rightarrow y'/w'$$
$$z \rightarrow z'/w'$$

in PHIGS 4×4 matrix, the values shall be stored such that:

```
    x'  = p[0][0]*x + p[0][1]*y + p[0][2]*z + p[0][3];
    y'  = p[1][0]*x + p[1][1]*y + p[1][2]*z + p[1][3];
    z'  = p[2][0]*x + p[2][1]*y + p[2][2]*z + p[2][3];
    w'  = p[3][0]*x + p[3][1]*y + p[3][2]*z + p[3][3];
```

where p is of type Pmatrix3.

3.12.2 Storage of colour arrays

The entries for the Ppat_rep data type shall be stored such that the colour index at the (i,j)-th entry of a DX×DY array of colour indices is given by:

```
colr_ind(i,j) = colr_rect.colr_array[i + DX*j];
```

where colr_rect is of type Ppat_rep and

```
colr_rect.dims.size_x = DX;
colr_rect.dims.size_y = DY;
                          i = 0,...,DX-1;
                          j = 0,...,DY-1;
```