**NAME**

intro_phigs – introduction to the PEX-SI graphics library

**DESCRIPTION**

The PEX-SI graphics library implements the *Programmer's Hierarchical Interactive Graphics System* (PHIGS) and portions of the proposed PHIGS extension (PHIGS PLUS). This PHIGS implementation is designed specifically and exclusively for use with X servers that implement the PEX extension.

These manual pages describe the syntax of each function in the PEX-SI graphics library. There is one entry for each function.

**Reserved Names**

PEX-SI applications should avoid using variable and function names that conflict with names that PEX-SI uses internally. Since PEX-SI programs are linked to the Xlib library, do not use symbols that start with the prefixes reserved by that package, symbols in the Math library, or system calls. The public symbol names provided by PEX-SI all start with *p*. To avoid conflicts with the PEX-SI name space, do not define any of the following symbols:

- PHIGS or PHIGS PLUS C function names
- Names with the prefixes **phigs**, **phg**, **P**, or **p**.
- System library symbols, such as **time**.

**Abbreviations**

The following list describes the abbreviations used in this manual.

| Abbreviation | Name |
|---|---|
| ANSI | American National Standards Institute |
| ASF | aspect source flag |
| CSS | Central Structure Store |
| DC | device coordinates |
| GDP | generalized drawing primitive |
| GSE | generalized structure element |
| HLHSR | hidden line/hidden surface removal |
| ISO | International Organisation for Standards |
| MC | modelling coordinates |
| NPC | normalized projection coordinates |
| NURB | non-uniform rational B-spline |
| PDT | PHIGS description table |
| PET | prompt/echo type |
| RGB | red/green/blue colour model |
| TLC | text local coordinates |
| WC | world coordinates |
| WS | workstation |

PHIGS has four operating state variables, each with two possible values:

| Variable Name | Open State Enumeraion Constant | Closed State Enumeration Constant |
|---|---|---|
| System State | PHOP | PHCL |
| Workstation State | WSOP | WSCL |
| Structure State | STOP | STCL |
| Archive State | AROP | ARCL |

Each PHIGS function requires that PHIGS be in a certain operating state before the function can be called. The *SYNOPSIS* section of each function specifies the required value that each of the four operating state variables must have to call that function. An asterisk (∗) is used when an operating state variable may be in either state (that is, does not matter). All of the operating state variables are initially set to the closed value.

**Extensions to the PHIGS Standard**

The PEX-SI library contains the following optional functions that are not in the PHIGS specification, but are extremely useful.

WORKSTATION TYPE CREATE
WORKSTATION TYPE DESTROY
WORKSTATION TYPE GET
WORKSTATION TYPE SET

**Include Files**

PEX-SI applications should use the constants and enumerated values defined in the PHIGS files to decrease the possibility of passing bad data and avoid conflict with future changes. C applications should include the file **phigs.h**. See the example programs in Xsrc/extensions/clients/PEX/demo.

**C Binding Notes**

The C binding used for the PEX-SI graphics library matches the PHIGS C Binding (Draft ISO DIS 9593-4:199x(E), 1 August 1990), for the PHIGS standard functions, with several deviations. These deviations either correct bugs in the DIS binding or anticipate changes that will be made when the DIS is approved. The following lists the PEX SI deviations from the DIS binding.

1. In the definition of the data type Pcolr_rep, the name of the Phsv field is *hsv*, not *hlv*.

2. All Pdouble usage in the DIS binding is Pfloat in the PEX-SI binding.

3. Everywhere the DIS binding lists the parameter ‘‘Pstore *store’’, PEX-SI uses ‘‘Pstore store’’. In other words, the store object is passed directly, and not as a pointer the store object.

4. The Pstring_data and Pstring_data3 structures in PEX-SI are different from those specified in the DIS binding. Pstring_data and Pstring_data3 are defined as follows in PEX-SI:

```
typedef struct {
        Pint    buffer_size;    /* input buffer size */
        Pint    init_pos;       /* initial editing position */
        union {
                Pint    unused;
        } pet_r1;
} Pstring_data;

typedef  Pstring_data    Pstring_data3;
```

5. The following functions do not have the *in_buf_size* and *init_pos* parameters listed in the DIS binding. Those fields are instead in the Pstring_data (or Pstring_data3) structure passed to these functions:
INITIALIZE STRING
INITIALIZE STRING 3
INQUIRE STRING DEVICE STATE
INQUIRE STRING DEVICE STATE 3
INQUIRE DEFAULT STRING DEVICE STATE
INQUIRE DEFAULT STRING DEVICE STATE 3

6. The Pstroke_data and Pstroke_data3 structures in PEX-SI are different from those specified in the DIS binding. Pstroke_data and Pstroke_data3 are defined as follows in PEX-SI:

```
typedef struct {
        Pint    buffer_size;    /* input buffer size */
        Pint    init_pos;       /* initial editing position */
        Pfloat  x_interval;     /* x interval */
        Pfloat  y_interval;     /* y interval */
        Pfloat  time_interval;  /* time interval */
        union {
```

```
                struct {
                        Pint        unused;
                } pet_r1;
                struct {
                        Pint        unused;
                } pet_r2;
                struct {
                        Pmarker_attrs    marker_attrs;    /* marker attributes */
                } pet_r3;
                struct {
                        Pline_attrs      line_attrs;      /* line attributes */
                } pet_r4;
        } pets;
} Pstroke_data;

/* Pstroke_data3 -- stroke data record 3 */
typedef struct {
        Pint    buffer_size;        /* input buffer size */
        Pint    init_pos;           /* initial editing position */
        Pfloat  x_interval;         /* x interval */
        Pfloat  y_interval;         /* y interval */
        Pfloat  z_interval;         /* z interval */
        Pfloat  time_interval;      /* time interval */
        union {
                struct {
                        Pint    unused;
                } pet_r1;
                struct {
                        Pint    unused;
                } pet_r2;
                struct {
                        Pmarker_attrs    marker_attrs;    /* marker attributes*/
                } pet_r3;
                struct {
                        Pline_attrs      line_attrs;      /* line attributes*/
                } pet_r4;
        } pets;
} Pstroke_data3;
```

7. The following functions do not have the *in_buf_size* and *init_pos* fields listed in the DIS binding. Those fields are instead in the Pstroke_data or Pstroke_data3 structure passed to these functions:
INITIALIZE STROKE
INITIALIZE STROKE 3
INQUIRE STROKE DEVICE STATE
INQUIRE STROKE DEVICE STATE 3
INQUIRE DEFAULT STROKE DEVICE STATE
INQUIRE DEFAULT STROKE DEVICE STATE 3

8. The Pval_data and Pval_data3 structures in PEX-SI are different from those specified in the DIS binding. Pval_data and Pval_data3 are defined as follows in PEX-SI:

```
typedef struct {
        Pfloat  low;    /* low range limit */
        Pfloat  high;   /* high range limit */
        union {
```

```
                struct {
                        Pint    unused;
                } pet_r1;
                struct {
                        Pchar   *label;
                        Pchar   *format;
                        Pchar   *low_label;
                        Pchar   *high_label;
                } pet_u1;
        } pets;
} Pval_data;

typedef  Pval_data      Pval_data3;
```

9. The following functions do not have the *low_value* and *high_value* parameters listed in the DIS binding. Those fields are instead in the Pval_data (or Pval_data3) structure passed to these functions.
INITIALIZE VALUATOR
INITIALIZE VALUATOR 3
INQUIRE VALUATOR DEVICE STATE
INQUIRE VALUATOR DEVICE STATE 3
INQUIRE DEFAULT VALUATOR DEVICE STATE
INQUIRE DEFAULT VALUATOR DEVICE STATE 3

10. The PEX-SI binding uses ‘‘char *’’ instead of ‘‘void *’’.

11. Pedge_attrs.colr_asf is Pedge_attrs.colr_ind_asf in PEX-SI.

12. Pline_attrs.colr_asf is Pline_attrs.colr_ind_asf in PEX-SI.

13. Pint_attrs.colr_asf is Pint_attrs.colr_ind_asf in PEX-SI.

14. Pmarker_attrs.colr_asf is Pmarker_attrs.colr_ind_asf in PEX-SI.

15. In INQUIRE ANNOTATION FACILITIES the last parameter has been replaced with two Pfloat pointers:

**pinq_anno_facs ( type, length, start, error_ind, styles, total_length, num_anno_char_hts, min_char_ht, max_char_ht )**

| | | |
|---|---|---|
| **Pwstype** | **type;** | *workstation type* |
| **Pint** | **length;** | *length of application list* |
| **Pint** | **start;** | *starting position* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pint_list** | ***styles;** | *OUT list annotation styles* |
| **Pint** | ***total_length;** | *OUT length of list in PHIGS* |
| **Pint** | ***num_anno_char_hts;** | *OUT number of character heights* |
| **Pfloat** | ***min_char_ht;** | *OUT minimum character height* |
| **Pfloat** | ***max_char_ht;** | *OUT maximum character height* |

16. The second parameter to the function pset_pat_ref_point_vec is "Pvec3 pat_ref_vec[2]", not "Pvec3 *pat_ref_vec[2]".

17. The connection identifier argument to the function pinq_ws_conn_type is "Pconnid *conn_id".

18. The function identification numbers in the file *phigscfunc.h* do not all agree with the numbers specified in the DIS C binding. the transformation utilities (numbers 90 through 109 in the DIS binding) do not have identification numbers in PEX-SI because those functions never invoke error handling. See *phigscfun.h* for the function numbers that PEX-SI assigns.

    The macro names used in *phigscfunc.h* do correspond to those in the DIS binding, but the numbers associated with each macro may be different. If your application uses the defined macros, the differences in the identification numbers should not cause any problems.

A definition of the binding is in the file `Xsrc/extensions/lib/PEX/c_binding/phigs.c_bind`. This file contains stubs for all of the binding functions and is used to build the lint library. The data structures are defined in the file `Xsrc/extensions/lib/PEX/include/phigs.h`.

**Structure Storage**

The API supports client side and server side CSS. Client side CSS is used if the connecting server does not support structure storage.

**PHIGS Monitor (PM)**

To meet many of the goals and constraints of PHIGS, PHIGS PLUS, X, and PEX, the PEX-SI graphics library is implemented as two processes. One process resides in the client's process and one resides in a separate process called the PHIGS Monitor (PM). The PM processes window system and input device events associated with a client's PHIGS workstations, implements the window system event queue, and services event requests and inquiries from the client-side API code. The PM also contains the CSS for PHIGS workstations that exist on the client side.

There is one PHIGS monitor per PHIGS client, and it monitors all PHIGS workstations opened by that client regardless of what server each workstation is on. The PM process always runs on the same machine as the client.

**SEE ALSO**

COLOUR(7)
PHIGS DESCRIPTION TABLE(7)
PHIGS TRAVERSAL STATE LIST(7)
PHIGS WORKSTATION DESCRIPTION TABLE(7)

**NAME**

ADD NAMES TO SET – create a structure element containing additions to the current name set

**SYNOPSIS**

**C Syntax**

```
void
padd_names_set ( set )
Pint_list          *set;          set of names to be added
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

The ADD NAMES TO SET function puts a structure element containing additions to the traversal-time *current name set* into the currently open structure according to the current edit mode. The current name set is compared during traversal to the workstation's *name set filters* to determine if primitives that follow in the structure network are invisible, highlighted, and/or selectable by PICK input devices. Each name in the *name set* is a small non-negative integer.

If the current edit mode is INSERT, the ADD NAMES TO SET element is inserted into the open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, the ADD NAMES TO SET element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

**C Input Parameter**

*set* A pointer to a Pint_list structure containing the set of names to be added. The Pint_list data structure is defined in phigs.h as:

```
typedef struct {
        Pint    num_ints;    /* number of Pints in list */
        Pint    *ints;       /* list of integers */
} Pint_list;
```

**Execution**

When traversal of a posted structure network starts, the current name set is empty. During traversal, the member names specified by the ADD NAMES TO SET element are added to the current name set by the union operation. REMOVE NAMES FROM SET elements remove names from the current name set.

The updated current name set applies to primitives that follow in the structure network by set-intersection with the workstation's *name set filters*, which are set by SET INVISIBILITY FILTER (3P), SET HIGHLIGHTING FILTER (3P), and SET PICK FILTER (3P). Each workstation has a single *invisibility filter*, a single *highlighting filter*, and a *pick filter* for each PICK input device. The actual appearance of highlighting is workstation-dependent.

Each filter contains an *inclusion set* and an *exclusion set* of names, both empty by default. A primitive is *eligible* if at least one name in the current name set is in the inclusion set and no name in the current name set is in the exclusion set. If the current name set is empty, subsequent primitives are not eligible. If the workstation's inclusion set is empty (the default), no primitives are eligible. That is, no primitives are invisible, highlighted, or selectable by PICK input devices.

Each name is a small non-negative integer. PHIGS conformance requires support for at least 64 names; PEXSI supports the range 0 to 1023. The same names may be added and removed any number of times during traversal. Since the presence or absence of each name in the current nameset and the workstation's filter affects the eligibility of subsequent primitives, 1024 names provide up to 1024 different simultaneous groupings of primitives.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

REMOVE NAMES FROM SET (3P)
SET INVISIBILITY FILTER (3P)
SET HIGHLIGHTING FILTER (3P)
SET PICK FILTER (3P)
INCREMENTAL SPATIAL SEARCH (3P)
INCREMENTAL SPATIAL SEARCH 3 (3P)

**NAME**

ANNOTATION TEXT RELATIVE – create structure element specifying 2D annotation text primitive

**SYNOPSIS**

**C Syntax**

```
void
panno_text_rel ( ref_pt, anno_offset, text )
Ppoint      *ref_pt;        reference point
Pvec        *anno_offset;   annotation offset
char        *text;          annotation text string
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

The ANNOTATION TEXT RELATIVE subroutine puts a structure element containing the full specification of a two-dimensional ANNOTATION TEXT RELATIVE primitive into the currently open structure.

The ANNOTATION TEXT RELATIVE primitive is a character string. The location of the string in the display is controlled by the *reference point* and *annotation offset* subroutine parameters. ANNOTATION TEXT RELATIVE primitives differ from TEXT primitives in that the reference point is specified in Modelling Coordinates (MC). The *z* coordinate is assumed to be 0. The characters are generated in the Normalized Projection Coordinate (NPC) Space. Aspects of the text display, such as the font, colour, spacing, height, and alignment, are controlled by the current values of the primitive attributes listed below.

If the current edit mode is INSERT, the structure element created by the ANNOTATION TEXT RELATIVE subroutine is inserted into the open structure after the element pointed to by the current element pointer. If the current edit mode is REPLACE, the new ANNOTATION TEXT RELATIVE element replaces the element in the structure pointed to by the element pointer. In either case, the element pointer is updated to point to the new ANNOTATION TEXT RELATIVE element.

**C Input Parameters**

*ref_pt* A pointer to a Ppoint structure containing the *x* and *y* MC that locate the annotation text. The Ppoint structure is defined in phigs.h as follows:

```
typedef struct  {
    Pfloat    x;    /* x coordinate */
    Pfloat    y;    /* y coordinate */
} Ppoint;
```

*anno_offset*

A pointer to a Pvec structure containing the *x* and *y* coordinates of the offset of the text string from the transformed reference point. The annotation offset specifies an offset in NPC. The Pvec structure is defined in phigs.h as follows:

```
typedef struct  {
    Pfloat    delta_x;    /* delta x value */
    Pfloat    delta_y;    /* delta y value */
} Pvec;
```

*text* A pointer to the character string to be written into the display.

**Execution**

When the structure is traversed, the ANNOTATION TEXT RELATIVE element draws the specified character string on the plane in the NPC System defined by the reference point and the annotation offset. These parameters define a Text Local Coordinate (TLC) System in the NPC System. The annotation offset added to the

transformed reference point defines the origin of this TLC System. The *x* and *y* axes of the TLC System are parallel to and have the same direction as the *x* and *y* axes of the NPC System.

The precise position of the text is defined in relation to this plane by the current values of the text primitive attributes ANNOTATION TEXT CHARACTER UP VECTOR, ANNOTATION TEXT CHARACTER, BASE VECTOR, ANNOTATION TEXT PATH, and ANNOTATION TEXT ALIGNMENT. The reference point is subject to the current transformations in the transformation pipeline from the MC System to the workstation display. The text itself is only subject to the transformations in the transformation pipeline from the NPC System to the workstation display.

Other aspects of the appearance of the text are controlled by the attributes TEXT FONT, TEXT PRECISION, ANNOTATION STYLE, ANNOTATION TEXTCHARACTER HEIGHT, ANNOTATION TEXT CHARACTER WIDTH, CHARACTER EXPANSION FACTOR, CHARACTER SPACING, and TEXT COLOUR INDEX.

ANNOTATION TEXT CHARACTER WIDTH and ANNOTATION TEXT BASE VECTOR are implicit attributes derived from ANNOTATION TEXT CHARACTER HEIGHT and ANNOTATION TEXT UP VECTOR, respectively.

**Attributes Applied**

The attributes listed below are used to display the ANNOTATION TEXT RELATIVE primitive when the structure is traversed. The Aspect Source Flags (ASFs) tell where to access the output display attributes. These attributes can come directly from the traversal state list, or they can be accessed indirectly, using the appropriate index in the traversal state list and the corresponding bundled representation in the PHIGS workstation state list.

| | |
|---|---|
| text font | text font ASF |
| text precision | text precision ASF |
| character expansion factor | character expansion factor ASF |
| character spacing | character spacing ASF |
| text colour | text colour index ASF |
| annotation character height | |
| annotation character up vector | |
| annotation text path | |
| annotation text alignment | |
| annotation style | |
| text index | |
| depth cue index | |
| name set | |

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

TEXT (3P)

**NAME**

ANNOTATION TEXT RELATIVE 3 – create structure element specifying 3D annotation text primitive

**SYNOPSIS**

**C Syntax**

```
void
panno_text_rel3 ( ref_pt, anno_offset, text )
Ppoint3      *ref_pt;          reference point
Pvec3        *anno_offset;     annotation offset
char         *text;            annotation text string
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

The ANNOTATION TEXT RELATIVE 3 subroutine puts a structure element containing the full specification of a three-dimensional ANNOTATION TEXT RELATIVE 3 primitive into the currently open structure.

The ANNOTATION TEXT RELATIVE 3 primitive is a character string. The location of the string in the display is controlled by the *reference point* and *annotation offset* subroutine parameters. ANNOTATION TEXT RELATIVE 3 primitives differ from TEXT 3 primitives in that the reference point is specified in Modelling Coordinates (MC), and the plane upon which the characters are generated will always be parallel to the display surface and in the Normalized Projection Coordinate (NPC) Space. Aspects of the text display, such as the font, colour, spacing, height, and alignment, are controlled by the current values of the primitive attributes listed below.

If the current edit mode is INSERT, the structure element created by the ANNOTATION TEXT RELATIVE 3 subroutine is inserted into the open structure after the element pointed to by the current *element pointer*. If the current edit mode is REPLACE, the new ANNOTATION TEXT RELATIVE 3 element replaces the element in the structure pointed to by the element pointer. In either case, the element pointer is updated to point to the new ANNOTATION TEXT RELATIVE 3 element.

**C Input Parameters**

*ref_pt* A pointer to a Ppoint3 structure containing the *x*, *y*, and *z* MC that locate the annotation text. The Ppoint3 structure is defined in phigs.h as follows:

```
typedef struct  {
        Pfloat    x;    /* x coordinate */
        Pfloat    y;    /* y coordinate */
        Pfloat    z;    /* z coordinate */
} Ppoint3;
```

*anno_offset*

A pointer to a Pvec3 structure containing the *x*, *y*, and *z* coordinates of the offset of the text string from the transformed reference point. The annotation offset specifies an offset in NPC. The Pvec3 structure is defined in phigs.h as follows:

```
typedef struct  {
        Pfloat    delta_x;    /* x magnitude */
        Pfloat    delta_y;    /* y magnitude */
        Pfloat    delta_z;    /* z magnitude */
} Pvec3;
```

*text* A pointer to the character string to be written into the display.

**Execution**

When the structure is traversed, the ANNOTATION TEXT RELATIVE 3 element draws the specified character string on the plane in the NPC System defined by reference point and the annotation offset. These parameters define a Text Local Coordinate (TLC) System in the NPC System. The annotation offset added to the transformed reference point defines the origin of this TLC System. The *x* and *y* axes of the TLC System are parallel to and have the same direction as the *x* and *y* axes of the NPC System.

The precise position of the text is defined in relation to this plane by the current values of the text primitive attributes ANNOTATION TEXT CHARACTER UP VECTOR, ANNOTATION TEXT CHARACTER, BASE VECTOR, ANNOTATION TEXT PATH, and ANNOTATION TEXT ALIGNMENT. The reference point is subject to the current transformations in the transformation pipeline from the MC System to the workstation display. The text itself is only subject to the transformations in the transformation pipeline from the NPC System to the workstation display.

Other aspects of the appearance of the text are controlled by the attributes TEXT FONT, TEXT PRECISION, ANNOTATION STYLE, ANNOTATION TEXT CHARACTER HEIGHT, ANNOTATION TEXT CHARACTER WIDTH, CHARACTER EXPANSION FACTOR, CHARACTER SPACING, and TEXT COLOUR INDEX.

ANNOTATION TEXT CHARACTER WIDTH and ANNOTATION TEXT BASE VECTOR are implicit attributes derived from ANNOTATION TEXT CHARACTER HEIGHT and ANNOTATION TEXT UP VECTOR, respectively.

**Attributes Applied**

The attributes listed below are used to display the ANNOTATION TEXT RELATIVE 3 primitive when the structure is traversed. The Aspect Source Flags (ASFs) tell where to access the output display attributes. These attributes can come directly from the traversal state list, or they can be accessed indirectly, using the appropriate index in the traversal state list and the corresponding bundled representation in the PHIGS workstation state list.

| | |
|---|---|
| text font | text font ASF |
| text precision | text precision ASF |
| character expansion factor | character expansion factor ASF |
| character spacing | character spacing ASF |
| text colour | text colour index ASF |
| annotation character height | |
| annotation character up vector | |
| annotation text path | |
| annotation text alignment | |
| annotation style | |
| text index | |
| depth cue index | |
| name set | |

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

TEXT 3 (3P)

**NAME**

APPLICATION DATA – create a structure element containing application data

**SYNOPSIS**

**C Syntax**

```
void
pappl_data ( data )
Pdata          *data;          application data
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

The APPLICATION DATA subroutine puts a structure element containing application data into the currently open structure.

If the current edit mode is INSERT, the APPLICATION DATA element is inserted into the currently open structure after the element currently pointed to by the element pointer. If the edit mode is REPLACE, the APPLICATION DATA element replaces the element pointed to. In either case, the element pointer is updated to point to the new APPLICATION DATA element.

**C Input Parameter**

*data* A pointer to a Pdata structure containing the application data. Pdata is defined in phigs.h as follows:

```
typedef struct {
        size_t  size;     /* size of data */
        void    *data;    /* pointer to data */
} Pdata;
```

The size component specifies the number of bytes the data component is pointing to.

**Execution**

The APPLICATION DATA element is ignored during structure traversal.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

INQUIRE CURRENT ELEMENT TYPE AND SIZE (3P)
INQUIRE CURRENT ELEMENT CONTENT (3P)
INQUIRE ELEMENT TYPE AND SIZE (3P)
INQUIRE ELEMENT CONTENT (3P)

**NAME**

ARCHIVE ALL STRUCTURES – archive all structures into an archive file

**SYNOPSIS**

**C Syntax**

```
void
par_all_structs ( archive_id )
Pint            archive_id;          archive identifier
```

**Required PHIGS Operating States**

(PHOP, *, *, AROP)

**DESCRIPTION**

**Purpose**

Use ARCHIVE ALL STRUCTURES to archive all structures in the Central Structure Store to the specified open archive file.

**C Input Parameter**

*archive_id*

The archive identifier specifying the open archive file to write to.

**Execution**

All structures in the Central Structure Store are copied to the indicated open archive file. If any of the structures to be archived currently exist in the archive file, the conflict is resolved as follows:

- If the archival conflict resolution flag is MAINTAIN, the conflicting structure will not be copied into the archive (the archive contents are maintained)
- If the archival conflict resolution flag is UPDATE, the conflicting structure in the archive will be overwritten (the archive contents are updated)
- If the archival conflict resolution flag is ABANDON, no structures will be copied to the archive at all

The archival conflict resolution flag is set by the SET CONFLICT RESOLUTION subroutine.

**ERRORS**

| | |
|---|---|
| 007 | Ignoring function, function requires state (PHOP, *, *, AROP) |
| 404 | Ignoring function, the specified archive file is not open |
| 405 | Ignoring function, name conflict occurred while conflict resolution flag has value ABANDON |
| 406 | Warning, the archive file is full. Any structures that were archived were archived in total |

**SEE ALSO**

OPEN ARCHIVE FILE (3P)
SET CONFLICT RESOLUTION (3P)
ARCHIVE STRUCTURE NETWORKS (3P)
ARCHIVE STRUCTURES (3P)
DELETE ALL STRUCTURES FROM ARCHIVE (3P)

**NAME**

ARCHIVE STRUCTURE NETWORKS – archive specified structure networks into an archive file

**SYNOPSIS**

**C Syntax**

```
void
par_struct_nets ( archive_id, struct_ids )
Pint        archive_id;      archive identifier
Pint_list   *struct_ids;     list of structure identifiers
```

**Required PHIGS Operating States**

(PHOP, *, *, AROP)

**DESCRIPTION**

**Purpose**

Use ARCHIVE STRUCTURE NETWORKS to archive a list of structure networks in the Central Structure Store to the specified open archive file.

**C Input Parameters**

*archive_id*

The archive identifier specifying the open archive file to write to.

*struct_ids*

A pointer to a Pint_list structure containing the list of the root structure identifiers of the networks to be archived. The Pint_list structure is defined in phigs.h as follows:

```
typedef struct {
        Pint    num_ints;       /* number of Pints in list */
        Pint    *ints;          /* list of integers */
} Pint_list;
```

The *num_ints* component specifies the number of structure identifiers in the list. The *ints* component is a pointer to a list, *num_ints* long, of the structure identifiers.

**Execution**

The structures belonging to the specified networks are copied from the Central Structure Store to the indicated open archive file. If any of the structures to be archived currently exist in the archive file, the conflict is resolved as follows:

- If the archival conflict resolution flag is MAINTAIN, the conflicting structure will not be copied into the archive (the archive contents are maintained)
- If the archival conflict resolution flag is UPDATE, the conflicting structure in the archive will be overwritten (the archive contents are updated)
- If the archival conflict resolution flag is ABANDON, no structures will be copied to the archive at all

The archival conflict resolution flag is set by the SET CONFLICT RESOLUTION subroutine.

If any of the structures to be archived do not exist in the Central Structure Store, a warning is generated, and the archiving operation continues for the remaining structures.

**ERRORS**

| | |
|---|---|
| 007 | Ignoring function, function requires state (PHOP, *, *, AROP) |
| 404 | Ignoring function, the specified archive file is not open |
| 200 | Warning, ignoring structures that do not exist |
| 405 | Ignoring function, name conflict occurred while conflict resolution flag has value ABANDON |
| 406 | Warning, the archive file is full. Any structures that were archived were archived in total |

**SEE ALSO**

OPEN ARCHIVE FILE (3P)
SET CONFLICT RESOLUTION (3P)
ARCHIVE ALL STRUCTURES (3P)
ARCHIVE STRUCTURES (3P)
DELETE STRUCTURE NETWORKS FROM ARCHIVE (3P)

**NAME**

ARCHIVE STRUCTURES – archive specified structures into an archive file

**SYNOPSIS**

**C Syntax**

```
void
par_structs ( archive_id, struct_ids )
Pint            archive_id;         archive identifier
Pint_list       *struct_ids;        list of structure identifiers
```

**Required PHIGS Operating States**

(PHOP, *, *, AROP)

**DESCRIPTION**

**Purpose**

Use ARCHIVE STRUCTURES to archive a list of structures in the Central Structure Store to the specified open archive file.

**C Input Parameters**

*archive_id*

The archive identifier specifying the open archive file to write to.

*struct_ids*

A pointer to a Pint_list structure containing the list of structure identifiers to be archived. The Pint_list structure is defined in phigs.h as follows:

```
typedef struct {
        Pint    num_ints;       /* number of Pints in list */
        Pint    *ints;          /* list of integers */
} Pint_list;
```

The *num_ints* component specifies the number of structure identifiers in the list. The *ints* component is a pointer to a list, *num_ints* long, of the structure identifiers.

**Execution**

The specified structures are copied from the Central Structure Store to the indicated open archive file. If any of the structures to be archived currently exist in the archive file, the conflict is resolved as follows:

- If the archival conflict resolution flag is MAINTAIN, the conflicting structure will not be copied into the archive (the archive contents are maintained)
- If the archival conflict resolution flag is UPDATE, the conflicting structure in the archive will be overwritten (the archive contents are updated)
- If the archival conflict resolution flag is ABANDON, no structures will be copied to the archive at all

The archival conflict resolution flag is set by the SET CONFLICT RESOLUTION subroutine.

If any of the structures to be archived do not exist in the Central Structure Store, a warning is generated and the archiving operation continues for the remaining structures.

**ERRORS**

| | |
|---|---|
| 007 | Ignoring function, function requires state (PHOP, *, *, AROP) |
| 404 | Ignoring function, the specified archive file is not open |
| 200 | Warning, ignoring structures that do not exist |
| 405 | Ignoring function, name conflict occurred while conflict resolution flag has value ABANDON |
| 406 | Warning, the archive file is full. Any structures that were archived were archived in total |

**SEE ALSO**

OPEN ARCHIVE FILE (3P)
SET CONFLICT RESOLUTION (3P)
ARCHIVE STRUCTURE NETWORKS (3P)
ARCHIVE ALL STRUCTURES (3P)
DELETE STRUCTURES FROM ARCHIVE (3P)

**NAME**

AWAIT EVENT – move event from input queue to current event report

**SYNOPSIS**

**C Syntax**

**void**
**pawait_event ( timeout, ws_id, class, in_num )**

| | | |
|---|---|---|
| **Pfloat** | **timeout;** | *timeout ( seconds )* |
| **Pint** | ***ws_id;** | *OUT workstation identifier* |
| **Pin_class** | ***class;** | *OUT device class* |
| **Pint** | ***in_num;** | *OUT logical input device number* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

If the input event queue is not empty, AWAIT EVENT moves the next event in the queue to the current event report in the PHIGS state list. It returns the workstation identifier, input class, and input device number of the input device that generated the event.

If the input queue is empty, AWAIT EVENT suspends PHIGS until an event is entered into the queue or the time specified by *timeout* has elapsed.

**C Input Parameters**

*timeout* The time, in seconds, to wait for an event if none are present in the input queue.

**C Output Parameters**

*ws_id* If the event queue is not empty, PHIGS sets the variable pointed to by *ws_id* to the identifier of the workstation that was the source of the input event. If the queue is empty, PHIGS does not set this variable.

*class* If the event queue is not empty, PHIGS sets the variable pointed to by *class* to the class of the device that generated the input event. If the queue is empty, PHIGS sets this variable to PIN_NONE. Pin_class is an enumerated type that may take the following values:

```
typedef enum {
        PIN_NONE,
        PIN_LOC,
        PIN_STROKE,
        PIN_VAL,
        PIN_CHOICE,
        PIN_PICK,
        PIN_STRING
} Pin_class;
```

*in_num* A pointer to an integer variable. If the event queue is not empty, PHIGS sets this variable to the identifier of the input device that generated the input event.

**Execution**

The AWAIT EVENT function writes the logical input value of the oldest input event in the input queue to the current event report in the PHIGS state list. The workstation id, device class, and device number of the input device that generated the event are returned in the output parameters. The current event report is accessed by the appropriate GET device class function; for example, GET LOCATOR or GET STRING.

If there are no events in the input queue when AWAIT EVENT is called, AWAIT EVENT does not return until an event is added to the queue or until the time specified by the timeout parameter has elapsed. If a timeout occurs, PIN_NONE is returned as the device class and the other two values, workstation id and device number, are undefined.

Window system events such as repaints and resizes, and events for all active input devices are processed as usual while AWAIT EVENT is waiting for a timeout to expire.

While the timeout can be any positive floating-point value, the effective timeout resolution depends on the operating system in use.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

256 Warning, the input queue has overflowed

**SEE ALSO**

SET VALUATOR MODE (3P)
SET STROKE MODE (3P)
SET STRING MODE (3P)
SET PICK MODE (3P)
SET LOCATOR MODE (3P)
SET CHOICE MODE (3P)
GET CHOICE (3P)
GET LOCATOR (3P)
GET LOCATOR 3 (3P)
GET PICK (3P)
GET STRING (3P)
GET STROKE (3P)
GET STROKE 3 (3P)
GET VALUATOR (3P)
INQUIRE MORE SIMULTANEOUS EVENTS (3P)
INQUIRE INPUT QUEUE OVERFLOW (3P)

**NAME**

BUILD TRANSFORMATION MATRIX – generate a 2D transformation matrix to perform a transformation specified by a shift vector, rotation angle, and scale factors relative to a specified fixed point

**SYNOPSIS**

**C Syntax**

**void**
**pbuild_tran_matrix ( pt, shift, angle, scale, error_ind, matrix )**

| | | |
|---|---|---|
| **Ppoint** | ***pt;** | *fixed point* |
| **Pvec** | ***shift;** | *shift vector* |
| **Pfloat** | **angle;** | *rotation angle* |
| **Pvec** | ***scale;** | *scale vector* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pmatrix** | **matrix;** | *OUT transformation matrix* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use BUILD TRANSFORMATION MATRIX to calculate the 2D homogeneous $(3 \times 3)$ transformation matrix that performs the transformation specified by the input parameters.

The returned matrix may be passed as an argument to SET LOCAL TRANSFORMATION or SET GLOBAL TRANSFORMATION to modify the modelling transformation that is applied to output primitives during traversal.

**C Input Parameters**

*pt* *pt* points to the Ppoint structure containing the *x* and *y* coordinates of a fixed point in Modelling Coordinates. Scaling and rotation are performed relative to this fixed point. Ppoint is defined in phigs.h as follows:

```
typedef struct  {
        Pfloat    x;    /* x coordinate */
        Pfloat    y;    /* y coordinate */
} Ppoint;
```

*shift* *shift* points to a Pvec structure containing the *x* and *y* coordinates that define the shift (translation) applied by the transformation. Pvec is defined in phigs.h as follows:

```
typedef struct {
        Pfloat    delta_x;    /* x magnitude */
        Pfloat    delta_y;    /* y magnitude */
} Pvec;
```

*angle* This is the angle of rotation, in radians, applied by the transformation. A positive angle is a counter-clockwise rotation; a negative angle is clockwise rotation.

*scale* *scale* points to a Pvec structure containing *x* and *y* values defining the scale factors to be applied by the transformation.

**C Output Parameters**

*error_ind*

*error_ind* points to the location that stores the error number for any error detected by this function.

*matrix* A $3 \times 3$ homogeneous transformation matrix that performs the transformation defined by the input parameters. The matrix is returned in a Pmatrix array defined in phigs.h as follows:

```
typedef  Pfloat  Pmatrix[3][3];
```

**Execution**

BUILD TRANSFORMATION MATRIX returns a 2D $(3 \times 3)$ homogeneous transformation matrix that performs the transformation specified by the input parameters.

The transformation is performed in the following order:

- scale
- rotate
- shift

Scaling and rotation are done in relation to the fixed point.

**ERRORS**

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

**SEE ALSO**

SET LOCAL TRANSFORMATION (3P)
SET GLOBAL TRANSFORMATION (3P)
COMPOSE TRANSFORMATION MATRIX (3P)
COMPOSE MATRIX (3P)

**NAME**

BUILD TRANSFORMATION MATRIX 3 – generate a 3D transformation matrix to perform a transformation specified by a shift vector, rotation angles, and scale factors relative to a specified fixed point

**SYNOPSIS**

**C Syntax**

**void**
**pbuild_tran_matrix3 ( pt, shift, x_angle, y_angle, z_angle, scale, error_ind, matrix )**

| | | |
|---|---|---|
| **Ppoint3** | ***pt;** | *fixed point* |
| **Pvec3** | ***shift;** | *shift vector* |
| **Pfloat** | **x_angle;** | *rotation angle X* |
| **Pfloat** | **y_angle;** | *rotation angle Y* |
| **Pfloat** | **z_angle;** | *rotation angle Z* |
| **Pvec3** | ***scale;** | *scale vector* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pmatrix3** | **matrix;** | *OUT transformation matrix* |

**Required PHIGS Operating States**

(PHOP, ∗, ∗, ∗)

**DESCRIPTION**

**Purpose**

Use BUILD TRANSFORMATION MATRIX 3 to build a 3D homogeneous $(4 \times 4)$ transformation matrix that performs the transformation specified by the input parameters.

The returned matrix may be passed as an argument to SET LOCAL TRANSFORMATION 3 or SET GLOBAL TRANSFORMATION 3 to modify the modelling transformation applied to output primitives during traversal.

**C Input Parameters**

*pt* A pointer to a Ppoint3 structure containing the *x*, *y*, and *z* coordinates of a fixed point in Modelling Coordinates. Scaling and rotation are performed relative to this fixed point. The Ppoint3 structure is defined in phigs.h as follows:

```
typedef struct  {
        Pfloat    x;    /* x coordinate */
        Pfloat    y;    /* y coordinate */
        Pfloat    z;    /* z coordinate */
} Ppoint3;
```

*shift* A pointer to a Pvec3 structure containing *x*, *y*, and *z* coordinates defining the shift (translation) to be applied by the transformation. A Pvec3 structure is defined in phigs.h as follows:

```
typedef struct {
        Pfloat    delta_x;    /* x magnitude */
        Pfloat    delta_y;    /* y magnitude */
        Pfloat    delta_z;    /* z magnitude */
} Pvec3;
```

*x_angle*, *y_angle*, *z_angle*

The angles, in radians, of rotation around the *x*, *y*, and *z* axes to be applied by the transformation. Positive angles specify counter-clockwise rotations; negative angles specify clockwise rotations.

*scale* A pointer to a Pvec3 structure containing *x*, *y*, and *z* values defining the scale factors to be applied by the transformation.

**C Output Parameters**

*error_ind*

The error number of any error detected by this function.

*matrix* A $4 \times 4$ homogeneous transformation matrix that performs the transformation defined by the input parameters. The matrix is returned in a Pmatrix array defined in phigs.h as follows:

typedef Pfloat Pmatrix3[4][4];

**Execution**

BUILD TRANSFORMATION MATRIX 3 returns a 3D homogeneous $(4 \times 4)$ transformation matrix which performs the transformation specified by the values of the input parameters.

The transformation is performed in the following order:

- scale
- rotate
- shift

Scaling and rotation are done in relation to the fixed point defined by *pt*.

**ERRORS**

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

**SEE ALSO**

SET LOCAL TRANSFORMATION 3 (3P)
SET GLOBAL TRANSFORMATION 3 (3P)
COMPOSE TRANSFORMATION MATRIX 3 (3P)
COMPOSE MATRIX 3 (3P)

**NAME**

CELL ARRAY – create structure element specifying 2D cell array

**SYNOPSIS**

**C Syntax**

```
void
pcell_array ( rectangle, colr_array )
Prect          *rectangle;      cell rectangle
Ppat_rep       colr_array;      colour array
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

The CELL ARRAY function places a structure element containing the full specification of a 2D cell array into the currently open structure, according to the current edit mode. A 2D cell array primitive is an array of cells with individual colours. Its area is defined by two points in Modelling Coordinates giving the opposite corners of a rectangle, which is taken to be aligned with the modelling coordinate axes.

If the current edit mode is INSERT, the CELL ARRAY element is inserted into the open structure after the element pointed to by the element pointer. If the edit mode is REPLACE, the CELL ARRAY element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new CELL ARRAY element.

**C Input Parameters**

*rectangle*

A pointer to a Prect structure giving the lower left and upper right corners of the rectangle that defines the area of the CELL ARRAY element. The Prect structure is defined in phigs.h as:

```
typedef struct  {
        Ppoint  p;      /* lower left */
        Ppoint  q;      /* upper right */
} Prect;
```

The Ppoint structure is defined in phigs.h as:

```
typedef struct  {
        Pfloat  x;  /* x coordinate */
        Pfloat  y;  /* y coordinate */
} Ppoint;
```

*colr_array*

A pointer to the colour array. The Ppat_rep structure is defined in phigs.h as:

```
typedef struct  {
        Pint_size   dims;       /* patterns's dimensions */
        Pint        *colr_array; /* colour index array */
} Ppat_rep;
```

Pint_size is defined in phigs.h as follows:

```
typedef struct {
        Pint    size_x;         /* x size */
        Pint    size_y;         /* y size */
} Pint_size;
```

**Execution**

When the structure is traversed, the CELL ARRAY element draws the transformed boundaries of the cell rectangle, using the polyline attributes currently in effect. This is the minimal simulation for cell arrays. Applications should not depend on this behavior to remain the same in future releases.

The aspect source flags (ASFs) for line type, line width scale factor, and polyline colour index control whether the values used for these attributes are taken from the polyline representation specified by POLYLINE INDEX (ASF - BUNDLED), or from the individual specifications set for these attributes (ASF - INDIVIDUAL).

The coordinates used to specify the position of the CELL ARRAY primitive are Modelling Coordinates. These can be any coordinate units that are convenient to the application. At traversal, these coordinate values will be transformed by the current Local and Global Modelling Tranformations, the view representation selected by the current view index, and the workstation transformation current on the workstation to which the structure is posted.

**Attributes Applied**

The current values of the following attributes will be used to display the CELL ARRAY primitive when the structure is traversed. The value of an attribute is modified with the appropriate SET routine. The Aspect Source Flags (ASFs) tell where to access the output display attributes. These attributes can come directly from the traversal state list, or they can be accessed indirectly, using the appropriate index in the traversal state list and the corresponding bundled representation in the workstation state list.

| | |
|---|---|
| polyline colour | polyline colour index ASF |
| linewidth scale factor | linewidth scale factor ASF |
| linetype | linetype ASF |
| polyline shading method | polyline shading method ASF |
| polyline index | |
| depth cue index | |
| name set | |

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

113 Ignoring function, the colour index value is less than zero

117 Ignoring function, one of the dimensions of the colour index array is less than zero

**SEE ALSO**

CELL ARRAY 3 (3P)

**NAME**

CELL ARRAY 3 – create structure element specifying 3D cell array

**SYNOPSIS**

**C Syntax**

```
void
pcell_array3 ( parallelogram, colr_array )
Pparal      parallelogram;      cell parallelogram: [0]=P; [1]=Q; [2]=R
Ppat_rep    colr_array;         colour array
```

**Required PHIGS Operating States**

(PHOP, ∗, STOP, ∗)

**DESCRIPTION**

**Purpose**

The CELL ARRAY 3 function places a structure element containing the full specification of a 3D cell array into the currently open structure, according to the current edit mode. A 3D cell array primitive is an array of cells with individual colours. Its area is defined by three points P, Q, and R in Modelling Coordinates giving the corners of a parallelogram as P, Q, R, and (Q + R – P).

If the current edit mode is INSERT, the CELL ARRAY 3 element is inserted into the open structure after the element pointed to by the element pointer. If the edit mode is REPLACE, the CELL ARRAY 3 element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new CELL ARRAY 3 element.

**C Input Parameters**

*parallelogram*

A pointer to a Pparal structure giving the three corners of the parallelogram that defines the area of the CELL ARRAY 3 element. The Pparal structure is defined in phigs.h as follows:

```
typedef struct  {
        Ppoint3         p;      /* point p */
        Ppoint3         q;      /* point q */
        Ppoint3         r;      /* point r */
} Pparal;
```

Ppoint3 is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          x;      /* x coordinate */
        Pfloat          y;      /* y coordinate */
        Pfloat          z;      /* z coordinate */
} Ppoint3;
```

*colr_array*

The colour array. The Ppat_rep structure is defined in phigs.h as follows:

```
typedef struct  {
        Pint_size   dims;       /* pattern's dimensions */
        Pint        *colr_array;/* colour index array */
} Ppat_rep;
```

Pint_size is defined in phigs.h as follows:

```
typedef struct {
        Pint            size_x;         /* x size */
        Pint            size_y;         /* x size */
} Pint_size;
```

**Execution**

When the structure is traversed, the CELL ARRAY 3 element draws the transformed boundaries of the cell parallelogram, using the polyline attributes currently in effect. The aspect source flags (ASFs) for line type, line width scale factor, and polyline colour index control whether the values used for these attributes are taken from the polyline representation specified by POLYLINE INDEX (ASF - BUNDLED), or from the individual specifications set for these attributes (ASF - INDIVIDUAL). This is the minimal simulation for cell arrays. Applications should not depend on this behavior to remain the same in future releases.

The coordinates used to specify the position of the CELL ARRAY 3 primitive are Modelling Coordinates. These can be any coordinate units that are convenient to the application. At traversal, these coordinate values will be transformed by the current Local and Global Modelling Tranformations, the view representation selected by the current view index, and the workstation transformation current on the workstation to which the structure is posted.

**Attributes Applied**

The current values of the following attributes will be used to display the CELL ARRAY 3 primitive when the structure is traversed. The value of an attribute is modified with the appropriate SET routine. The aspect source flags (ASFs) tell where to access the output display attributes. These attributes can come directly from the traversal state list, or they can be accessed indirectly, using the appropriate index in the traversal state list and the corresponding bundled representation in the workstation state list.

| | |
|---|---|
| polyline colour | polyline colour index ASF |
| linewidth scale factor | linewidth scale factor ASF |
| linetype | linetype ASF |
| polyline shading method | polyline shading method ASF |
| polyline index | |
| depth cue index | |
| name set | |

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

113 Ignoring function, the colour index value is less than zero

117 Ignoring function, one of the dimensions of the colour index array is less than zero

**SEE ALSO**

CELL ARRAY (3P)

**NAME**

CELL ARRAY 3 PLUS – create a 3D extended cell array element

**SYNOPSIS**

**C Syntax**

**void**
**pcell_array3_plus ( paral, dim, colour_model, colrs )**

| | | |
|---|---|---|
| **Pparal** | ***paral;** | *cell parallelogram: [0]=P; [1]=Q; [2]=R* |
| **Pint_size** | ***dim;** | *dimension of cell: DX and DY* |
| **Pint** | **colour_model;** | *colour model* |
| **Pcoval** | ***colrs;** | *colour array* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

CELL ARRAY 3 PLUS creates a 3D cell array made of cells filled with general colour in each cell. Its area is defined by three points P, Q, and R in Modelling Coordinates giving the corners of a parallelogram as P, Q, R and (Q+R-P).

**C Input Parameters**

*paral* A pointer to an array of three Ppoint3 structures giving three corners of the parallelogram that defines the area of the CELL ARRAY 3 PLUS element. Pparal is defined in phigs.h as follows:

```
typedef struct {
        Ppoint3         p;              /* point p */
        Ppoint3         q;              /* point q */
        Ppoint3         r;              /* point r */
} Pparal;
```

*colour_model*

The colour model for specified cell colours.

| | | |
|---|---|---|
| 0 | PINDIRECT | *Colour Index Specified* |
| 1 | PMODEL_RGB | *Red, Green, Blue Colour Model* |
| 2 | PMODEL_CIELUV | *Commission Internationale de l'Eclairage Colour Model* |
| 3 | PMODEL_HSV | *Hue, Saturation, Value Colour Model* |
| 4 | PMODEL_HLS | *Hue, Lightness, Saturation Colour Model* |

**C Output Parameters**

*dim* A pointer to a Pdim structure specifying the dimensions of the colour cells array. Pint_size is defined in phigs.h as follows:

```
typedef struct {
        Pint            size_x;         /* dimension (number of divisions) along X */
        Pint            size_y;         /* dimension (number of divisions) along Y */
} Pint_size;
```

*colrs* A pointer to an array of Pcoval of dimension defined by *dim*, containing the colour values specifying cell colours. Pcoval is defined in phigs.h as follows:

```
typedef union {
        Pint            index;      /* index in workstation colour bundle table */
        Pcolr_rep       direct;     /* direct colour components */
} Pcoval;
```

Pcolr_rep is defined as:

```
typedef union {
        Prgb            rgb;        /* Red Green Blue colour specification */
        Pcieluv         cieluv;     /* CIE L*U*V* colour specification */
        Phls            hls;        /* Hue Lightness Saturation colour specification */
        Phsv            hsv;        /* Hue Saturation Value colour specification */
        Pdata           unsupp;     /* Colour in unsupported colour model */
} Pcolr_rep
```

Prgb is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          red;        /* red, hue, etc */
        Pfloat          green;      /* green, saturation, lightness, etc */
        Pfloat          blue;       /* blue, value, saturation, etc */
} Prgb;
```

Pcieluv is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          cieluv_x;       /* x coefficient */
        Pfloat          cieluv_y;       /* y coefficient */
        Pfloat          cieluv_y_lum;   /* y luminance */
} Pcieluv;
```

Phsv is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          hue;        /* hue */
        Pfloat          satur;      /* saturation */
        Pfloat          value;      /* value */
} Phsv;
```

Phls is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          hue;        /* hue */
        Pfloat          lightness;  /* lightness */
        Pfloat          satur;      /* saturation */
} Phls;
```

Pdata is defined in phigs.h as follows:

```
typedef struct {
        size_t          size;       /* size of data */
        char            *data       /* pointer to data */
} Pdata;
```

**Execution**

When the structure is traversed, the CELL ARRAY 3 PLUS element draws the transformed boundaries of the cell parallelogram, using the polyline attributes currently in effect. This is the minimal simulation for CELL

ARRAY 3 PLUS.

**Attributes Applied**

The attributes listed below are used to display the CELL ARRAY 3 PLUS primitive when the structure is traversed. The Aspect Source Flags (ASFs) tell where to access the output display attributes. These attributes can come directly from the traversal state list, or they can be accessed indirectly, using the appropriate index in the traversal state list and the corresponding bundled representation in the workstation state list.

| | |
|---|---|
| polyline colour | polyline colour index ASF |
| linewidth scale factor | linewidth scale factor ASF |
| linetype | linetype ASF |
| polyline shading method | polyline shading method ASF |
| polyline index | |
| depth cue index | |
| name set | |

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

117 Iqnoring function, one of the dimensions of the colour index array is less than zero

**SEE ALSO**

CELL ARRAY 3 (3P)

**NAME**

CHANGE STRUCTURE IDENTIFIER – change the identifier assigned to a structure

**SYNOPSIS**

**C Syntax**

```
void
pchange_struct_id ( orig_struct_id, result_struct_id )
Pint        orig_struct_id;        original structure id
Pint        result_struct_id;      result structure id
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use CHANGE STRUCTURE IDENTIFIER to change the identifier of a specified structure.

Subroutines which create new structures, for example OPEN STRUCTURE or EXECUTE STRUCTURE, include a parameter to assign an identifier for the newly created structure. The identifier is the number used to reference the structure in the Central Structure Store (CSS).

**C Input Parameters**

*orig_struct_id*

Specifies the structure for which the identifier is to be changed.

*result_struct_id*

Specifies the new identifier to assign to the structure.

**Execution**

The CHANGE STRUCTURE IDENTIFIER subroutine changes the identifier associated with *orig_struct_id* to *result_struct_id*. The subroutine does not change any references that may exist in other structures to the original or the resulting structure. The result of CHANGE STRUCTURE IDENTIFIER varies with the state of the original and resulting structures when the subroutine is called. The possible effects on the structures are described below.

*The Original Structure*

If there are no references to the original structure anywhere in the CSS, CHANGE STRUCTURE IDENTIFIER deletes the original structure identifier from the CSS. If there are references to the original structure, the original structure continues to exist after the subroutine returns, but it will be empty.

If original structure identifier is the open structure when CHANGE STRUCTURE IDENTIFIER is called, it continues to exist as the open structure after the subroutine returns, but it will be empty. The element pointer will be set to 0.

If the original structure is posted, it remains posted after CHANGE STRUCTURE IDENTIFIER, but it remains as an empty structure. The display reflects the change in the posted structure according to the workstation's display update state.

*The Resulting Structure*

The resulting structure always exists at the end of CHANGE STRUCTURE IDENTIFIER. If the resulting structure identifier does not exist when the subroutine is called, it is created and contains the elements of the original structure. If the original structure identifier does not exist when the subroutine is called, the resulting structure is empty.

If the resulting structure identifier already exists before CHANGE STRUCTURE IDENTIFIER, the subroutine will replace its contents with the contents of the original structure identifier. Any existing references to the resulting structure identifier are not changed.

If the resulting structure identifier is the open structure when the subroutine is called, CHANGE STRUCTURE IDENTIFIER closes the structure, replaces the contents with the elements from original structure identifier, and reopens the structure. The element pointer will be set to point to the last element.

If the resulting structure identifier is posted when CHANGE STRUCTURE IDENTIFIER is called, it will remain posted after the subroutine is complete. The display is changed to reflect the new contents of the structure according to the display update state of the workstation to which the structure is posted.

**ERRORS**

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

**SEE ALSO**

CHANGE STRUCTURE IDENTIFIER AND REFERENCES (3P)
CHANGE STRUCTURE REFERENCES (3P)

**NAME**

CHANGE STRUCTURE IDENTIFIER AND REFERENCES – change the identifier assigned to a structure and all references to it

**SYNOPSIS**

**C Syntax**

```
void
pchange_struct_id_refs ( orig_struct_id, result_struct_id )
Pint        orig_struct_id;        original structure id
Pint        result_struct_id;      result structure id
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

CHANGE STRUCTURE IDENTIFIER AND REFERENCES changes the identifier of a specified structure and all references to the original identifier to the new identifier. References may be both EXECUTE STRUCTURE elements and workstation postings.

**C Input Parameters**

*orig_struct_id*

Specifies the structure for which the identifier and references are to be changed.

*result_struct_id*

Specifies the new identifier to assign to the structure.

**Execution**

The CHANGE STRUCTURE IDENTIFIER AND REFERENCES subroutine changes the identifier associated with *orig_struct_id* to *result_struct_id*. This subroutine also changes all references to the original structure identifier to instead refer to the resulting structure identifier. The result of CHANGE STRUCTURE IDENTIFIER AND REFERENCES varies with the state of the original and resulting structures when the subroutine is called. The possible effects on the structures are described below.

Structure references are established by creating EXECUTE STRUCTURE elements or by posting a structure. A single call to CHANGE STRUCTURE IDENTIFIER AND REFERENCES changes both types of references to the specified structure. Details of how the subroutine changes each type of structure reference are provided below.

If original structure identifier and resulting structure identifier specify the same structure, the subroutine does not take any action.

*The Original Structure*

The structure identified by *orig_struct_id* will no longer exist after this subroutine is executed, unless it was the open structure. In this case, it continues to exist as the open structure after the subroutine returns, but it will be empty and the element pointer will be set to 0.

*The Resulting Structure*

The resulting structure always exists at the end of CHANGE STRUCTURE IDENTIFIER AND REFERENCES. If *result_struct_id* does not exist when the subroutine is called, it is created and contains the elements of the original structure. If *orig_struct_id* does not exist when the subroutine is called, the resulting structure is empty.

If *result_struct_id* already exists before CHANGE STRUCTURE IDENTIFIER AND REFERENCES, the subroutine will replace its contents with the contents of the original structure.

If the resulting structure is the open structure when the subroutine is called, CHANGE STRUCTURE IDENTIFIER AND REFERENCES closes the structure, replaces the contents with the elements from the original structure and reopens the structure. The element pointer will be set to point to the last element.

*How EXECUTE STRUCTURE References are Changed*

CHANGE STRUCTURE IDENTIFIER AND REFERENCES changes all EXECUTE STRUCTURE elements throughout the Central Structure Store that reference original structure identifier so that they reference resulting structure identifier instead. Any references to *result_struct_id* that already exist when CHANGE STRUCTURE IDENTIFIER AND REFERENCES is called will not be changed by the subroutine.

*How Posted Structures are Changed*

If *orig_struct_id* is posted to a workstation when CHANGE STRUCTURE IDENTIFIER AND REFERENCES is called, the subroutine unposts the original structure and posts *result_struct_id* with the same priority that the original structure had.

If *result_struct_id* is posted when CHANGE STRUCTURE IDENTIFIER AND REFERENCES is called, it will remain posted and its priority will not change. If *orig_struct_id* is also posted, it will be unposted.

Changes in a posted structure network are processed immediately and may immediately affect the display. The actual visual effects that appear on the display surface will depend on the workstation's current display update state.

**ERRORS**

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

**SEE ALSO**

CHANGE STRUCTURE IDENTIFIER (3P)
CHANGE STRUCTURE REFERENCES (3P)

**NAME**

CHANGE STRUCTURE REFERENCES – change all references to a specified structure to instead refer to another specified structure

**SYNOPSIS**

**C Syntax**

```
void
pchange_struct_refs ( orig_struct_id, result_struct_id )
Pint        orig_struct_id;        original structure id
Pint        result_struct_id;      result structure id
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purposer**

CHANGE STRUCTURE REFERENCES modifies the arrangement of the structure network by changing all references to *orig_struct_id* to refer to *result_struct_id* instead. The subroutine changes both EXECUTE STRUCTURE references and any posting to a workstation.

The effect is to substitute the resulting structure for the original structure wherever it appears in a structure network or on a workstation posting.

**C Input Parameters**

*orig_struct_id*

CHANGE STRUCTURE REFERENCES changes the references to this structure.

*result_struct_id*

Specifies the structure the original structure references will point to when CHANGE STRUCTURE REFERENCES returns.

**Execution**

The CHANGE STRUCTURE REFERENCES subroutine changes all references to the original structure identifier to refer instead to the resulting structure identifier. Structure references are established by creating EXECUTE STRUCTURE elements or by posting a structure. A single call to CHANGE STRUCTURE REFERENCES changes both types of references to the specified structure. The following details how the subroutine changes each type of structure reference.

*How EXECUTE STRUCTURE References are Changed*

CHANGE STRUCTURE REFERENCES changes all EXECUTE STRUCTURE elements throughout the Central Structure Store that reference *orig_struct_id* so that they reference *result_struct_id* instead. Any references to *result_struct_id* that already exist when CHANGE STRUCTURE REFERENCES is called will not be changed by the subroutine.

If the resulting structure does not exist when the subroutine is called, it will be created as an empty structure and the original structure references assigned to it.

If *orig_struct_id* and *result_struct_id* specify the same structure, the subroutine does not take any action.

*How Posted Structures are Changed*

If the original structure is posted to a workstation when CHANGE STRUCTURE REFERENCES is called, the subroutine unposts it and posts the resulting structure with the same priority that the original structure had.

If the resulting structure is posted when CHANGE STRUCTURE REFERENCES is called, it will remain posted and its priority will not change. If the original structure is also posted, it will be unposted.

Changes in a posted structure network are processed immediately and may immediately affect the display. The actual visual effects that appear on the display surface will depend on the workstation's current display update state.

**ERRORS**

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

**SEE ALSO**

CHANGE STRUCTURE IDENTIFIER (3P)
CHANGE STRUCTURE IDENTIFIER AND REFERENCES (3P)

**NAME**

CLOSE ARCHIVE FILE - close named archive file

**SYNOPSIS**

**C Syntax**

```
void
pclose_ar_file ( archive_id )
Pint            archive_id;          archive identifier
```

**Required PHIGS Operating States**

(PHOP, *, *, AROP)

**DESCRIPTION**

**Purpose**

Use CLOSE ARCHIVE FILE to close an archive file.

**C Input Parameter**

*archive_id*

An integer identifying the archive file to be closed.

**Execution**

If the archive associated with the specified archive file identifier has changed since it was opened, it is written to the file and the file is closed. If no changes were made, the file will be closed without being rewritten.

The archive file identifier is disassociated with the file it referred to and removed from the set of open archive files maintained in the PHIGS state list. The PHIGS archive state is changed to archive closed (ARCL) if no other archive files are open.

**ERRORS**

007 Ignoring function, function requires state (PHOP, *, *, AROP)

404 Ignoring function, the specified archive file is not open

**SEE ALSO**

INQUIRE ARCHIVE FILES (3P)
INQUIRE ARCHIVE STATE VALUE (3P)
OPEN ARCHIVE FILE (3P)

**NAME**

CLOSE PHIGS – close the PHIGS environment

**SYNOPSIS**

**C Syntax**

**void**
**pclose_phigs ()**

**Required PHIGS Operating States**

(PHOP, WSCL, STCL, ARCL)

**DESCRIPTION**

**Purpose**

CLOSE PHIGS closes the PHIGS environment and frees any resources associated with it.

Before calling CLOSE PHIGS, all open structures, archives, and workstations must be closed.

**C Input Parameters**

None

**Execution**

When CLOSE PHIGS is called:

- The PHIGS system state variable is set to PHCL
- The PHIGS state list becomes unavailable
- The workstation description tables become unavailable
- All PHIGS buffers are released
- All PHIGS files are closed

**ERRORS**

004 Ignoring function, function requires state (PHOP, WSCL, STCL, ARCL)

**SEE ALSO**

INQUIRE SYSTEM STATE VALUE (3P)
OPEN PHIGS (3P)

**NAME**

CLOSE STRUCTURE – close the currently open structure

**SYNOPSIS**

**C Syntax**

**void**
**pclose_struct ()**

**Required PHIGS Operating States**

(PHOP, ∗, STOP, ∗)

**DESCRIPTION**

**Purpose**

CLOSE STRUCTURE closes the currently open structure.

**C Input Parameters**

None

**Execution**

CLOSE STRUCTURE closes the open structure and sets the structure state to STCL.

**ERRORS**

005 Ignoring function, function requires state (PHOP, ∗, STOP, ∗)

**SEE ALSO**

INQUIRE OPEN STRUCTURE (3P)
INQUIRE STRUCTURE STATUS (3P)
OPEN STRUCTURE (3P)

**NAME**

CLOSE WORKSTATION – close a workstation

**SYNOPSIS**

**C Syntax**

```
void
pclose_ws ( ws_id )
Pint            ws_id;          workstation identifier
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

CLOSE WORKSTATION closes the specified workstation and releases all resources it uses.

**C Input Parameter**

*ws_id* The identifier of the workstation to close.

**Execution**

CLOSE WORKSTATION performs the following functions:

- An implicit UPDATE WORKSTATION is performed on the specified workstation, with the regeneration flag set to PERFORM
- The workstation state list is released
- The workstation identifier is deleted from the set of open workstations in the PHIGS state list
- The workstation identifier is deleted from the list of workstations to which structures are posted in the structure state list
- Any events in the input queue from devices on this workstation are removed from the queue
- The specific workstation description table for this workstation, created when the workstation was opened, becomes unavailable and the workstation type value associated with it becomes undefined
- The connection to this workstation is released
- If no other workstations are open, the PHIGS workstation operating state is set to WSCL

If an input queue overflow has been caused by a logical input device associated the workstation being closed, the error state list entry, identification of one of the logical input devices that caused an input queue overflow, will become undefined when the workstation is closed.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

256 Warning, the input queue has overflowed

**SEE ALSO**

INQUIRE SET OF OPEN WORKSTATIONS (3P)
INQUIRE WORKSTATION STATE VALUE (3P)
OPEN WORKSTATION (3P)

**NAME**

COMPOSE MATRIX – generate the composition of two homogeneous 2D matrices

**SYNOPSIS**

**C Syntax**

```
void
pcompose_matrix ( a, b, error_ind, m )
Pmatrix     a;          matrix a
Pmatrix     b;          matrix b
Pint        *error_ind; OUT error indicator
Pmatrix     m;          OUT result matrix
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use COMPOSE MATRIX to produce a 2D $(3 \times 3)$ homogeneous matrix that is the composition (matrix multiplication product) of two specified $3 \times 3$ matrices.

The returned matrix may be passed to the SET LOCAL TRANSFORMATION and SET GLOBAL TRANSFORMATION functions, or to any PHIGS function accepting a parameter of type Pmatrix.

**C Input Parameters**

*a*, *b* The 2D $(3 \times 3)$ matrices used to calculate the composition matrix. Matrices are specified as a Pmatrix array defined in phigs.h as follows:

```
typedef  Pfloat  Pmatrix[3][3];
```

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*m* The resulting composition matrix.

**Execution**

COMPOSE MATRIX returns a $3 \times 3$ matrix that is the result of multiplying the two specified matrices *a* and *b*:

*result* = *a* x *b*

**ERRORS**

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

**SEE ALSO**

COMPOSE MATRIX 3 (3P)
COMPOSE TRANSFORMATION MATRIX (3P)
BUILD TRANSFORMATION MATRIX (3P)
SET LOCAL TRANSFORMATION (3P)
SET GLOBAL TRANSFORMATION (3P)

**NAME**

COMPOSE MATRIX 3 – generate the composition of two homogeneous 3D matrices

**SYNOPSIS**

**C Syntax**

```
void
pcompose_matrix3 ( a, b, error_ind, m )
Pmatrix3        a;              matrix a
Pmatrix3        b;              matrix b
Pint            *error_ind;     OUT error indicator
Pmatrix3        m;              OUT result matrix
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use COMPOSE MATRIX 3 to produce a 3D $(4 \times 4)$ homogeneous matrix that is the composition (matrix multiplication product) of two specified $4 \times 4$ matrices.

The returned matrix may be passed to the SET LOCAL TRANSFORMATION 3 and SET GLOBAL TRANSFORMATION 3 functions, or to any PHIGS function accepting a parameter of type Pmatrix3.

**C Input Parameters**

*a*, *b* The 3D $(4 \times 4)$ matrices to use to calculate the composition matrix. Matrices are specified as a Pmatrix3 array defined in phigs.h as follows:

```
typedef Pfloat Pmatrix3[4][4];
```

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*m* The resulting composition matrix.

**Execution**

COMPOSE MATRIX 3 returns a 4 x 4 matrix that is the result of multiplying the two specified matrices *a* and *b*:

*result* = *a* x *b*

**ERRORS**

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

**SEE ALSO**

COMPOSE MATRIX (3P)
COMPOSE TRANSFORMATION MATRIX 3 (3P)
BUILD TRANSFORMATION MATRIX 3 (3P)
SET LOCAL TRANSFORMATION 3 (3P)
SET GLOBAL TRANSFORMATION 3 (3P)

**NAME**

COMPOSE TRANSFORMATION MATRIX – compose a 2D transformation matrix, which is the composition of a specified matrix and a transformation matrix defined by a fixed point, shift vector, rotation angle, and scale factors

**SYNOPSIS**

**C Syntax**

**void**
**pcompose_tran_matrix ( m, pt, shift, angle, scale, error_ind, result )**

| | | |
|---|---|---|
| **Pmatrix** | **m;** | *transformation matrix* |
| **Ppoint** | ***pt;** | *fixed point* |
| **Pvec** | ***shift;** | *shift vector* |
| **Pfloat** | **angle;** | *rotation angle* |
| **Pvec** | ***scale;** | *scale vector* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pmatrix** | **result;** | *OUT transformation matrix* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use COMPOSE TRANSFORMATION MATRIX to generate a 2D homogeneous $(3 \times 3)$ matrix that composes an existing $3 \times 3$ matrix with a transformation specified by 2D values for scaling, rotation, and translation about a fixed point.

The returned matrix may be passed as an argument to SET LOCAL TRANSFORMATION or SET GLOBAL TRANSFORMATION to modify the modelling transformation that is applied to output primitives during traversal.

**C Input Parameters**

*m* The 2D $(3 \times 3)$ homogeneous transformation matrix to use in the composition. This matrix is a Pmatrix type, defined in phigs.h as follows:

```
typedef Pfloat Pmatrix[3][3];
```

*pt* A pointer to a Ppoint structure containing the *x* and *y* coordinates of a fixed point in Modelling Coordinates. Scaling and rotation are performed relative to this fixed point. Ppoint is defined in phigs.h as follows:

```
typedef struct {
        Pfloat  x;   /* x coordinate */
        Pfloat  y;   /* y coordinate */
} Ppoint;
```

*shift* A pointer to a Pvec structure containing *x* and *y* coordinates defining the shift (translation) to be added to the transformation. Pvec is defined in phigs.h as follows:

```
typedef struct {
        Pfloat  delta_x;   /* x magnitude */
        Pfloat  delta_y;   /* y magnitude */
} Pvec;
```

*angle* The angle of rotation, in radians, to be added to the transformation. A positive angle is a counterclockwise rotation. A negative angle is clockwise rotation.

*scale* A pointer to a Pvec structure containing *x* and *y* values defining the scale factors to be applied to the transformation.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*result* The resulting 2D $(3 \times 3)$ transformation matrix.

**Execution**

COMPOSE TRANSFORMATION MATRIX returns the matrix that performs the transformation resulting from the composition of the input matrix and the matrix specified by the input parameters *shift*, *angle*, and *scale*. Rotation and scaling are calculated relative to the fixed point.

The composition is performed as:

*result* = *a* x *b*

Where *b* is the input matrix and *a* is the matrix built from the other input parameters.

The order of operations used to build the transformation matrix *b* is:

- scale
- rotate
- shift (translate)

**ERRORS**

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

**SEE ALSO**

COMPOSE TRANSFORMATION MATRIX 3 (3P)
COMPOSE MATRIX (3P)
BUILD TRANSFORMATION MATRIX (3P)
SET LOCAL TRANSFORMATION (3P)
SET GLOBAL TRANSFORMATION (3P)

**NAME**

COMPOSE TRANSFORMATION MATRIX 3 – compose a 3D transformation matrix, which is the composition of a specified matrix and a transformation matrix defined by a fixed point, shift vector, rotation angle, and scale factors

**SYNOPSIS**

**C Syntax**

**void**
**pcompose_tran_matrix3 ( m, pt, shift, x_ang, y_ang, z_ang, scale, error_ind, result )**

| | | |
|---|---|---|
| **Pmatrix3** | **m;** | *transformation matrix* |
| **Ppoint3** | ***pt;** | *fixed point* |
| **Pvec3** | ***shift;** | *shift vector* |
| **Pfloat** | **x_ang;** | *rotation angle X* |
| **Pfloat** | **y_ang;** | *rotation angle Y* |
| **Pfloat** | **z_ang;** | *rotation angle Z* |
| **Pvec3** | ***scale;** | *scale vector* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pmatrix3** | **result;** | *OUT transformation matrix* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use COMPOSE TRANSFORMATION MATRIX 3 to generate a 3D $(4 \times 4)$ matrix that composes an existing $4 \times 4$ matrix with a transformation specified by 3D values for scaling, rotation, and translation about a fixed point.

The returned matrix may be passed as an argument to SET LOCAL TRANSFORMATION 3 or SET GLOBAL TRANSFORMATION 3 to modify the modelling transformation applied to output primitives during traversal.

**C Input Parameters**

*m* The 3D $(4 \times 4)$ homogeneous transformation matrix to use in the composition. This matrix is a Pmatrix3 type, defined in phigs.h as follows:

```
typedef Pfloat Pmatrix3[4][4];
```

*pt* A pointer to a Ppoint3 structure containing the *x*, *y*, and *z* and coordinates of a fixed point in Modelling Coordinates. Scaling and rotation are performed relative to this fixed point. Ppoint3 is defined in phigs.h as follows:

```
typedef struct {
        Pfloat  x;   /* x coordinate */
        Pfloat  y;   /* y coordinate */
        Pfloat  z;   /* z coordinate */
} Ppoint3;
```

*shift* A pointer to a Pvec3 structure containing *x*, *y*, and *z* coordinates defining the shift (translation) to be added to the transformation. Pvec3 is defined in phigs.h as follows:

```
typedef struct {
        Pfloat  delta_x;   /* x magnitude */
        Pfloat  delta_y;   /* y magnitude */
        Pfloat  delta_z;   /* z magnitude */
} Pvec3;
```

*x_ang*, *y_ang*, *z_ang*
The angles of rotation about the indicated axes, in radians, to be added to the transformation. A positive angle is a counter-clockwise rotation. A negative angle is clockwise rotation.

*scale* A pointer to a Pvec3 structure containing *x*, *y*, and *z* values defining the scale factors to be applied

to the transformation.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*result* The resulting 3D $(4 \times 4)$ transformation matrix.

**Execution**

COMPOSE TRANSFORMATION MATRIX 3 returns the matrix that performs the transformation resulting from the composition of the input matrix and the matrix specified by the input parameters shift, angle, and scale. Rotation and scaling are calculated relative to the fixed point.

The composition is performed as:

*result* = *a* x *b*

Where *b* is the input matrix and *a* is the matrix built from the other input parameters.

The order of operations used to build the transformation matrix *b* is:

- scale
- rotate
- shift (translate)

**ERRORS**

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

**SEE ALSO**

COMPOSE TRANSFORMATION MATRIX (3P)
BUILD TRANSFORMATION MATRIX 3 (3P)
COMPOSE MATRIX 3 (3P)
SET LOCAL TRANSFORMATION 3 (3P)
SET GLOBAL TRANSFORMATION 3 (3P)

**NAME**

COMPUTE FILL AREA SET GEOMETRIC NORMAL – compute geometric normal of the fill area set

**SYNOPSIS**

**C Syntax**

**void**
**pcomp_fill_area_set_gnorm ( vflag, nfa, vdata, error_ind, normal )**

| | | |
|---|---|---|
| **Pint** | **vflag;** | *what vertex data is available* |
| **Pint** | **nfa;** | *number of fill areas* |
| **Pfacet_vdata_list3** | ***vdata;** | *fill area data* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pvec3** | ***normal;** | *OUT geometric normal* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

COMPUTE FILL AREA SET GEOMETRIC NORMAL computes the geometric normal of the fill area set.

This is a PHIGS Extension function based on PHIGS PLUS and is not part of the PHIGS standard.

**C Input Parameters**

All of the following data types are predefined in phigs.h.

*vflag* The data per vertex flag specifies the available data for each vertex of the primitive.

| | | |
|---|---|---|
| 0 | PVERT_COORD | *Coordinates Specified* |
| 1 | PVERT_COORD_COLOUR | *Coordinates and Vertex Colour Specified* |
| 2 | PVERT_COORD_NORMAL | *Coordinates and Vertex Normal Specified* |
| 3 | PVERT_COORD_COLOUR_NORMAL | *Coordinates, Vertex Colour, and Vertex Normal Specified* |

*nfa* The number of fill areas in the set.

*vdata* A pointer, to a list of nfa Pfacet_vdata_list3 structures, that specifies the fill area set and optionally associated colour information. Pfacet_vdata_list3 is defined as follows:

```
typedef struct {
        Pint                num_vertices;   /* number of vertices */
        Pfacet_vdata_arr3   vertex_data;    /* array of facet vertex data */
} Pfacet_vdata_list3;
```

Pfacet_vdata_arr3 is defined as follows:

```
typedef union {
        Ppoint3         *points;        /* point */
        Pptco3          *ptcolrs;       /* point and color */
        Pptnorm3        *ptnorms;       /* point and normal */
        Pptconorm3      *ptconorms;     /* point, color, and norml */
        /* implementation dependent types can go here */
} Pfacet_vdata_arr3;
```

Ppoint3 is defined as follows:

```
typedef struct {
        Pfloat    x;    /* x coordinate */
        Pfloat    y;    /* y coordinate */
```

```
    Pfloat    z;    /* z coordinate */
} Ppoint3;
```

Pptco3 is defined as follows:

```
typedef struct {
    Ppoint3    point;    /* point coordinates */
    Pcoval     colr;     /* colour */
} Pptco3;
```

Ppoint3 is defined above. Pcoval is defined as follows:

```
typedef union {
    Pint          index;     /* index in workstation colour bundle table */
    Pcolr_rep     direct;    /* direct colour components */
} Pcoval;
```

Pcolr_rep is defined as follows:

```
typedef union {
    Prgb          rgb;          /* Red Green Blue colour specification */
    Pcieluv       cieluv;       /* CIE L*U*V* colour specification */
    Phls          hls;          /* Hue Lightness Saturation colour specification */
    Phsv          hsv;          /* Hue Saturation Value colour specification */
    Pdata         unsupp;       /* Colour in unsupported colour model */
} Pcolr_rep
```

Prgb is defined in phigs.h as follows:

```
typedef struct {
    Pfloat        red;          /* red, hue, etc */
    Pfloat        green;        /* green, saturation, lightness, etc */
    Pfloat        blue;         /* blue, value, saturation, etc */
} Prgb;
```

Pcieluv is defined in phigs.h as follows:

```
typedef struct {
    Pfloat        cieluv_x;     /* x coefficient */
    Pfloat        cieluv_y;     /* y coefficient */
    Pfloat        cieluv_y_lum; /* y luminance */
} Pcieluv;
```

Phsv is defined in phigs.h as follows:

```
typedef struct {
    Pfloat        hue;          /* hue */
    Pfloat        satur;        /* saturation */
    Pfloat        value;        /* value */
} Phsv;
```

Phls is defined in phigs.h as follows:

```
typedef struct {
    Pfloat        hue;          /* hue */
    Pfloat        lightness;    /* lightness */
    Pfloat        satur;        /* saturation */
```

```
} Phls;
```

Pdata is defined in phigs.h as follows:

```
typedef struct {
        size_t          size;           /* size of data */
        char            *data           /* pointer to data */
} Pdata;
```

Pptnorm3 is defined as follows:

```
typedef struct {
        Ppoint3  point;     /* point coordinates */
        Pvec3    normal;    /* norm */
} Pptnorm3;
```

Ppoint3 is defined above. Pvec3 is defined as follows:

```
typedef struct {
        Pfloat          x_val;          /* x magnitude */
        Pfloat          y_val;          /* y magnitude */
        Pfloat          z_val;          /* z magnitude */
} Pvec3;
```

Pptconorm3 is defined as follows:

```
typedef struct {
        Ppoint3         point;          /* point coordinates */
        Pcoval          colr;           /* colour */
        Pvec3           normal;         /* norm */
} Pptconorm3;
```

Ppoint3, Pcoval, and Pvec3 are defined above.

**C Output Parameters**

*error_ind*
A pointer to the location to store the error number.

*norm* A pointer to the location to store the computed geometric normal. (Pvec3 is defined above.)

**Execution**

The normalized geometric norm is computed for the fill area set defined by the supplied vertex coordinates. The geometric normal is computed by first selecting three points: *A*, *B*, and *C*. Point *A* is the first point in the first list of vertices. Point *B* is the next point in that list that is noncoincident with *A*. Point *C* is the next point in that list, after *B*, that is noncolinear with *A* and *B*. The geometric normal is the cross product of the vector extending from *A* to *B* with the vector extending from *A* to *C*.

If it is not possible to find three such points in the first list of vertices, then the rest of the lists are searched in order to select three appropriate points from a single list. In case the search fails in all lists, then it is determined that the fill area set is degenerate.

**ERRORS**

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

612 Warning, the fill area is degenerate

**SEE ALSO**

FILL AREA SET 3 (3P)
FILL AREA SET 3 WITH DATA (3P+)

**NAME**

COPY ALL ELEMENTS FROM STRUCTURE – copy elements of specified structure into open structure

**SYNOPSIS**

**C Syntax**

```
void
pcopy_all_elems_struct ( struct_id )
Pint            struct_id;          structure identifier
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

COPY ALL ELEMENTS FROM STRUCTURE copies the structure elements of the structure identifier into the currently open structure. The elements are copied into the open structure after the element pointed to by the element pointer.

**C Input Parameter**

*struct_id*

Identifies the structure from which to copy the elements.

**Execution**

The COPY ALL ELEMENTS FROM STRUCTURE subroutine copies all the structure elements in the source structure identifier and inserts them into the currently open structure after the element pointed to by the element pointer. The element pointer is then updated to point to the last element inserted. The current edit mode has no effect on this subroutine.

If source structure identifier is the open structure, its contents are copied into itself after the element pointed to by the element pointer.

If source structure identifier refers to an empty structure or to a structure that does not exist, the subroutine does not perform any action and does not generate an error.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

CHANGE STRUCTURE IDENTIFIER (3P)
EXECUTE STRUCTURE (3P)

**NAME**

CREATE STORE – creates a Store, and returns a handle to it

**SYNOPSIS**

**C Syntax**

```
void
pcreate_store ( err, store )
Pint            *err;           OUT Error
Pstore          *store;         OUT Handle to store object
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

CREATE STORE creates a Store object and returns the handle to it. Store objects are used to hold the return values of complex inquiry functions.

**C Output Parameter**

*err* The status of the creation of the new Store.

*store* The handle to the newly created Store.

**Execution**

CREATE STORE creates a new Store and returns a handle to it via the *store* parameter. The parameter *err* returns the error number of any error detected. If a zero is returned the creation was successful.

A Store is an opaque pointer that is passed as a parameter to a function returning complex data. A Store contains the memory to hold the results of a complex inquiry. A Store may be passed to or used by more than one inquiry function, but at any one time a Store only holds the result of the last inquiry function that used it as an argument. A Store continues to hold the information from the last inquiry function until it is freed by DELETE STORE, or until the Store is used as an argument to a subsequent inquiry function, in which case the information is overwritten.

**ERRORS**

900

**SEE ALSO**

DELETE STORE (3P)

**NAME**

DELETE ALL STRUCTURES – remove all existing structures from the Central Structure Store

**SYNOPSIS**

**C Syntax**

**void**
**pdel_all_structs ()**

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use DELETE ALL STRUCTURES to remove all the existing structures from the Central Structure Store.

**C Input Parameters**

None

**Execution**

The DELETE ALL STRUCTURES subroutine removes all the existing structures from the Central Structure Store. To do this, DELETE ALL STRUCTURES executes the DELETE STRUCTURE function for each existing structure. All structure identifiers and structure elements are deleted, with the exception of an open structure.

If a structure is open, DELETE ALL STRUCTURES retains the structure with its current structure identifier, but removes all the structure elements from it. The result is to replace the open structure with an empty structure.

Any structure that is posted to a workstation is unposted.

**ERRORS**

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

**SEE ALSO**

DELETE STRUCTURE (3P)
DELETE STRUCTURE NETWORK (3P)
DELETE ALL STRUCTURES FROM ARCHIVE (3P)

**NAME**

DELETE ALL STRUCTURES FROM ARCHIVE – delete all structures from an archive file

**SYNOPSIS**

**C Syntax**

```
void
pdel_all_structs_ar ( archive_id )
Pint            archive_id;          archive identifier
```

**Required PHIGS Operating States**

(PHOP, *, *, AROP)

**DESCRIPTION**

**Purpose**

Use DELETE ALL STRUCTURES FROM ARCHIVE to delete all structures in the specified open archive file.

**C Input Parameter**

*archive_id*

The archive identifier specifying the open archive file to delete from.

**Execution**

All structures in the specified archive file are deleted and the archive file is left in the same state as if it had just been created.

**ERRORS**

007 Ignoring function, function requires state (PHOP, *, *, AROP)

404 Ignoring function, the specified archive file is not open

**SEE ALSO**

DELETE ALL STRUCTURES (3P)
DELETE STRUCTURE NETWORKS FROM ARCHIVE (3P)
DELETE STRUCTURES FROM ARCHIVE (3P)

**NAME**

DELETE ELEMENT – delete structure element

**SYNOPSIS**

**C Syntax**

**void**
**pdel_elem ()**

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

DELETE ELEMENT deletes the structure element currently pointed to by the element pointer in the open structure.

**C Input Parameters**

None

**Execution**

The DELETE ELEMENT subroutine removes the structure element currently pointed to by the element pointer in the open structure and renumbers the remaining elements in the structure to maintain a consecutive, ascending order in the structure.

When the element is deleted, the element pointer is set to point to the element preceding the deleted element in the structure.

If the element pointer is 0 (pointing to the position in the structure before any elements) when DELETE ELEMENT is called, the subroutine will not delete any element and will not adjust the element pointer.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

SET EDIT MODE (3P)
EMPTY STRUCTURE (3P)
DELETE ELEMENT RANGE (3P)
DELETE ELEMENTS BETWEEN LABELS (3P)

**NAME**

DELETE ELEMENT RANGE – delete a block of elements in a structure

**SYNOPSIS**

**C Syntax**

```
void
pdel_elem_range ( ep1_value, ep2_value )
Pint        ep1_value;      element pointer 1 value
Pint        ep2_value;      element pointer 2 value
```

**Required PHIGS Operating States**

(PHOP, ∗, STOP, ∗)

**DESCRIPTION**

**Purpose**

Use DELETE ELEMENT RANGE to delete all structure elements in the open structure between, and including, two specified element numbers.

**C Input Parameters**

*ep1_value*

Specifies the beginning of the element range to be deleted.

*ep2_value*

Specifies the end of the element range to be deleted.

**Execution**

The DELETE ELEMENT RANGE subroutine removes structure elements from the open structure between and including the lower and the higher of the element positions specified by element pointer 1 value and element pointer 2 value. The remaining elements in the structure are renumbered and the element pointer is updated to point to the element preceding the deleted elements.

If an element position is less than 1, the range of elements to be deleted will start from element position 0. If an element position is greater than the number of elements in the open structure, the range of elements to be deleted will end at the last element of the open structure.

**ERRORS**

005 Ignoring function, function requires state (PHOP, ∗, STOP, ∗)

**SEE ALSO**

DELETE ELEMENT (3P)
DELETE ELEMENTS BETWEEN LABELS (3P)
EMPTY STRUCTURE (3P)

**NAME**

DELETE ELEMENTS BETWEEN LABELS – delete all elements in the open structure between specified labels

**SYNOPSIS**

**C Syntax**

```
void
pdel_elems_labels ( label1_id, label2_id )
Pint        label1_id;      label 1 identifier
Pint        label2_id;      label 2 identifier
```

**Required PHIGS Operating States**

(PHOP, ∗, STOP, ∗)

**DESCRIPTION**

**Purpose**

Use DELETE ELEMENTS BETWEEN LABELS to delete structure elements in the open structure between two specified labels. The elements containing the labels are not deleted.

**C Input Parameters**

*label1_id*

The subroutine begins deleting structures with the structure element immediately following this label.

*label2_id*

The subroutine ends deleting structures with the structure element immediately preceding this label.

**Execution**

The DELETE ELEMENTS BETWEEN LABELS subroutine removes all the elements in the open structure between the two specified labels. The remaining elements are renumbered and the element pointer is updated to point to label 1 identifier.

The subroutine searches for the specified labels only from the current position of the element pointer in the structure to the end of the structure. The next occurrence of an element containing label 1 identifier is first found; then, starting from the element containing label 1 identifier, the next occurrence of an element containing label 2 identifier is found.

If either of the label identifiers cannot be found between the current position of the element pointer and the end of the structure, no elements are deleted and an error is generated.

**ERRORS**

005 Ignoring function, function requires state (PHOP, ∗, STOP, ∗)

206 Ignoring function, one or both of the labels does not exist in the open structure between the element pointer and the end of the structure

**SEE ALSO**

DELETE ELEMENT (3P)
DELETE ELEMENT RANGE (3P)
EMPTY STRUCTURE (3P)

**NAME**

DELETE STORE – Destroys a Store object

**SYNOPSIS**

**C Syntax**

```
void
pdel_store ( store )
Pstore          store;          Handle to store object
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

DELETE STORE destroys the specified Store object and frees all memory associated with it.

**C Input Parameter**

*store* The Store to be deleted.

**Execution**

DELETE STORE frees all memory associated with a Store. A Store is an opaque pointer that is passed as a parameter to a function returning complex data. A Store contains the memory to hold the results of a complex inquiry. A Store may be passed to or used by more than one inquiry function, but at any one time a Store only holds the result of the last inquiry function that used it as an argument. A Store continues to hold the information from the last inquiry function until it is freed by DELETE STORE, or until the Store is used as an argument to a subsequent inquiry function, in which case the information is overwritten.

**ERRORS**

None.

**SEE ALSO**

CREATE STORE (3P)

**NAME**

DELETE STRUCTURE – remove specified structure

**SYNOPSIS**

**C Syntax**

```
void
pdel_struct ( struct_id )
Pint            struct_id;          structure identifier
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

DELETE STRUCTURE removes the specified structure from the Central Structure Store (CSS), along with all references to the structure and all postings of the structure to workstations.

If the specified structure is closed, the subroutine removes the structure elements, structure identifier, and any references to the structure.

If the structure is the currently open structure, the structure identifier is retained and remains open, but the structure contents and any references to the structure are deleted.

**C Input Parameter**

*struct_id*

Specifies the structure to be deleted.

**Execution**

The effect of DELETE STRUCTURE when the specified structure is closed at the time the subroutine is called is different from the effect when structure identifier is the open structure. The two situations are described below.

*Deleting a Closed Structure*

If structure identifier is not the currently open structure, the DELETE STRUCTURE subroutine removes the specified structure from the PHIGS Central Structure Store. The subroutine deletes the structure identifier, structure contents, and all references to structure identifier contained in other structures.

If the currently open structure contains an element referencing structure identifier, the element in the open structure is deleted and the remaining elements renumbered. If the element pointer is pointing to the deleted element, the pointer is set to the preceding element. If the element pointer is pointing to an element following the deleted element, the element pointer is updated such that it still refers to the same element.

*Deleting the Open Structure*

If structure identifier is the currently open structure, the result of DELETE STRUCTURE is to replace the open structure with an empty, unreferenced structure. The effect is the same as calling the following subroutines in sequence:

- CLOSE STRUCTURE
- DELETE STRUCTURE
- OPEN STRUCTURE

**ERRORS**

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

**SEE ALSO**

DELETE ALL STRUCTURES (3P)
DELETE STRUCTURE NETWORK (3P)
DELETE STRUCTURES FROM ARCHIVE (3P)
EMPTY STRUCTURE (3P)

**NAME**

DELETE STRUCTURE NETWORK – delete network of structures from central structure store

**SYNOPSIS**

**C Syntax**

```
void
pdel_struct_net ( struct_id, ref_flag )
Pint            struct_id;          structure identifier
Pref_flag       ref_flag;           reference handling flag
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use DELETE STRUCTURE NETWORK to remove from the Central Structure Store a specified structure and all structures referenced directly or indirectly by that structure.

A structure is referenced from within another structure by an EXECUTE STRUCTURE element created by the EXECUTE STRUCTURE subroutine. A structure network is a hierarchical arrangement of structures in which the original structure references subordinate structures which, in turn, reference other structures, and so on.

More than one EXECUTE STRUCTURE element can refer to the same structure, so that a structure referenced in the network to be deleted can also be referenced by structures in other networks. DELETE STRUCTURE NETWORK includes a reference handling flag parameter which controls whether to delete or keep these structures.

**C Input Parameters**

*struct_id*

Specifies the structure which is the beginning of the structure network to be deleted.

*ref_flag* Controls how structures in the selected network that are also referenced by other structures outside the network are to be handled. This is an enumerated type defined in phigs.h as follows:

| | |
|---|---|
| PFLAG_DEL | *Delete all structures* |
| PFLAG_KEEP | *Keep structures referenced outside the network* |

The use of these values is described in the *Execution* section below.

**Execution**

The DELETE STRUCTURE NETWORK subroutine deletes the structure specified by *struct_id* and the structure network beginning with that structure. The structure network is the chain of structures referenced by EXECUTE STRUCTURE elements in *struct_id* and its descendent structures.

If *struct_id* does not exist, the subroutine takes no action and does not return an error.

The reference handling flag parameter controls how structures that are referenced both by this structure network and by another structure outside the network are handled.

- If reference handling flag is PFLAG_DEL, DELETE STRUCTURE NETWORK removes all structures referenced in the structure network beginning with structure identifier even if those structures are also referenced by another structure outside this network.
- If reference handling flag is PFLAG_KEEP, DELETE STRUCTURE NETWORK will **not** remove structures referenced in this network which are also referenced by another structure outside this network. However, the structure specified by *struct_id* identifier is always deleted.

All structures that are actually deleted are deleted as if DELETE STRUCTURE were called for each individual structure; that is, the structure identifier, its contents, and all references to it are removed.

**ERRORS**

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

**SEE ALSO**

DELETE ALL STRUCTURES (3P)
DELETE STRUCTURE (3P)
DELETE STRUCTURE NETWORKS FROM ARCHIVE (3P)

**NAME**

DELETE STRUCTURE NETWORKS FROM ARCHIVE – delete specified structure networks from an archive file

**SYNOPSIS**

**C Syntax**

```
void
pdel_struct_nets_ar ( archive_id, struct_ids )
Pint          archive_id;          archive identifier
Pint_list     *struct_ids;         list of structure identifiers
```

**Required PHIGS Operating States**

(PHOP, *, *, AROP)

**DESCRIPTION**

**Purpose**

Use DELETE STRUCTURE NETWORKS FROM ARCHIVE to delete a list of structure networks from the specified open archive file.

**C Input Parameters**

*archive_id*

The archive identifier specifying the open archive file to delete from.

*struct_ids*

A pointer to a Pint_list structure containing the list of the root structure identifiers of the archived structure networks to be deleted. The Pint_list structure is defined in phigs.h as follows:

```
typedef struct {
        Pint    num_ints;       /* number of Pints in list */
        Pint    *ints;          /* list of integers */
} Pint_list;
```

The *num_ints* component specifies the number of structure identifiers in the list. The *ints* component is a pointer to a list, *num_ints* long, of the structure identifiers.

**Execution**

The structures belonging to the specified networks are deleted from the archive file. No attempt is made to ensure that the deleted structures are not referenced from other structures in the archive file.

**ERRORS**

007 Ignoring function, function requires state (PHOP, *, *, AROP)

404 Ignoring function, the specified archive file is not open

407 Warning, some of the specified structures do not exist on the archive file

**SEE ALSO**

DELETE ALL STRUCTURES FROM ARCHIVE (3P)
DELETE STRUCTURES FROM ARCHIVE (3P)
DELETE STRUCTURE NETWORK (3P)

**NAME**

DELETE STRUCTURES FROM ARCHIVE – delete specified structures from an archive file

**SYNOPSIS**

**C Syntax**

```
void
pdel_structs_ar ( archive_id, struct_ids )
Pint            archive_id;          archive identifier
Pint_list       *struct_ids;         list of structure identifiers
```

**Required PHIGS Operating States**

(PHOP, *, *, AROP)

**DESCRIPTION**

**Purpose**

Use DELETE STRUCTURES FROM ARCHIVE to delete a list of structures from the specified open archive file.

**C Input Parameters**

*archive_id*

The archive identifier specifying the open archive file to delete from.

*struct_ids*

A pointer to a Pint_list structure containing the list of structure identifiers to be deleted. The Pint_list structure is defined in phigs.h as follows:

```
typedef struct {
        Pint    num_ints;       /* number of Pints in list */
        Pint    *ints;          /* list of integers */
} Pint_list;
```

The *num_ints* component specifies the number of structure identifiers in the list. The *ints* component is a pointer to a list, *num_ints* long, of the structure identifiers.

**Execution**

The specified structures are deleted from the archive file. No attempt is made to ensure that the deleted structures are not referenced from other structures in the archive file.

**ERRORS**

007 Ignoring function, function requires state (PHOP, *, *, AROP)

404 Ignoring function, the specified archive file is not open

407 Warning, some of the specified structures do not exist on the archive file

**SEE ALSO**

DELETE ALL STRUCTURES FROM ARCHIVE (3P)
DELETE STRUCTURE NETWORKS FROM ARCHIVE (3P)
DELETE STRUCTURE (3P)

**NAME**

ELEMENT SEARCH – search for next matching element in specified structure

**SYNOPSIS**

**C Syntax**

**void**
**pelem_search ( struct_id, start_el, dir, incl, excl, error_ind, status, found_el )**

| | | |
|---|---|---|
| **Pint** | **struct_id;** | *structure identifier* |
| **Pint** | **start_el;** | *starting element pointer* |
| **Psearch_dir** | **dir;** | *search direction* |
| **Pelem_type_list** | ***incl;** | *element incl. list* |
| **Pelem_type_list** | ***excl;** | *element excl. list* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Psearch_status** | ***status;** | *OUT search status* |
| **Pint** | ***found_el;** | *OUT found element pointer* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

ELEMENT SEARCH searches the specified structure for the next element matching the search criteria. The search starts at the specified element (or the limit of the structure) and proceeds in the direction specified. The search terminates if an element is found, or if the opposite limit of the structure is reached.

The search criteria is based on element types. An element satisfies the criteria if its element type is in the inclusion set, and *not* in the exclusion set.

**C Input Parameters**

*struct_id*
Identifier of the structure to search.

*start_el* Position in the specified structure of the first element tested against the search criteria.

*dir* Search direction. Psearch_dir is an enumerated type defined in phigs.h to have the following values:

PDIR_BACKWARD
PDIR_FORWARD

*incl* A pointer to a Pelem_type_list structure containing the list of element types in the inclusion set.

The Pelem_type_list structure and its sub-types are defined in phigs.h as:

```
typedef struct {
        Pint            num_elem_types;   /* number of element types */
        Pelem_type      *elem_types;      /* list of element types */
} Pelem_type_list;
```

The Pelem_type enumeration lists all the supported element types. It is enumerated in phigs.h as follows:

```
typedef enum {
   PELEM_ALL,
   PELEM_NIL,
   PELEM_POLYLINE3,
   PELEM_POLYLINE,
   PELEM_POLYMARKER3,
   PELEM_POLYMARKER,
   PELEM_TEXT3,
```

PELEM_TEXT,
PELEM_ANNO_TEXT_REL3,
PELEM_ANNO_TEXT_REL,
PELEM_FILL_AREA3,
PELEM_FILL_AREA,
PELEM_FILL_AREA_SET3,
PELEM_FILL_AREA_SET,
PELEM_CELL_ARRAY3,
PELEM_CELL_ARRAY,
PELEM_GDP3,
PELEM_GDP,
PELEM_LINE_IND,
PELEM_MARKER_IND,
PELEM_TEXT_IND,
PELEM_INT_IND,
PELEM_EDGE_IND,
PELEM_LINETYPE,
PELEM_LINEWIDTH,
PELEM_LINE_COLR_IND,
PELEM_MARKER_TYPE,
PELEM_MARKER_SIZE,
PELEM_MARKER_COLR_IND,
PELEM_TEXT_FONT,
PELEM_TEXT_PREC,
PELEM_CHAR_EXPAN,
PELEM_CHAR_SPACE,
PELEM_TEXT_COLR_IND,
PELEM_CHAR_HT,
PELEM_CHAR_UP_VEC,
PELEM_TEXT_PATH,
PELEM_TEXT_ALIGN,
PELEM_ANNO_CHAR_HT,
PELEM_ANNO_CHAR_UP_VEC,
PELEM_ANNO_PATH,
PELEM_ANNO_ALIGN,
PELEM_ANNO_STYLE,
PELEM_INT_STYLE,
PELEM_INT_STYLE_IND,
PELEM_INT_COLR_IND,
PELEM_EDGE_FLAG,
PELEM_EDGETYPE,
PELEM_EDGEWIDTH,
PELEM_EDGE_COLR_IND,
PELEM_PAT_SIZE,
PELEM_PAT_REF_POINT_VECS,
PELEM_PAT_REF_POINT,
PELEM_ADD_NAMES_SET,
PELEM_REMOVE_NAMES_SET,
PELEM_INDIV_ASF,
PELEM_HLHSR_ID,
PELEM_LOCAL_MODEL_TRAN3,
PELEM_LOCAL_MODEL_TRAN,
PELEM_GLOBAL_MODEL_TRAN3,

```
    PELEM_GLOBAL_MODEL_TRAN,
    PELEM_MODEL_CLIP_VOL3,
    PELEM_MODEL_CLIP_VOL,
    PELEM_MODEL_CLIP_IND,
    PELEM_RESTORE_MODEL_CLIP_VOL,
    PELEM_VIEW_IND,
    PELEM_EXEC_STRUCT,
    PELEM_LABEL,
    PELEM_APPL_DATA,
    PELEM_GSE,
    PELEM_PICK_ID,
    PELEM_POLYLINE_SET3_DATA,
    PELEM_FILL_AREA_SET3_DATA,
    PELEM_TRI_STRIP3_DATA,
    PELEM_QUAD_MESH3_DATA,
    PELEM_SET_OF_FILL_AREA_SET3_DATA,
    PELEM_NUNI_BSP_CURVE,
    PELEM_NUNI_BSP_SURF,
    PELEM_CELL_ARRAY3_PLUS,
    PELEM_TEXT_COLR,
    PELEM_MARKER_COLR,
    PELEM_EDGE_COLR,
    PELEM_LINE_COLR,
    PELEM_CURVE_APPROX_CRIT,
    PELEM_LINE_SHAD_METH,
    PELEM_INT_COLR,
    PELEM_BACK_INT_COLR,
    PELEM_BACK_INT_STYLE,
    PELEM_BACK_INT_STYLE_IND,
    PELEM_REFL_PROPS,
    PELEM_BACK_REFL_PROPS,
    PELEM_INT_SHAD_METH,
    PELEM_BACK_INT_SHAD_METH,
    PELEM_INT_REFL_EQN,
    PELEM_BACK_INT_REFL_EQN,
    PELEM_SURF_APPROX_CRIT,
    PELEM_FACE_DISTING_MODE,
    PELEM_FACE_CULL_MODE,
    PELEM_LIGHT_SRC_STATE,
    PELEM_DCUE_IND,
    PEL_COLR_MAPPING_IND,
    PELEM_RENDERING_COLR_MODEL,
    PELEM_NUM_EL_TYPES
} Pelem_type;
```

*excl* A pointer to a *Pelem_type_list* structure containing the list of element types in the exclusion set. Pelem_type_list is defined above. Excluding all element types will result in failure to find matching elements.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*status* A pointer to the location to store the search status. Psearch_status is an enumerated type defined in phigs.h to have the following values:

PSEARCH_STATUS_FAILURE
PSEARCH_STATUS_SUCCESS

*found_el*

A pointer to the location to store the position of the element found. If status is PSEARCH_STATUS_FAILURE, *found_el* is undefined.

**Execution**

If the start element position is less than zero, the search begins at element position zero, whose element type is considered NIL. If the start element position is greater than the number of elements in the structure, the search begins at the last element in the open structure. Otherwise, the search starts at the specified element.

If this element satisfies the search criteria, the element position is returned along with a status indicator of success. Otherwise the next element in the direction specified is considered. The search will continue until either an element fulfills the search criteria or the far end of the structure is reached. If the search is unsuccessful, then a status indicator of failure is returned.

The current element pointer and currently open structure (if any) are not used, and are unaffected.

**ERRORS**

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

201 Ignoring function, the specified structure does not exist

**SEE ALSO**

LABEL (3P)
INQUIRE ELEMENT POINTER (3P)
INCREMENTAL SPATIAL SEARCH (3P)

**NAME**

EMERGENCY CLOSE PHIGS – perform an emergency close of PHIGS

**SYNOPSIS**

**C Syntax**

**void**
**pemergency_close_phigs ()**

**Required PHIGS Operating States**

(*, *, *, *)

**DESCRIPTION**

**Purpose**

EMERGENCY CLOSE PHIGS closes the PHIGS system even if the error state is ON. The purpose of this function is to terminate PHIGS and save as much graphical information as possible, even though the application has encountered errors it can't recover from.

**C Input Parameters**

None

**Execution**

When EMERGENCY CLOSE PHIGS is called, the following actions are performed:

- The open structure (if any) is closed
- Any open archive files are closed
- Any open workstations are updated and closed
- PHIGS is closed

No action is taken if PHIGS is already closed when EMERGENCY CLOSE PHIGS is called.

**ERRORS**

No Error

**SEE ALSO**

CLOSE PHIGS (3P)

**NAME**

EMPTY STRUCTURE – remove all elements from structure

**SYNOPSIS**

**C Syntax**

```
void
pempty_struct ( struct_id )
Pint            struct_id;          structure id
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use EMPTY STRUCTURE to remove all the structure elements from a specified structure. The structure continues to exist as an empty structure even if there are no references to it in other structures. Any references that do exist are not changed by the EMPTY STRUCTURE subroutine.

**C Input Parameter**

*struct_id*

Specifies the structure from which all structure elements should be removed.

**Execution**

The EMPTY STRUCTURE subroutine removes all the structure elements from structure identifier. The structure specified may be the open structure, another existing structure, or a non-existent structure.

If structure identifier is the currently open structure or an existing structure, EMPTY STRUCTURE removes all elements from the structure and sets the element pointer to 0. Any references to this structure in other structures are not changed.

If the structure specified by structure identifier does not exist, EMPTY STRUCTURE creates a new empty structure.

**ERRORS**

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

**SEE ALSO**

DELETE ELEMENT RANGE (3P)
DELETE ELEMENTS BETWEEN LABELS (3P)
DELETE STRUCTURE (3P)

**NAME**

ERROR HANDLING – default PHIGS error handling routine

**SYNOPSIS**

**C Syntax**

```
void
perr_hand ( errnum, funcnum, fname )
Pint        errnum;     error number
Pint        funcnum;    Number of function that detected the error
char        *fname;     name of error file
```

**Required PHIGS Operating States**

(*, *, *, *)

**DESCRIPTION**

**Purpose**

When the error handling mode is ON, ERROR HANDLING is called by PHIGS functions when they detect an error.

An application may replace the default ERROR HANDLING function in order to modify the default response to errors, which is simply to call ERROR LOGGING. Any such replacement must have the name above and accept the same parameters.

**C Input Parameters**

*errnum* The number of the error detected. The error numbers and corresponding error messages are listed with each function description in this reference.

*funcnum*
The identification of the function that detected the error.

*fname* A pointer to a character string containing the name of the error file. The error file is specified in the call to OPEN PHIGS.

**Execution**

ERROR HANDLING is called by a PHIGS function that has deteced an error. The default ERROR HANDLING function performs no other processing than to call ERROR LOGGING.

See OPEN PHIGS for a description of the PHIGS interaction with the error file.

**ERRORS**

No Errors

**SEE ALSO**

ERROR LOGGING (3P)
SET ERROR HANDLING MODE (3P)

**NAME**

ERROR LOGGING – log PEX-SI errors to error file

**SYNOPSIS**

**C Syntax**

```
void
perr_log ( errnum, funcnum, fname )
Pint        errnum;      error number
Pint        funcnum;     Number of function that detected the error
char        *fname;      name of error file
```

**Required PHIGS Operating States**

(*, *, *, *)

**DESCRIPTION**

**Purpose**

ERROR LOGGING writes an error message and the identification of the PHIGS function detecting the error to the *error file*. ERROR LOGGING is always called by the default ERROR HANDLING function and may also be called by the application.

**C Input Parameters**

*errnum* The number of the error detected. The error numbers and corresponding error messages for each function are listed with each function description in this reference.

*funcnum*
The identification of the function which detected the error.

*fname* A pointer to the character string containing the name of the error file specified in the OPEN PHIGS function.

**Execution**

ERROR LOGGING writes the error number, a description of the error, and the function number of the function that detected the error to the error file.

ERROR LOGGING produces error messages of the form:

PEX-SI error <*number*> in <*function name*>: <*error message text*>

**ERRORS**

No Error

**SEE ALSO**

ERROR HANDLING (3P)
SET ERROR HANDLING MODE (3P)
OPEN PHIGS (3P)

**NAME**

ESCAPE – invoke an implementation-dependent function

**SYNOPSIS**

**C Syntax**

```
void
pescape ( func_id, in, store, out )
Pint                func_id;          escape function identifier
Pescape_in_data     *in;              input data for the function
Pstore              store;
Pescape_out_data    **out;            OUT output data of the function
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

ESCAPE is a standard way of invoking PHIGS functions provided by the implementation but not defined in the PHIGS standard. Such functions are called *escape functions.* This single function can access any of the implementation-dependent escape functions provided. These functions either perform an action or return information not defined by the PHIGS standard.

The provided escape functions are:

- Enforce synchronized error reporting
- Inquire a workstation's drawable and display
- Disable bounds checking of Device Coordinate parameters to PHIGS functions

See the separate manual page for each individual escape for a full description of its function.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing data associated with the device state. This buffer is passed as the *store* argument.

The store buffer is a data area managed by PHIGS. While the application is responsible for creating the inital buffer through a call to STORE, PHIGS manages this area such that there is sufficient memory for the specific inquiry. The data record within the store buffer is accessed by the pointer pointed to by *out*.

*func_id* An integer specifying which escape function to perform. The recognized identifiers are:

| | | |
|---|---|---|
| –1 | PUESC_ERRSYNC | *Error Synchronization* |
| –2 | PUESC_DPYINFO | *Inquire x_tool's drawable and display* |
| –3 | PUESC_IGNORE_DC_ERRORS | *Disable DC bounds checking* |

*in_data* A pointer to a Pescape_in_data union containing the input data record required by the escape function specified in the *func_id* parameter. Pescape_in_data is defined in phigs.h as follows:

```
typedef union {
        /* Implementation dependent escape input data records. */
        struct {
                Perrsync        sync_on;
        } escape_in_u1;
        struct {
                Pint            ws_id;
        } escape_in_u2;
        struct {
                Pint            ignore_DC_errors;       /* ignore subsequent DC errors if non-zero */
        } escape_in_u3;
```

```
} Pescape_in_data;
```

**C Output Parameters**

*out_data*

A pointer to a pointer to a Pescape_out_data union. If the specified escape function returns information, PHIGS copies a pointer to the requested information to the variable pointed to by this parameter. This parameter is unused if the escape function does not return any information. Pescape_out_data is defined in phigs.h as follows:

```
typedef union {
        /* Implementation dependent escape input data records. */
        struct {
           Pint             err_ind;          /* error indicator */
           Display          *display;         /* may be returned as NULL */
           XID              drawable_id;
           XID              input_overlay_id; /* will be 0 if category not OUTIN */
           char             *display_name;
        } escape_out_u2;
} Pescape_out_data;
```

**ERRORS**

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

350 Warning, the specified escape is not available on one or more workstations in this implementation. The escape will be processed by those workstations on which it is available

351 Ignoring function, one of the fields within the escape data record is in error

**SEE ALSO**

ESCAPE -1 (3P)
ESCAPE -2 (3P)
ESCAPE -3 (3P)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

ESCAPE -1 – enforce synchronized error reporting

**SYNOPSIS**

**C Syntax**

```
void
pescape ( func_id, in, store, out )
Pint                func_id;          escape function identifier
Pescape_in_data     *in;              input data for the function
Pstore              store;            not used.
Pescape_out_data    **out;            OUT not used
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

ESCAPE -1 enables or disables the enforced synchronization of error reporting.

**C Input Parameters**

*func_id* The function identifier for this escape is PUESC_ERRSYNC, defined in phigs.h to be -1.

*in* A pointer to a Pescape_in_data union, which contains the escape_in_u1 member used by this escape function. This member is defined in phigs.h as:

```
struct {
        Perrsync    sync_on;
} escape_in_u1;
```

*sync_on* One of the following enumerated values:

| | | |
|---|---|---|
| 0 | PERRSYNC_OFF | *Asynchronous Error Reporting* |
| 1 | PERRSYNC_ON | *Synchronous Error Reporting* |

*store* Not used.

**C Output Parameters**

*out* There is no output data record for this escape.

**Execution**

Errors detected by a PHIGS function may not necessarily be reported before that function returns to the application. When this is the case, it is done to enhance performance. When error reporting is delayed, all errors detected will eventually be reported, but during a subsequent call to a PHIGS function. When reported, each error will be correctly associated with the function number of the function that detected the error.

Some applications may need to ensure that errors are reported immediately. This can be indicated by ESCAPE -1. If the specified error synchronization mode is ON, all errors detected by a PHIGS function will be reported before that function returns to the application. If the error synchronization mode is OFF (the default), errors detected by a PHIGS function will be reported either before that function returns or during a subsequent PHIGS function call.

**SEE ALSO**

ERROR HANDLING (3P)

**NAME**

ESCAPE -2 – inquire workstation's display and window information

**SYNOPSIS**

**C Syntax**

**void**
**pescape ( func_id, in, store, out )**

| | | |
|---|---|---|
| **Pint** | **func_id;** | *escape function identifier* |
| **Pescape_in_data** | ***in;** | *input data for the function* |
| **Pstore** | **store;** | *pointer to the buffer* |
| **Pescape_out_data** | ****out;** | *OUT output data of the function* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

ESCAPE -2 returns display and window information for the specified workstation.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing data associated with the device state. This buffer is passed as the *store* argument.

The store buffer is a data area managed by PHIGS. While the application is responsible for creating the inital buffer through a call to STORE, PHIGS manages this area such that there is sufficient memory for the specific inquiry. The data record within the store buffer is accessed by the pointer pointed to by *out*.

*func_id* The function identifier for this escape is PUESC_DPYINFO, which is defined in phigs.h to be -2.

*in* A pointer to a Pescape_in_data union, which contains the escape_in_u2 member used by this escape function. This member is defined in phigs.h as:

```
struct {
        Pint            ws_id;
} escape_in_u2;
```

*ws_id* The indentifier of the workstation for which the display and window information is to be returned.

*store* The memory buffer PHIGS is to use for storing the information returned. This buffer must exist prior to calling this function (see CREATE STORE (3P)).

**C Output Parameters**

*out* A pointer to a pointer to a Pescape_out_data union, which contains the escape_out_u2 member used by this escape function. PHIGS copies a pointer to the requested information to the variable pointed to by this parameter. This member is defined in phigs.h as:

```
struct {
        Pint            err_ind;
        Display         *display;           /* may be returned as NULL */
        XID             drawable_id;
        XID             input_overlay_id;   /* will be 0 if category not OUTIN */
        char            *display_name;
} escape_out_u2;
```

*err_ind* will contain the error number of any error detected by this function.

*display* is a pointer to the Display used by the PHIGS workstation. This pointer may be returned as NULL if PHIGS is not maintaining a connection to the display in the application's process. If *display* is NULL, the *display_name* parameter, also returned by this escape, can be used to identify the server used by the PHIGS workstation.

*drawable_id* is the X window id of the drawable used by the specified workstation for PHIGS output.

*input_overlay_id* is the X window id of the window used by the specified workstation for Locator, Pick, and Stroke input.

*display_name* is the name of the display used by the specified workstation.

**Execution**

The INQUIRE WORKSTATION'S DRAWABLE AND DISPLAY escape returns Display and window information associated with the specified workstation. While intended to be used for x_tool type workstations, where the application doesn't create the window or establish the display connection, this escape will also work with x_drawable workstations.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open.

**SEE ALSO**

ESCAPE (3P)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

ESCAPE -3 – disable or enable PHIGS DC bounds checking

**SYNOPSIS**

**C Syntax**

**void**
**pescape ( func_id, in, store, out )**

| | | |
|---|---|---|
| **Pint** | **func_id;** | *escape function identifier* |
| **Pescape_in_data** | ***in;** | *input data for the function* |
| **Pstore** | **store;** | *not used* |
| **Pescape_out_data** | ****out;** | *OUT not used* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

ESCAPE -3 disables or enables Device Coordinate (DC) bounds checking by PHIGS functions with DC input parameters (such as SET WORKSTATION VIEWPORT and the input device initialization functions).

**C Input Parameters**

*func_id* The function identifier for this escape is PUESC_IGNORE_DC_ERRORS, which is defined in phigs.h to be -3.

*in* A pointer to a Pescape_in_data union, which contains the escape_in_u3 member used by this escape function. This member is defined in phigs.h as:

```
struct {
        Pint    ignore_DC_errors;    /* ignore errors if non-zero */
} escape_in_u3;
```

*ignore_DC_errors* indicates if DC bounds checking is enabled (0, the default) or disabled (non-zero).

*store* Not used.

**C Output Parameters**

*out* There is no output data record for this escape.

**Execution**

This escape disables or enables bounds checking of DC parameters specifed to PHIGS functions. A number of PHIGS workstation functions have DC values as parameters. Normally these values are checked against the current DC size of the workstation when the function is called, and an error is reported if any of the values lie outside the current limits. The function then returns without performing the desired operation. This escape is used to override that behavior by indicating that DC values outside the current workstation limits should be accepted. This allows such DC values to be entered into the workstation state list and used in subsequent PHIGS operations. The use of these values in not defined by the PHIGS standard, but PHIGS will use them as it would values that are within range. This may produce results that do not conform to the PHIGS standard. It is the application's responsibility to ensure that the results of PHIGS' use of these out-of-range values is meaningful.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

**SEE ALSO**

ESCAPE (3P)

**NAME**

ESCAPE -4 – map a X drawable point to a PHIGS pick path

**SYNOPSIS**

**C Syntax**

```
void
pescape ( func_id, in, store, out )
Pint                func_id;          /* escape function identifier */
Pescape_in_data     *in;              /* input data for the function */
Pstore              store;            /* pointer to the buffer */
Pescape_out_data    **out;            /* OUT output data of the function */
```

**Required PHIGS Operating States**

(PHOP, *, WSOP *)

**DESCRIPTION**

**Purpose**

ESCAPE -4 attempts to map a single X drawable coordinate point to a PHIGS pick path using the specified workstation.

ESCAPE -4 provides similar functionality to a PHIGS pick device, but it uses a point provided by the application rather than directly by the operator. This escape is not associated with any PHIGS input device and can be used without initializing any PHIGS input device. It is intended to allow the application to implement its own pick device using the standard X Event mechanism.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing data associated with the device state. This buffer is passed as the *store* argument.

The store buffer is a data area managed by PHIGS. While the application is responsible for creating the inital buffer through a call to STORE, PHIGS manages this area such that there is sufficient memory for the specific inquiry. The data record within the store buffer is accessed by the pointer pointed to by *out*.

*func_id* The function identifier for this escape is PUESC_DRAWABLE_POINT_TO_PICK, which is defined in phigs.h to be -4.

*in* A pointer to a Pescape_in_data union, which contains the *escape_in_u4* member used by this escape function. This member is defined in phigs.h as:

```
struct {
        Pint            ws_id;
        Ppoint          point;          /* a drawable point -- not DC */
        Pfloat          ap_size;        /* aperture size, half-width in DC units */
        Ppath_order     order;
        Pint            depth;          /* depth of path to return */
        Pint            pet;
        Pecho_switch                    echo_switch;
        Plimit3                         echo_volume;
        Pfilter                         filter;/* detectibility filter */
} escape_in_u4;         /* drawable point to pick */
```

*ws_id* Indicates the workstation to use. The structures posted to this workstation and the current workstation state will be used to attempt the mapping.

*point* A point, in X drawable units, that indicates the location of the pick aperture.

*ap_size* The size of the pick aperture, specified as the half-width of the aperture box in DC units.

*order* The order of the returned pick path. The order can be either PORDER_TOP_FIRST, in which case the pick path will be returned in traversal order, or PORDER_BOTTOM_FIRST, in which case the pick path will be returned in the reverse of traversal order. The order and depth

parameters specify which portion of the complete pick path, if any, is returned.

*depth* The portion of the pick path to return. If the order parameter is PORDER_TOP_FIRST then the first "depth" elements will be returned, if the order parameter is PORDER_BOTTOM_FIRST then the first "depth" elements in the inversely ordered pick path will be returned.

*pet* Indicates the echo method to use for picked primitives.

*echo_switch*
Indicates whether or not picked primitives should be echoed.

*echo_volume*
The echo volume.

*filter* The detectibility filter to use for the pick.

*store* This parameter is not used for this escape.

**C Output Parameters**

*out* A pointer to a pointer to a Pescape_out_data union, which contains the *escape_out_u4* member used by this escape function. PHIGS copies a pointer to the pick status and pick path, if any, to this member of the Pescape_out_data union. This member is defined in phigs.h as:

```
struct {
        Pin_status      status;
        Ppick_path      pick;
} escape_out_u4;
```

*status* contains the status of the pick. PIN_STATUS_OK indicates that the pick was successful and a pick path is returned in the *pick* field. PIN_STATUS_NONE indicates that no primitive met the pick criteria. The contents of the *pick* field are undefined in this case.

*pick* contains the pick path if the status is PIN_STATUS_OK.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

254 Ignoring function, invalid echo area/volume; XMIN >= XMAX, YMIN >= YMAX, or ZMIN > ZMAX

255 Ignoring function, one of the echo area/volume boundary points is outside the range of the device

**SEE ALSO**

ESCAPE (3P)
INITIALIZE PICK 3 (3P)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

ESCAPE -5 – map a list of X drawable points to PHIGS World Coordinate points

**SYNOPSIS**

**C Syntax**

**void**
**pescape ( func_id, in, store, out )**

| | | |
|---|---|---|
| **Pint** | **func_id;** | *escape function identifier* |
| **Pescape_in_data** | ***in;** | *input data for the function* |
| **Pstore** | **store;** | *not used* |
| **Pescape_out_data** | ****out;** | *OUT output data of the function* |

**Required PHIGS Operating States**

(PHOP, *, WSOP *)

**DESCRIPTION**

**Purpose**

ESCAPE -5 attempts to map a list of X drawable coordinate points to a list of PHIGS World Coordinate points using the views of the specified workstation. This provides similar functionality to a PHIGS stroke device, but uses points provided by the application rather than directly by the operator. The method used to map the points is the same one used by PHIGS stroke devices.

This escape is not associated with any PHIGS input device and can be used without initializing any PHIGS input device. It is intended to allow the application to implement its own stroke device using the standard X Event mechanism.

**C Input Parameters**

*func_id* The function identifier for this escape is PUESC_DRAWABLE_POINTS_TO_WC, which is defined in phigs.h to be -5.

*in* A pointer to a Pescape_in_data union, which contains the escape_in_u5 member used by this escape function. This member is defined in phigs.h as:

```
struct {
        Pint            ws_id;          /* workstation identifier */
        Ppoint_list3    points;         /* x,y are drawable coords, z is DC*/
} escape_in_u5;         /* drawable points to WC */
```

*ws_id* Indicates the workstation to use. The views on this workstation will be used in the conversion to World Coordinates. mapping.

*points* The list of points to map. The x and y coordinates are drawable coordinate values, as typically provided by X events. The Z coordinate is a PHIGS DC Z value. Ppoint_list3 is defined in phigs.h as follows:

```
struct {
        Pint            num_points;     /* number of Ppoint3s in the list */
        Ppoint3         *points;        /* list of points */
} Ppoint_list3
```

Ppoint3 is defined in phigs.h as follows:

```
struct {
        Pfloat          x;              /* x coordinate */
        Pfloat          y;              /* y coordinate */
        Pfloat          z;              /* z coordinate */
} Ppoint_list3
```

*store* This parameter is not used in this escape function.

**C Output Parameters**

*out* A pointer to a pointer to a Pescape_out_data union, which contains the escape_out_u5 member used by this escape function. PHIGS copies a pointer to the converted points, if any, to this member of the Pescape_out_data union. This member is defined in phigs.h as:

```
struct {
        Pint            view_index;
        Ppoint_list3    points;
} escape_out_u5;
```

*view_index* The index of the highest priority view that contains all the points and is used to transform the points to World Coordinates. The view index is only meaningful if the number of points returned is greater than zero.

*points* The World Coordinate points. The points array in this structure must be allocated prior to calling this function and must be large enough to hold at least the number of points specified in the input parameters. The number of points returned will be 0 if the input points do not all lie within one of the workstation's current view representations.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

**SEE ALSO**

ESCAPE (3P)
INITIALIZE STROKE 3 (3P)

**NAME**

ESCAPE -6 – redraw a list of a PHIGS workstation window regions

**SYNOPSIS**

**C Syntax**

**void**
**pescape ( func_id, in, store, out )**

| | | |
|---|---|---|
| **Pint** | **func_id;** | *escape function identifier* |
| **Pescape_in_data** | ***in;** | *input data for the function* |
| **Pstore** | **store;** | *not used* |
| **Pescape_out_data** | ****out;** | *OUT not used* |

**Required PHIGS Operating States**

(PHOP, *, WSOP *)

**DESCRIPTION**

**Purpose**

ESCAPE -6 redraws the specified list of regions of a PHIGS workstation window without updating the workstation state list. This escape function is intended to provide a means for applications to redraw the workstation window in response to X exposure events, without updating the state of the workstation state list. When this function is called the workstation redraws the specified regions.

**C Input Parameters**

*func_id* The function identifier for this escape is PUESC_REDRAW_REGIONS, which is defined in phigs.h to be -6.

*in* A pointer to a Pescape_in_data union, which contains the escape_in_u6 member used by this escape function. This member is defined in phigs.h as:

```
struct {
        Pint        ws_id;          /* workstation id */
        Pint        num_regions;    /* number of regions */
        XRectangle  *regions;       /* the regions to redraw */
} escape_in_u6;     /* redraw regions */
```

*ws_id* Indicates the workstation to redraw.

*num_regions*
The number of regions specified.

*regions* The regions of the workstation window to redraw.

*store* This parameter is not used in this escape function.

**C Output Parameters**

*out* This parameter is not used by this escape.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

**SEE ALSO**

ESCAPE (3P)

**NAME**

ESCAPE -7 – synchronize with a workstation

**SYNOPSIS**

**C Syntax**

```
void
pescape ( func_id, in, store, out )
Pint                func_id;          escape function identifier
Pescape_in_data     *in;              input data for the function
Pstore              store;            not used
Pescape_out_data    **out;            OUT not used
```

**Required PHIGS Operating States**

(PHOP, *, WSOP *)

**DESCRIPTION**

**Purpose**

ESCAPE -7 synchronizes with a workstation. This function calls XSync() on all connections PHIGS is using to communicate with the server associated with the specified workstation. This has the effect of flushing X connections and waiting for requests sent to the workstation to be processed.

**C Input Parameters**

*func_id* The function identifier for this escape is PUESC_WS_SYNCH, which is defined in phigs.h to be -7.

*in* A pointer to a Pescape_in_data union, which contains the escape_in_u7 member used by this escape function. This member is defined in phigs.h as:

```
struct {
        Pint        ws_id;          /* workstation id */
} escape_in_u7;     /* workstation synch */
```

*ws_id* Indicates the workstation to synchronize with.

*store* This parameter is not used in this escape function.

**C Output Parameters**

*out* This parameter is not used by this escape.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

**SEE ALSO**

**C Output Parameters**

*out* This parameter is not used by this escape.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

**SEE ALSO**

ESCAPE (3P)

**NAME**

EVALUATE VIEW MAPPING MATRIX – generate a transformation matrix to map a 2D VRC window to a 2D NPC viewport

**SYNOPSIS**

**C Syntax**

```
void
peval_view_map_matrix ( map, error_ind, m )
Pview_map       *map;           view mapping
Pint            *error_ind;     OUT error indicator
Pmatrix         m;              OUT view mapping matrix
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use EVALUATE VIEW MAPPING MATRIX to calculate the viewing transformation matrix that transforms a specified window in View Reference Coordinates (VRC) to a specified viewport in Normalized Projection Coordinates (NPC).

The view mapping matrix returned by this function may be used as an argument to the SET VIEW REPRESENTATION function to specify the mapping from VRC to NPC. See the descriptions for the functions SET VIEW REPRESENTATION and EVALUATE VIEW ORIENTATION MATRIX in this function reference for more information.

**C Input Parameters**

*map* A pointer to a Pview_map structure defining the view mapping. This structure is defined in phigs.h as follows:

```
typedef struct {
        Plimit    win;        /* window limits */
        Plimit    proj_vp;    /* viewport limits */
} Pview_map;
```

*win* is a Plimit structure containing *u* and *v* VRC values defining the window to be mapped to NPC. Plimit is defined in phigs.h as follows:

```
typedef struct {
        Pfloat    x_min;    /* x minimum */
        Pfloat    x_max;    /* x maximum */
        Pfloat    y_min;    /* y minimum */
        Pfloat    y_max;    /* y maximum */
} Plimit;
```

The fields in Plimit define the VRC window as follows:

Plimit.x_min = minimum *u* coordinate value
Plimit.x_max = maximum *u* coordinate value
Plimit.y_min = minimum *v* coordinate value
Plimit.y_max = maximum *v* coordinate value

*proj_vp* is a Plimit structure containing the definition of the NPC viewport.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*m* A Pmatrix structure containing the 2D $(3 \times 3)$ transformation matrix that performs the specified view mapping. Pmatrix is defined in phigs.h as follows:

```
typedef Pfloat Pmatrix[3][3];
```

**Execution**

If the input parameters are properly defined, EVALUATE VIEW MAPPING MATRIX returns a 2D $(3 \times 3)$ transformation matrix in the output parameter view mapping matrix. This transformation matrix performs the specified mapping from the VRC window to the NPC viewport.

The following restrictions apply to the VRC window definition:

*u* minimum < *u* maximum
*v* minimum < *v* maximum

The following restrictions apply to the viewport definition:

0 <= *x* minimum < *x* maximum <= 1
0 <= *y* minimum < *y* maximum <= 1

Where zero and 1 are the NPC limits.

The range of VRC coordinate units is determined by the view orientation transformation.

The window to be mapped is specified in VRC by the window input parameter. It is a rectangle located on the $n = 0$ plane whose sides are parallel to the *u* and *v* axes. The rectangle is defined by minimum *u* and *v* values at the lower left hand corner and by the maximum *u* and *v* values at the upper right hand corner.

The viewport to which the window is mapped is specified in NPC by the viewport input parameter. It is a rectangle located in NPC space on the $z = 0$ plane whose sides are parallel to the *x* and *y* axes. The rectangle is defined by the *x* and *y* coordinates of the lower left corner and by the *x* and *y* coordinates of the upper right corner. The coordinate values must be within the closed unit square range of NPC space, [0,1] x [0,1].

**ERRORS**

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

151 Ignoring function, invalid window; XMIN ≥ XMAX or YMIN ≥ YMAX.

152 Ignoring function, invalid viewport; XMIN ≥ XMAX or YMIN ≥ YMAX.

155 Ignoring function, the projection viewport limits are not within NPC range

**NAME**

EVALUATE VIEW MAPPING MATRIX 3 – generate a transformation matrix to map a 3D VRC window to a 3D NPC viewport

**SYNOPSIS**

**C Syntax**

```
void
peval_view_map_matrix3 ( map, error_ind, m )
Pview_map3      *map;           view mapping
Pint            *error_ind;     OUT error indicator
Pmatrix3        m;              OUT view mapping matrix
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use EVALUATE VIEW MAPPING MATRIX 3 to calculate the viewing transformation matrix that transforms a specified 3D window in View Reference Coordinates (VRC) to a specified 3D viewport in Normalized Projection Coordinates (NPC). The viewing transformation may be either a parallel or perspective transformation.

The view mapping matrix returned by this function may be used as an argument to the SET VIEW REPRESENTATION 3 function. See the descriptions of the functions SET VIEW REPRESENTATION 3 and EVALUATE VIEW ORIENTATION MATRIX 3 for more information.

**C Input Parameters**

*map* A pointer to a Pview_map3 structure defining the view mapping. This structure is defined in phigs.h as follows:

```
typedef struct {
        Plimit      win;            /* window limits */
        Plimit3     proj_vp;        /* viewport limits */
        Pproj_type  proj_type;      /* projection type */
        Ppoint3     proj_ref_point; /* projection reference point */
        Pfloat      view_plane;     /* view plane distance */
        Pfloat      back_plane;     /* back plane distance */
        Pfloat      front_plane;    /* front plane distance */
} Pview_map3;
```

*win* is a Plimit structure containing *u* and *v* VRC values defining the window limits. Plimit is defined in phigs.h as follows:

```
typedef struct {
        Pfloat  x_min;  /* x minimum */
        Pfloat  x_max;  /* x maximum */
        Pfloat  y_min;  /* y minimum */
        Pfloat  y_max;  /* y maximum */
} Plimit;
```

The fields in Plimit define the VRC window as follows:

Plimit.x_min = minimum *u* coordinate value
Plimit.x_max = maximum *u* coordinate value
Plimit.y_min = minimum *v* coordinate value
Plimit.y_max = maximum *v* coordinate value

*proj_vp* is a Plimit3 structure containing the NPC viewport limits. *min* and *max* correspond to the back lower left and front upper right coordinates of the viewport volume, respectively. Plimit3 is

defined in phigs.h as follows:

```
typedef struct {
        Pfloat    x_min;    /* x minimum */
        Pfloat    x_max;    /* x maximum */
        Pfloat    y_min;    /* y minimum */
        Pfloat    y_max;    /* y maximum */
        Pfloat    z_min;    /* z minimum */
        Pfloat    z_max;    /* z maximum */
} Plimit3;
```

*proj_type* is an enumerated type that may take the following values:

- PTYPE_PARAL
- PTYPE_PERSPECT

*proj_ref_point* is the Projection Reference Point, defined in VRC. Ppoint3 is defined in phigs.h as follows

```
typedef struct {
        Pfloat    x;    /* x coordinate */
        Pfloat    y;    /* y coordinate */
        Pfloat    z;    /* z coordinate */
} Ppoint3;
```

*view_plane* is the VRC location of the view plane on the *n* axis of the VRC coordinate system.

*back_plane* is the VRC location of the back plane on the *n* axis of the VRC coordinate system.

*front_plane* is the VRC location of the front plane on the *n* axis of the VRC coordinate system.

**C Output Parameters**

*err_ind* A pointer to the location to store the error number of any error detected by this function.

*m* A Pmatrix3 structure containing the 3D $(4 \times 4)$ transformation matrix that performs the specified mapping. Pmatrix3 is defined in phigs.h as follows:

```
typedef Pfloat Pmatrix3[4][4];
```

**Execution**

If the input parameters are properly defined, EVALUATE VIEW MAPPING MATRIX 3 returns a 3D $(4 \times 4)$ transformation matrix in the output parameter view mapping matrix. This transformation matrix performs the specified mapping from the VRC window to the NPC *viewport*.

The *front plane*, *back plane*, and *view plane* all define planes in VRC space parallel to the *uv* plane of the VRC system. The location of front and back along the *n* axis of VRC defines the front and back of the volume of VRC that will be mapped to NPC. The view plane locates the view window on the VRC *n* axis, and window defines the size of the view window by specifying maximum and minimum *u* and *v* values that establish the edges of the window. These values taken together establish the volume of VRC space that is mapped into the NPC viewport.

The type of projection may be parallel or perspective. The *projection reference point* orients the projectors defining the surfaces of the view volume. If the projection type is parallel, the projectors are all parallel to the vector joining the projection reference point and the center of the view window (located on the view plane). If the projection type is perspective, the projectors all converge at the projection reference point. Thus, the view volume is a parallelpiped for parallel views, and a portion of a double rectangular cone for perspective views.

**ERRORS**

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

151 Ignoring function, invalid window; XMIN ≥ XMAX or YMIN ≥ YMAX

152 Ignoring function, invalid viewport; XMIN ≥ XMAX, YMIN ≥ YMAX, or ZMIN > ZMAX

158 Ignoring function, front plane and back plane distances are equal when *z*-extent of the projection viewport is zero

162 Ignoring function, the projection reference point is between the front and back planes

163 Ignoring function, the projection reference point cannot be positioned on the view plane

164 Ignoring function, the back plane is in front of the front plane

155 Ignoring function, the projection viewport limits are not within NPC range

**NAME**

EVALUATE VIEW ORIENTATION MATRIX – calculate a transformation matrix that orients the View Reference Coordinate system in the World Coordinate system

**SYNOPSIS**

**C Syntax**

```
void
peval_view_ori_matrix ( vrp, vup, error_ind, m )
Ppoint      *vrp;          view reference point
Pvec        *vup;          view up vector
Pint        *error_ind;    OUT error indicator
Pmatrix     m;             OUT view orientation matrix
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use EVALUATE VIEW ORIENTATION MATRIX to calculate a *view orientation matrix*, used to transform World Coordinates (WC) to View Reference Coordinates (VRC). This matrix establishes the *u* and *v* VRC axes in relation to WC space. The view orientation matrix calculated by this function can be passed as an argument to SET VIEW REPRESENTATION.

See the descriptions of the functions SET VIEW REPRESENTATION and EVALUATE VIEW MAPPING MATRIX in this function reference for more information.

**C Input Parameters**

*vrp* A pointer to a Ppoint structure containing *x* and *y* WCs that specify the view reference point. Ppoint is defined in phigs.h as follows:

```
typedef struct {
        Pfloat      x;          /* x coordinate */
        Pfloat      y;          /* y coordinate */
} Ppoint;
```

*vup* A pointer to a Pvec structure containing *x* and *y* WC values that specify the view up vector. Pvec is defined in phigs.h as follows:

```
typedef struct {
        Pfloat      delta_x;    /* x magnitude */
        Pfloat      delta_y;    /* y magnitude */
} Pvec;
```

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*m* The 2D homogeneous $(3 \times 3)$ view orientation matrix. Pmatrix is defined in phigs.h as follows:

```
typedef  Pfloat  Pmatrix[3][3];
```

**Execution**

If the input parameters are consistent and well defined, EVALUATE VIEW ORIENTATION MATRIX returns a 2D homogeneous $(3 \times 3)$ transformation matrix in the output parameter view orientation matrix. This matrix transforms WC to VRC according to the values of the input parameters.

The *view reference point* defines the point in WC space that is to be used as the origin of the VRC system. It is specified as a 2D point in the $z = 0$ plane of the WC system and is usually a point on or near the object to be viewed.

The *view up vector* defines the UP direction of the VRC system. It is specified as a 2D vector relative to the view reference point. This vector defines a direction in the WC $z = 0$ plane. This direction becomes the *v* axis of the VRC system. The *n* axis of VRC is parallel to the *z* axis of the WC system, and the *u* axis is determined so that *u*, *v*, and *n* axes form a right-handed coordinate system.

**ERRORS**

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

160 Ignoring function, the view up vector has length zero

**NAME**

EVALUATE VIEW ORIENTATION MATRIX 3 – calculate a transformation matrix that orients the View Reference Coordinate (VRC) system in the World Coordinate (WC) system

**SYNOPSIS**

**C Syntax**

**void**
**peval_view_ori_matrix3 ( vrp, vpn, vup, error_ind, m )**

| | | |
|---|---|---|
| **Ppoint3** | ***vrp;** | *view reference point* |
| **Pvec3** | ***vpn;** | *view plane normal* |
| **Pvec3** | ***vup;** | *view up vector* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pmatrix3** | **m;** | *OUT view orientation matrix* |

**Required PHIGS Operating States**

(PHOP, ∗, ∗, ∗)

**DESCRIPTION**

**Purpose**

Use EVALUATE VIEW ORIENTATION MATRIX 3 to calculate a *view orientation matrix*, used to transform World Coordinates (WC) to View Reference Coordinates (VRC). This matrix establishes the *u*, *v* and *n* VRC axes in relation to World Coordinate Space. The *view orientation matrix* calculated by this function can be passed as an argument to SET VIEW REPRESENTATION 3.

See the descriptions of the functions SET VIEW REPRESENTATION 3 and EVALUATE VIEW MAPPING MATRIX 3 in this function reference for more information.

**C Input Parameters**

*vrp* A pointer to a Ppoint3 structure containing *x*, *y* and *z* world coordinates that specify the view reference point. Ppoint3 is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          x;              /* x coordinate */
        Pfloat          y;              /* y coordinate */
        Pfloat          z;              /* z coordinate */
} Ppoint3;
```

*vpn* A pointer to a Pvec3 structure containing *x*, *y*, and *z* World Coordinate values that specify the view plane normal vector. Pvec3 is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          delta_x;        /* x magnitude */
        Pfloat          delta_y;        /* y magnitude */
        Pfloat          delta_z;        /* z magnitude */
} Pvec3;
```

*vup* A pointer to a Pvec3 structure containing *x*, *y*, and *z* world coordinate values that specify the view up vector. Pvec3 is defined above.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*m* The 3D homogeneous $(4 \times 4)$ view orientation matrix. Pmatrix3 is defined in phigs.h as follows:

```
typedef  Pfloat  Pmatrix3[4][4];
```

**Execution**

If the input parameters are consistent and well-defined, EVALUATE VIEW ORIENTATION MATRIX 3 returns a 3D homogeneous $(4 \times 4)$ transformation matrix in the output parameter *view orientation matrix*. This matrix transforms World Coordinates to View Reference Coordinates according to the values of the input parameters.

The *view reference point* defines the point in World Coordinate Space that is to be used as the origin of the View Reference Coordinate system. It is specified as a 3D world coordinate point and is usually a point on or near the object to be viewed.

The *view plane normal* is a 3D vector defined in World Coordinates relative to the view reference point. It defines the *n* axis of the VRC system. The plane in WC that contains the view reference point and is perpendicular to the view plane normal is called the *view reference plane*.

The *view up vector* is a 3D vector defined in World Coordinates relative to the view reference point. The projection of this vector onto the view reference plane parallel to the *view plane normal* determines the *v* axis of the VRC system.

The *u* axis of VRC is determined so that the *uvn* axes form a right-handed coordinate system.

**ERRORS**

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

159 Ignoring function, the view plane normal vector has length zero

160 Ignoring function, the view up vector has length zero

161 Ignoring function, the view up and view plane normal vectors are parallel thus the viewing coordinate system cannot be established

**NAME**

EXECUTE STRUCTURE – create a structure element to invoke another structure

**SYNOPSIS**

**C Syntax**

```
void
pexec_struct ( struct_id )
Pint            struct_id;          structure identifier
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

EXECUTE STRUCTURE puts a structure element containing the *structure identifier* into the currently open structure according to the current edit mode. If no structure exists with the *structure identifier*, an empty structure is created with that identifier. EXECUTE STRUCTURE elements are used to define hierarchical structure networks by invoking one structure from within another during structure traversal.

If the current edit mode is INSERT, the EXECUTE STRUCTURE element is inserted into the open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, the EXECUTE STRUCTURE element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new EXECUTE STRUCTURE element.

**C Input Parameter**

*struct_id*

Identifies structure to be invoked from the currently open structure.

**Execution**

When an EXECUTE STRUCTURE element is traversed, the traversal performs the following in order:

1. Suspends the traversal of the current structure.
2. Saves the current states of:
   all primitive attributes,
   local and global modelling transformations,
   the view index,
   HLHSR and PICK identifiers,
   the name set, and
   modelling clipping limits and clipping indicator.
3. Sets the global modelling transformation to the current composite modelling transformation. Sets the local modelling transformation to an identity matrix.
4. Traverses the structure specified by *structure identifier*. This process will be repeated for any EXECUTE STRUCTURE elements encountered in the structure with the *structure identifier* and its subordinate structures before returning to the current structure.
5. Restores all the saved states.
6. Resumes traversal of the current structure.

If *structure identifier* itself contains an EXECUTE STRUCTURE element, traversal of the structure with the *structure identifier* is also suspended, and traversal continues with that structure invocation. At the end of a structure (i.e., after its last element), traversal returns to the invoking structure, restoring the saved state, and continuing with later elements in the invoking structure, until its end. This continues until the entire structure network is traversed (i.e., until traversal of the posted structure is completed).

The application must avoid recursive structure networks: no EXECUTE STRUCTURE element may invoke any ancestor structure. The PHIGS implementation is not required to check for this problem, and PEX-SI does not.

Each subordinate structure inherits the current attribute values from its parent structure, except for the modelling transformations, as described in step 3 above. Unless the attributes are explicitly changed with the appropriate SET element, they retain the last value set in a structure higher in the structure network hierarchy. On the other hand, when traversal of a structure is suspended, the current attribute values are saved and later restored when traversal of the structure resumes. This means that structures can only affect the display of structures subordinate to them in the network; no structure can affect any attributes of its parent.

Implementations vary in the depth of structure execution supported, and in the efficiency of EXECUTE STRUCTURE traversal. PEX-SI is limited only by available virtual memory, and EXECUTE STRUCTURE is quite efficient.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

POST STRUCTURE (3P)
DELETE STRUCTURE NETWORK (3P)
CHANGE STRUCTURE REFERENCES (3P)
CHANGE STRUCTURE IDENTIFIER AND REFERENCES (3P)
INQUIRE PATHS TO ANCESTORS (3P)
INQUIRE PATHS TO DESCENDENTS (3P)

**NAME**

FILL AREA – create a structure element specifying a 2D fill area primitive

**SYNOPSIS**

**C Syntax**

```
void
pfill_area ( point_list )
Ppoint_list        *point_list;        array of points
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

The FILL AREA function puts a structure element containing the 2D specification of a FILL AREA primitive into the currently open structure. The FILL AREA primitive is a closed polygonal area defined by a series of two dimensional points in Modelling Coordinates. The *z* coordinate is assumed to be 0.

**Note:** The FILL AREA primitive INTERIOR attributes control the representation of the interior of the area defined by the specified coordinate points. FILL AREA does not have separate edge attributes. See the FILL AREA SET and FILL AREA SET 3 primitives for independent control of the edge properties.

If the current edit mode is INSERT, the structure element created by the FILL AREA subroutine is inserted into the open structure after the element pointed to by the structure's element pointer. If the edit mode is REPLACE, the FILL AREA element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new FILL AREA element.

**C Input Parameters**

*point_list*

A pointer to a list *num_points* long of Ppoint structures containing the *x* and *y* coordinates for each point used to define the FILL AREA polygon. The Ppoint_list structure is defined in phigs.h as follows:

```
typedef struct  {
        Pint    num_points;   /* number of Ppoints in the list */
        Ppoint  *points;      /* list of points */
} Ppoint_list;
```

Ppoint is defined as follows:

```
typedef struct {
        Pfloat          x;              /* x coordinate */
        Pfloat          y;              /* y coordinate */
} Ppoint;
```

**Execution**

When the structure is traversed, the FILL AREA element draws a closed polygonal area. The appearance of the interior of the defined area is determined by the attributes listed below. These attributes control the type of fill (SOLID, EMPTY, HOLLOW, HATCH, and PATTERN) and the colour with which the primitive is drawn.

The points are specified in Modelling Coordinates. These may be any coordinate units that are convenient to the application. At traversal, these coordinate values are transformed by the current Local and Global Modelling Transformations, the View Representation selected by the *current view index*, and the Workstation Transformation current on the workstation to which the structure is posted.

**Attributes Applied**

The attributes listed below are used to display the FILL AREA primitive when the structure is traversed. The Aspect Source Flags (ASFs) tell where to access the output display attributes. These attributes can come directly from the traversal state list, or they can be accessed indirectly, using the appropriate index in the traversal state list and the corresponding bundled representation in the PHIGS workstation state list.

| | |
|---|---|
| interior colour | interior colour index ASF |
| back interior colour | back interior colour ASF |
| interior style | interior style ASF |
| back interior style | back interior style ASF |
| interior style index | interior style index ASF |
| back interior style index | back interior style index ASF |
| interior shading method | interior shading method ASF |
| back interior shading method | back interior shading method ASF |
| interior reflectance equation | interior reflectance equation ASF |
| back interior reflectance equation | back interior reflectance equation ASF |
| area properties | area properties ASF |
| back area properties | back area properties ASF |
| interior index | |
| face distinguishing mode | |
| face culling mode | |
| depth cue index | |
| light source state | |
| name set | |

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

FILL AREA 3 (3P)
FILL AREA SET (3P)
FILL AREA SET 3 WITH DATA (3P+)

**NAME**

FILL AREA 3 – create a structure element specifying a 3D fill area primitive

**SYNOPSIS**

**C Syntax**

```
void
pfill_area3 ( point_list )
Ppoint_list3        *point_list;        array of points
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

The FILL AREA 3 function puts a structure element containing the 3D specification of a FILL AREA 3 primitive into the currently open structure. The FILL AREA 3 primitive is a closed polygonal area defined by a series of three dimensional Modelling Coordinate points.

**Note:** The FILL AREA 3 primitive INTERIOR attributes control the representation of the interior of the area defined by the specified coordinate points. FILL AREA 3 does not have separate edge attributes. See the FILL AREA SET and FILL AREA SET 3 primitives for independent control of the edge properties.

If the current edit mode is INSERT, the structure element created by the FILL AREA 3 subroutine is inserted into the open structure after the element pointed to by the structure's element pointer. If the edit mode is REPLACE, the FILL AREA 3 element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new FILL AREA 3 element.

**C Input Parameters**

*point_list*

A pointer to a list *num_points* long of Ppoint3 structures containing the *x*, *y*, and *z* coordinates for each point used to define the FILL AREA 3 polygon. The Ppoint_list3 structure is defined in phigs.h as follows:

```
typedef struct  {
        Pint        num_points;    /* number of Ppoint3 structures in the list */
        Ppoint3     *points;       /* list of points */
} Ppoint_list3;
```

Ppoint3 is defined as follows:

```
typedef struct  {
        Pfloat    x;    /* x coordinate */
        Pfloat    y;    /* y coordinate */
        Pfloat    z;    /* z coordinate */
} Ppoint3;
```

**Execution**

When the structure is traversed, the FILL AREA 3 element draws a closed polygonal area. The appearance of the interior of the defined area is determined by the attributes listed below. These attributes control the type of fill (SOLID, EMPTY, HOLLOW, HATCH, and PATTERN) and the colour in which the primitive is drawn.

The coordinate points are specified in Modelling Coordinate points. Modelling Coordinate units may be any that are convenient to the application. At traversal, these coordinate values are transformed by the current Local and Global Modelling Transformations, the View Representation selected by the current view index, and the Workstation Transformation current on the workstation to which the structure is posted.

**Attributes Applied**

The attributes listed below are used to display the FILL AREA 3 primitive when the structure is traversed. The Aspect Source Flags (ASFs) tell where to access the output display attributes. These attributes can come directly from the traversal state list, or they can be accessed indirectly, using the appropriate index in the

traversal state list and the corresponding bundled representation in the PHIGS workstation state list.

| | |
|---|---|
| interior colour | interior colour index ASF |
| back interior colour | back interior colour ASF |
| interior style | interior style ASF |
| back interior style | back interior style ASF |
| interior style index | interior style index ASF |
| back interior style index | back interior style index ASF |
| interior shading method | interior shading method ASF |
| back interior shading method | back interior shading method ASF |
| interior reflectance equation | interior reflectance equation ASF |
| back interior reflectance equation | back interior reflectance equation ASF |
| area properties | area properties ASF |
| back area properties | back area properties ASF |
| interior index | |
| face distinguishing mode | |
| face culling mode | |
| depth cue index | |
| light source state | |
| name set | |

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

FILL AREA (3P)
FILL AREA SET 3 (3P)
FILL AREA SET 3 WITH DATA (3P+)

**NAME**

FILL AREA SET – create a structure element specifying a 2D fill area set primitive

**SYNOPSIS**

**C Syntax**

```
void
pfill_area_set ( point_list_list )
Ppoint_list_list    *point_list_list;    list of point lists
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

The FILL AREA SET function puts a structure element containing the 2D specification of a FILL AREA SET primitive into the currently open structure. The FILL AREA SET primitive is a group of implicitly closed polygonal areas. This allows for specifying areas with holes or disjoint regions. The subroutine parameters specify the number of fill areas to be drawn, the number of points used to define each, and the Modelling coordinates of each point. The *z* coordinates are assumed to be zeroes. Each set of points defines a separate closed area. All of the areas specified in a single function call are drawn using the current values of the FILL AREA SET attributes listed below.

If the current edit mode is INSERT, the structure element created by the FILL AREA SET subroutine is inserted into the open structure after the element pointed to by the structure's element pointer. If the edit mode is REPLACE, the new FILL AREA SET element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new FILL AREA SET element.

**C Input Parameters**

*point_list_list*

A pointer to a list of Ppoint_list_list structures. Each Ppoint_list_list structure has 2 fields, the *num_points_list* field, which is the number of 2d point lists, and the *point_lists*, a pointer to a list of PPoint_list. Each Ppoint_list structure defines a fill area. Ppoint_list_list is defined in phigs.h as follows:

```
typedef struct {
        Pint        num_point_lists;      /* number of point lists  */
        Ppoint_list *point_lists;         /* list of point lists */
} Ppoint_list_list;
```

Ppoint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint        num_points;       /* number of Ppoint structures in the list */
        Ppoint      *points;          /* list of points */
} Ppoint_list;
```

The *num_points* component specifies the number of points used to define the fill area. The *points* component is a pointer to a list, *num_points* long, of Ppoint structures which contain the Modelling Coordinates of each vertex of the fill area. Ppoint is defined in phigs.h as follows:

```
typedef struct {
        Pfloat    x;    /* x coordinate */
        Pfloat    y;    /* y coordinate */
} Ppoint;
```

**Execution**

When the structure is traversed, the FILL AREA SET element draws the specified number of closed polygonal areas using the given points. Each fill area forms a boundary by connecting the specified points in order. Bounds will implicitly be closed by extending the boundary of the fill area from the last point to the first.

The points are specified in Modelling Coordinates. These may be any coordinate units that are convenient to the application. At traversal, these coordinates are transformed by the current Local and global modelling transformations, the view representation selected by the current view index, and the workstation transformation current on the workstation to which the structure is posted.

All the specified areas are drawn using the current values of the primitive attributes listed below. The edge attributes, EDGE FLAG, EDGETYPE, EDGEWIDTH SCALE FACTOR, and EDGE COLOUR INDEX allow you to control whether, and how, the edges of the areas are displayed independently of the interior representation.

**Using Edge Attributes with Interior Style**

If edges are not displayed, and the INTERIOR STYLE is HOLLOW, only the boundaries of the fill areas are drawn using the interior colour. If edges are displayed when the INTERIOR STYLE is HOLLOW, the edges will overlay the boundaries using the edge colour. If the EDGETYPE is a broken line, the boundaries will show through the breaks in the edge representation.

**Attributes Applied**

The attributes listed below are used to display the FILL AREA SET primitive when the structure is traversed. The Aspect Source Flags (ASFs) tell where to access the output display attributes. These attributes can come directly from the traversal state list, or they can be accessed indirectly, using the appropriate index in the traversal state list and the corresponding bundled representation in the PHIGS workstation state list.

| | |
|---|---|
| interior colour | interior colour index ASF |
| back interior colour | back interior colour ASF |
| interior style | interior style ASF |
| back interior style | back interior style ASF |
| interior style index | interior style index ASF |
| back interior style index | back interior style index ASF |
| interior shading method | interior shading method ASF |
| back interior shading method | back interior shading method ASF |
| interior reflectance equation | interior reflectance equation ASF |
| back interior reflectance equation | back interior reflectance equation ASF |
| area properties | area properties ASF |
| back area properties | back area properties ASF |
| interior index | |
| edge colour | edge colour index ASF |
| edge flag | edge flag ASF |
| edgetype | edgetype ASF |
| edgewidth scale factor | edgewidth scale factor ASF |
| edge index | |
| face distinguishing mode | |
| face culling mode | |
| depth cue index | |
| light source state | |
| name set | |

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

FILL AREA (3P)
FILL AREA SET 3 (3P)
FILL AREA SET 3 WITH DATA (3P+)

**NAME**

FILL AREA SET 3 – create a structure element specifying a 3D fill area set primitive

**SYNOPSIS**

**C Syntax**

```
void
pfill_area_set3 (point_list_list)
Ppoint_list_list3    *point_list_list;        list of point lists
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

The FILL AREA SET 3 function puts a structure element containing the 3D specification of a FILL AREA SET 3 primitive into the currently open structure. The FILL AREA SET 3 primitive is a group of implicitly closed polygonal areas. This allows for specifying areas with holes or disjoint coplanar regions. The subroutine parameters specify the number of areas to be drawn, the number of points used to define each area, and the coordinates of each point. Each set of points defines a separate closed area. All of the areas specified in a single function call are drawn using the current values of the FILL AREA SET 3 attributes listed below.

If the current edit mode is INSERT, the FILL AREA SET 3 element is inserted into the open structure after the element pointed to by the structure's element pointer. If the edit mode is REPLACE, the FILL AREA SET 3 element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new FILL AREA SET 3 element.

**C Input Parameters**

*point_list_list*

A pointer to a list of Ppoint_list_list3 structures. Each Ppoint_list_list3 structure has 2 fields, the *num_points_list* field, which is the number of 3d point lists, and the *point_lists*, a pointer to a list of PPoint_list3. Each Ppoint_list3 structure defines a fill area. Ppoint_list_list3 is defined in phigs.h as follows:

```
typedef struct {
        Pint        num_point_lists;      /* number of point lists  */
        Ppoint_list3 *point_lists;        /* list of point lists    */
} Ppoint_list_list3;
```

Ppoint_list3 is defined in phigs.h as follows:

```
typedef struct {
        Pint        num_points;      /* number of Ppoint3s in the list */
        Ppoint3     *points;         /* list of points    */
} Ppoint_list3;
```

The *num_points* component specifies the number of points used to define the fill area. The *points* component is a pointer to a list, *num_points* long, of Ppoint3 structures containing the coordinates of the fill area vertices in Modelling Coordinates. Ppoint3 is defined in phigs.h as follows:

```
typedef struct  {
        Pfloat    x;    /* x coordinate */
        Pfloat    y;    /* y coordinate */
        Pfloat    z;    /* z coordinate */
} Ppoint3;
```

**Execution**

When the structure is traversed, the FILL AREA SET 3 element draws the specified number of closed polygonal areas using the given points. Each fill area forms a boundary by connecting the specified points in order. Bounds will implicitly be closed by extending the boundary of the fill area from the last point to the

first.

The entire fill area set must be coplanar. Non-coplanar bounds are treated in a workstation-dependent fashion.

The points are specified in Modelling Coordinates. These may be any coordinate units that are convenient to the application. At traversal, these coordinates are transformed by the current Local and global modelling transformations, the view representation selected by the current view index, and the workstation transformation current on the workstation to which the structure is posted.

All the specified areas are drawn using the current values of the primitive attributes listed below. The edge attributes, EDGE FLAG, EDGETYPE, EDGEWIDTH SCALE FACTOR, and EDGE COLOUR INDEX, allow you to control whether, and how, the edges of the areas are displayed independently of the interior representation.

**Using Edge Attributes with Interior Style**

If edges are not displayed, and the INTERIOR STYLE is HOLLOW, only the boundaries of the fill areas are drawn using the interior colour. If the edges are displayed when the INTERIOR STYLE is HOLLOW, the edges will overlay the boundaries using the edge colour. If the EDGETYPE is a broken line, the boundaries will show through the breaks in the edge representation.

**Attributes Applied**

The attributes listed below are used to display the FILL AREA SET 3 primitive when the structure is traversed. The Aspect Source Flags (ASFs) tell where to access the output display attributes. These attributes can come directly from the traversal state list, or they can be accessed indirectly, using the appropriate index in the traversal state list and the corresponding bundled representation in the PHIGS workstation state list.

| | |
|---|---|
| interior colour | interior colour index ASF |
| back interior colour | back interior colour ASF |
| interior style | interior style ASF |
| back interior style | back interior style ASF |
| interior style index | interior style index ASF |
| back interior style index | back interior style index ASF |
| interior shading method | interior shading method ASF |
| back interior shading method | back interior shading method ASF |
| interior reflectance equation | interior reflectance equation ASF |
| back interior reflectance equation | back interior reflectance equation ASF |
| area properties | area properties ASF |
| back area properties | back area properties ASF |
| interior index | |
| edge colour | edge colour index ASF |
| edge flag | edge flag ASF |
| edgetype | edgetype ASF |
| edgewidth scale factor | edgewidth scale factor ASF |
| edge index | |
| face distinguishing mode | |
| face culling mode | |
| depth cue index | |
| light source state | |
| name set | |

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

FILL AREA 3 (3P)
FILL AREA SET (3P)
FILL AREA SET 3 WITH DATA (3P+)

**NAME**

FILL AREA SET 3 WITH DATA – creates a 3D fill area set structure element that includes colour and shading data

**SYNOPSIS**

**C Syntax**

**void**
**pfill_area_set3_data ( fflag, eflag, vflag, colour_model, fdata, nfa, edata, vdata )**

| | | |
|---|---|---|
| **Pint** | **fflag;** | *what data is specified per facet* |
| **Pint** | **eflag;** | *what data is specified per edge* |
| **Pint** | **vflag;** | *what data is specified per vertex* |
| **Pint** | **colour_model;** | *colour model* |
| **Pfacet_data3** | ***fdata;** | *facet data* |
| **Pint** | **nfa;** | *number of fill areas in the set* |
| **Pedge_data_list** | ***edata;** | *edge data list* |
| **Pfacet_vdata_list3** | ***vdata;** | *facet vertex data list* |

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

FILL AREA SET 3 WITH DATA creates a 3D fill area set primitive with colour and shading data.

The FILL AREA SET 3 WITH DATA is a set of closed polygonal areas defined by a series of three dimensional Modeling Coordinate points. You can specify a geometric normal for the entire fill area, and specify colour/normal information for each vertex *in addition* to point coordinates. The added data is used when applying lighting, shading, face distinguishing, and face culling.

**C Input Parameters**

*fflag* The data per facet flag specifies the available data for the entire fill area set.

| | | |
|---|---|---|
| 0 | PFACET_NONE | *No Facet Data Specified* |
| 1 | PFACET_COLOUR | *Facet Colours Specified* |
| 2 | PFACET_NORMAL | *Facet Normal Specified* |
| 3 | PFACET_COLOUR_NORMAL | *Facet Normal and Colours Specified* |

*eflag* The data per edge flag specifies the available information for each edge.

| | | |
|---|---|---|
| 0 | PEDGE_NONE | *No Edge Data Specified* |
| 1 | PEDGE_VISIBILITY | *Edge Visibility Flags Specified* |

*vflag* The data per vertex flag specifies the available data for each vertex of the primitive.

| | | |
|---|---|---|
| 0 | PVERT_COORD | *Coordinates Specified* |
| 1 | PVERT_COORD_COLOUR | *Coordinates and Vertex Colour Specified* |
| 2 | PVERT_COORD_NORMAL | *Coordinates and Vertex Normal Specified* |
| 3 | PVERT_COORD_COLOUR_NORMAL | *Coordinates, Vertex Colour, and Vertex Normal Specified* |

*colour_model*

The colour type for specified facet and/or vertex colours.

| | | |
|---|---|---|
| 0 | PINDIRECT | *Colour Index Specified* |
| 1 | PMODEL_RGB | *Red, Green, and Blue* |
| 2 | PMODEL_CIELUV | *CIE Colour Model* |
| 3 | PMODEL_HSV | *Hue, Saturation, and Value* |
| 4 | PMODEL_HLS | *Hue, Lightness, and Saturation* |

*fdata* A pointer to a Pfacet_data3 structure that specifies the fill area and optionally associated colour information. Pfacet_data3 is defined as follows:

```
typedef union {
        Pcoval          colr;           /* colour */
        Pvec3           normal;         /* unit normal*/
        Pconorm3        conorm;         /* colour and normal */
} Pfacet_data3;
```

Pcoval is defined as:

```
typedef union {
        Pint            ind;            /* index in workstation colour bundle table */
        Pcolr_rep       direct;         /* direct colour components */
} Pcoval;
```

Pcolr_rep is defined as:

```
typedef union {
        Prgb            rgb;            /* Red Green Blue colour specification */
        Pcieluv         cieluv;         /* CIE L*U*V* colour specification */
        Phls            hls;            /* Hue Lightness Saturation colour specification */
        Phsv            hsv;            /* Hue Saturation Value colour specification */
        Pdata           unsupp;         /* Colour in unsupported colour model */
} Pcolr_rep;
```

Prgb is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          red;            /* red, hue, etc */
        Pfloat          green;          /* green, saturation, lightness, etc */
        Pfloat          blue;           /* blue, value, saturation, etc */
} Prgb;
```

Pcieluv is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          cieluv_x;       /* x coefficient */
        Pfloat          cieluv_y;       /* y coefficient */
        Pfloat          cieluv_y_lum;   /* y luminance */
} Pcieluv;
```

Phsv is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          hue;            /* hue */
        Pfloat          satur;          /* saturation */
        Pfloat          value;          /* value */
} Phsv;
```

Phls is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          hue;            /* hue */
        Pfloat          lightness;      /* lightness */
        Pfloat          satur;          /* saturation */
} Phls;
```

Pdata is defined in phigs.h as follows:

```
typedef struct {
        size_t          size;           /* size of data */
        char            *data           /* pointer to data */
} Pdata;
```

Pvec3 is defined in phigs.h as follows:

```
typedef struct  {
        Pfloat          delta_x;        /* x magnitude */
        Pfloat          delta_y;        /* y magnitude */
        Pfloat          delta_z;        /* z magnitude */
} Pvec3;
```

Pconorm3 is defined in phigs.h as follows:

```
typedef struct  {
        Pcoval          colr;           /* colour */
        Pvec3           norm;           /* normal */
} Pconorm3;
```

Pcoval and Pvec3 are defined above.

*nfa* The number of fill areas in the set.

*edata* A pointer to an array (nfa) of Pedge_data_list structures that specifies the edge data. Pedge_data_list is defined as:

```
typedef struct {
        Pint            num_edges;      /* number of edges */
        Pedge_data_arr  edgedata;       /* edge data */
} Pedge_data_list;
```

Pedge_data_arr is defined as:

```
typedef union {
        Pedge_flag      *edges;         /* array of edge flags */
        /* implementation dependent data */
} Pedge_data_arr;
```

Pedge_flag is defined as:

```
typedef enum {
        PEDGE_OFF,
        PEDGE_ON
} Pedge_flag;
```

*vdata* A pointer to an array (nfa) of Pfacet_vdata_list3 structures that specifies each fill area in the set and optionally associated colour information. Pfacet_vdata_list3 is defined as:

```
typedef struct {
        Pint                num_vertices;   /* number of vertices */
        Pfacet_vdata_arr3   vertex_data;    /* facet vertex data */
} Pfacet_vdata_list3;
```

Pfacet_vdata_arr3 is defined as:

```
typedef union {
        Ppoint3         *points;        /* array of points */
        Pptco3          *ptcolrs;       /* array of points and colours */
        Pptnorm3        *ptnorms;       /* array of points and normals */
        Pptconorm3      *ptconorms;     /* array of points, colours, and normals */
        /* implementation dependent types can go here */
} Pfacet_vdata_arr3;
```

Ppoint3 is defined as:

```
typedef struct {
        Pfloat          x;              /* x coordinate */
        Pfloat          y;              /* y coordinate */
        Pfloat          z;              /* z coordinate */
} Ppoint3;
```

Pptco3 is defined as:

```
typedef struct {
        Ppoint3         point;          /* point coordinates */
        Pcoval          colr;           /* colour */
} Pptco3;
```

Ppoint3 anc Pcoval are defined above. Pptnorm3 is defined in phigs.h as follows:

```
typedef struct {
        Ppoint3         point;          /* point coordinates */
        Pvec3           norm;           /* normal */
} Pptnorm3;
```

Ppoint3 and Pvec3 are defined above. Pptconorm3 is defined in phigs.h as follows:

```
typedef struct {
        Ppoint3         point;          /* point coordinates */
        Pcoval          colr;           /* colour */
        Pvec3           norm;           /* normal */
} Pptconorm3;                           /* point with colour and normal */
```

Ppoint3, Pcoval, and Pvec3 are defined above.

**Execution**

Depending on the edit mode, a FILL AREA SET 3 WITH DATA element is inserted into the open structure after the element pointer, or replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the newly created FILL AREA SET 3 WITH DATA element.

When the structure is traversed, FILL AREA SET 3 WITH DATA draws an implicitly closed polygonal area with multiple boundaries. PHIGS implicitly closes the area by extending each fill area boundary from the last point to the first point. Fill areas can be empty, hollow, shaded, or filled with a color, pattern, or hatch. The interior of the primitive is defined by the odd winding rule. The FILL AREA SET 3 WITH DATA element can specify the facet color and normal vector as well as colours and normal vectors for each vertex. These colours and normal vectors are used in conjunction with the current lighting and depth cueing attributes to colour and shade the primitive. Note that unexpected lighting effects will result if the supplied normals are not of unit length.

The application should insure that the fill area set points are coplanar. Display of non-coplanar fill area set is device dependent. FILL AREA SET 3 WITH DATA elements containing fewer than three vertices can be created, but they are treated in a device dependent manner during structure traversal.

**Attributes Applied**

The attributes listed below are used to display the FILL AREA SET 3 WITH DATA primitive when the structure is traversed. The Aspect Source Flags (ASFs) tell where to access the output display attributes. These attributes can come directly from the traversal state list, or they can be accessed indirectly, using the appropriate index in the traversal state list and the corresponding bundled representation in the PHIGS workstation state list.

| | |
|---|---|
| interior colour | interior colour index ASF |
| back interior colour | back interior colour ASF |
| interior style | interior style ASF |
| back interior style | back interior style ASF |
| interior style index | interior style index ASF |
| back interior style index | back interior style index ASF |
| interior shading method | interior shading method ASF |
| back interior shading method | back interior shading method ASF |
| interior reflectance equation | interior reflectance equation ASF |
| back interior reflectance equation | back interior reflectance equation ASF |
| area properties | area properties ASF |
| back area properties | back area properties ASF |
| interior index | |
| edge colour | edge colour index ASF |
| edge flag | edge flag ASF |
| edgetype | edgetype ASF |
| edgewidth scale factor | edgewidth scale factor ASF |
| edge index | |
| face distinguishing mode | |
| face culling mode | |
| depth cue index | |
| light source state | |
| name set | |

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

FILL AREA SET 3 (3P)

**NAME**

FLUSH DEVICE EVENTS – remove all entries generated by a specified input device from the input event queue

**SYNOPSIS**

**C Syntax**

**void**
**pflush_events ( ws, class, dev )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pin_class** | **class;** | *device class* |
| **Pint** | **dev;** | *logical input device number* |

**Required PHIGS Operating States**

(PHOP, WSOP, ∗, ∗)

**DESCRIPTION**

**Purpose**

Use FLUSH DEVICE EVENTS to remove all the events in the input queue that were generated by a specified input device.

**C Input Parameters**

*ws* The workstation identifier of the workstation associated with the input device.

*class* The input class of the input device. Pin_class is an enumerated type defined in phigs.h that may take the following values:

- PIN_NONE
- PIN_LOC
- PIN_STROKE
- PIN_VAL
- PIN_CHOICE
- PIN_PICK
- PIN_STRING

*dev* The device number of the input device.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, ∗, ∗)

054 Ignoring function, the specified workstation is not open

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

256 Warning, the input queue has overflowed

**SEE ALSO**

AWAIT EVENT (3P)

**NAME**

GENERALIZED DRAWING PRIMITIVE – create 2D GDP elements

**SYNOPSIS**

**C Syntax**

```
void
pgdp ( point_list, gdp_id, gdp_data )
Ppoint_list     *point_list;        array of points
Pint            gdp_id;             gdp function identifier
Pgdp_data       *gdp_data;          data record pointer
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

GENERALIZED DRAWING PRIMITIVE creates an implementation dependent drawing primitive. There are no GDPs implemented in the PEX-SI graphics library.

**C Input Parameters**

*point_list*

A pointer to a structure containing a list of *x* and *y* values in Modelling Coordinates (MC). Ppoint_list is defined in phigs.h as:

```
typedef struct {
        Pint    num_points;     /* number of Ppoints in the list */
        Ppoint  *points;        /* list of points */
} Ppoint;
```

Ppoint is defined in phigs.h as follows:

```
typedef struct {
        Pfloat  x;      /* x coordinate */
        Pfloat  y;      /* y coordinate */
} Ppoint;
```

*gdp_id* The identifier of the GENERALIZED DRAWING PRIMITIVE to insert. There are no predefined GDPs.

*gdp_data*

A pointer to a Pgdp_data union containing the information needed to perform the function specified by *gdp_id*. Pgdp_data is defined in phigs.h as:

```
typdef union {
        struct {
                Pint            unused;
        }gdp_r1;
        Pdata   unsupp;         /* unsupported GDP data record */
} Pgdp_data;                    /* implementation dependent */
```

Pdata is defined in phigs.h as:

```
typdef struct {
        size_t  size;           /* size of data */
        char    *data;          /* pointer to data */
} Pdata;
```

**Execution**

If the current edit mode is INSERT, the structure element created by the GENERALIZED DRAWING PRIMITIVE function is inserted into the open structure after the element pointed to by the element pointer. If the current edit mode is REPLACE, the GENERALIZED DRAWING PRIMITIVE element replaces the element pointed to by

the element pointer. In either case, the element pointer is updated to point to the new structure element.

Support for GDPs is implementation and workstation dependent.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**NAME**

GENERALIZED DRAWING PRIMITIVE 3 – create 3D GDP elements

**SYNOPSIS**

**C Syntax**

```
void
pgdp3 ( point_list, gdp3_id, gdp_data )
Ppoint_list3    *point_list;        array of points
Pint            gdp3_id;            gdp function identifier
Pgdp_data3      *gdp_data;          gdp data record
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

GENERALIZED DRAWING PRIMITIVE 3 creates an implementation dependent drawing primitive. There are no GDPs implemented in the PEX-SI graphics library.

**C Input Parameters**

*point_list*

A pointer to a structure containing a list of *x* and *y* values in Modelling Coordinates (MC). Ppoint_list3 is defined in phigs.h as:

```
typedef struct {
        Pint    num_points;     /* number of Ppoints in the list */
        Ppoint3 *points;        /* list of points */
} Ppoint_list3;
```

Ppoint3 is defined in phigs.h as follows:

```
typedef struct {
        Pfloat  x;      /* x coordinate */
        Pfloat  y;      /* y coordinate */
        Pfloat  z;      /* z coordinate */
} Ppoint3;
```

*gdp3_id* The identifier of the generalized drawing primitive to insert. There are no predefined GDPs.

*gdp_data*

A pointer to a Pgdp_data3 union containing the information needed to perform the function specified by *gdp3_id*. Pgdp_data3 is defined in phigs.h as:

```
typdef union {
        struct {
                Pint            unused;
        }gdp3_r1;
        Pdata   unsupp;         /* unsupported GDP data record */
} Pgdp_data3;                   /* implementation dependent */
```

Pdata is defined in phigs.h as:

```
typdef struct {
        size_t  size;           /* size of data */
        char    *data;          /* pointer to data */
} Pdata;
```

**Execution**

If the current edit mode is INSERT, then GENERALIZED DRAWING PRIMITIVE 3 is inserted into the currently open structure after the element currently pointed to by the element pointer. If the edit mode is REPLACE, then GENERALIZED DRAWING PRIMITIVE 3 replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**NAME**

GENERALIZED STRUCTURE ELEMENT – create a generalized structure element

**SYNOPSIS**

**C Syntax**

```
void
pgse ( id, gse )
Pint            id;             gse identifier
Pgse_data       *gse;           gse data record
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

GENERALIZED STRUCTURE ELEMENT creates an implementation dependent structure element. There are no GSEs implemented in the PEX-SI graphics library.

**C Input Parameters**

*id* The identifier of the generalized structure element to insert.

*gse* A pointer to a Pgse_data union containing the information needed to perform the function specified by *id*. Pgse_data is defined in phigs.h as:

```
typedef union {
        struct {
                Pint            unused;
        } gse_r1;
        Pdata   unsupp;     /* unsupported GSE data record */
} Pgse_data;
```

**Execution**

If the current edit mode is INSERT, then GENERALIZED STRUCTURE ELEMENT is inserted into the currently open structure after the element currently pointed to by the element pointer. If the edit mode is REPLACE, then GENERALIZED STRUCTURE ELEMENT replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**NAME**

GET CHOICE – retrieve the CHOICE event in the PHIGS current event report

**SYNOPSIS**

**C Syntax**

```
void
pget_choice ( in_status, choice )
Pin_status      *in_status;     OUT choice status
Pint            *choice;        OUT choice
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

GET CHOICE retrieves the current event report from the PHIGS state list when the event report contains a Choice event. A CHOICE event consists of a status and a choice value. The status indicates if the operator selected one of the available choices from the input device. The choice value indicates the choice selected, if any.

**C Output Parameters**

*in_status*

PHIGS sets the variable pointed to by *in_status* to indicate whether or not one of the possible choices of the Choice device was selected. A value of PIN_STATUS_OK indicates that one of the possible choice values was selected. A value of PIN_STATUS_NONE indicates that none of the possible choices was selected. Pin_status is defined in phigs.h as follows:

```
typedef enum {
        PIN_STATUS_NONE,
        PIN_STATUS_OK,
        PIN_STATUS_NO_IN
} Pin_status;
```

The value PIN_STATUS_NO_IN will not be returned by this function. That value is for use with REQUEST CHOICE.

*choice* If the status returned is PIN_STATUS_OK, PHIGS sets the variable pointed to by *choice* to the choice value selected by the operator. The variable is not set if the status returned is PIN_STATUS_NONE.

**Execution**

When an input device that is set to Event mode is triggered by the operator, an event report is added to the input event queue. The event report contains the identity and current measure of the device. AWAIT EVENT moves the measure of the oldest event in the input queue to the current event report in the PHIGS state list. If the event is a Choice event, GET CHOICE is used to retrieve the contents of the event from the current event report.

If the choice event was generated by the operator's selecting one of the possible choice values, GET CHOICE returns the status PIN_STATUS_OK and the choice value selected. If none of the possible values was selected, the status returned will be PIN_STATUS_NONE and no choice value is returned.

See INITIALIZE CHOICE 3 for a description of the available Choice devices and how their measure values are determined.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

259 Ignoring function, the input device class of the current input report does not match the class being requested

**SEE ALSO**

INITIALIZE CHOICE (3P)
INITIALIZE CHOICE 3 (3P)
AWAIT EVENT (3P)

**NAME**

GET LOCATOR – retrieve the 2D components of the LOCATOR event in the PHIGS current event report

**SYNOPSIS**

**C Syntax**

```
void
pget_loc ( view_ind, loc_pos )
Pint        *view_ind;      OUT view index
Ppoint      *loc_pos;       OUT locator position
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

GET LOCATOR retrieves the current event report from the PHIGS state list when the event report contains a Locator event. A LOCATOR event consists of a position and a view index. The position is the World Coordinate (WC) point corresponding to the position on the workstation selected by the operator. The view index is the index of the view representation used to transform the selected position from Normalized Projection Coordinates (NPC) to WC.

**C Output Parameters**

*view_ind*

PHIGS sets the variable pointed to by *view_ind* to the index of the view representation used to transform the NPC locator position to a WC position.

*loc_pos* PHIGS sets the variable pointed to by *loc_pos* to the 2D world coordinate locator position corresponding to the device coordinate position selected by the operator. Ppoint is defined in phigs.h as follows:

```
typedef struct {
        Pfloat    x;    /* x coordinate */
        Pfloat    y;    /* y coordinate */
} Ppoint;
```

**Execution**

When an input device in Event mode is triggered by the operator, an event report is added to the input event queue. The event report contains the identity and current measure of the device. AWAIT EVENT moves the measure of the oldest event in the input queue to the current event report in the PHIGS state list. If the event is a Locator event, GET LOCATOR or GET LOCATOR 3 is used to retrieve the contents of the event from the current event report.

The LOCATOR position is computed by mapping the DC workstation position selected by the operator to a WC point. The workstation transform is used to transform the DC position to a NPC position prior to transforming it to WC. See INITIALIZE LOCATOR and SET VIEW TRANSFORMATION INPUT PRIORITY for a description of how PHIGS determines the view representation to use to map the NPC position to WC.

See INITIALIZE LOCATOR for a description of the available LOCATOR devices and how their measure values are determined.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

259 Ignoring function, the input device class of the current input report does not match the class being requested

**SEE ALSO**

INITIALIZE LOCATOR (3P)
AWAIT EVENT (3P)
GET LOCATOR 3 (3P)

SET VIEW TRANSFORMATION INPUT PRIORITY (3P)

**NAME**

GET LOCATOR 3 – retrieve the LOCATOR event in the PHIGS current event report.

**SYNOPSIS**

**C Syntax**

```
void
pget_loc3 ( view_ind, loc_pos )
Pint        *view_ind;      OUT view index
Ppoint3     *loc_pos;       OUT locator position
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

GET LOCATOR 3 retrieves the current event report from the PHIGS state list when the event report contains a Locator event. A LOCATOR event consists of a position and a view index. The position is the World Coordinate (WC) point corresponding to the position on the workstation selected by the operator. The view index is the index of the view representation used to transform the selected position from Normalized Projection Coordinates (NPC) to WC.

**C Output Parameters**

*view_ind*

PHIGS sets the variable pointed to by *view_ind* to the index of the view representation used to transform the NPC locator position to a WC position.

*loc_pos* PHIGS sets the variable pointed to by *loc_pos* to the 3D world coordinate locator position corresponding to the device coordinate position selected by the operator. Ppoint3 is defined in phigs.h as follows:

```
typedef struct {
        Pfloat    x;    /* x coordinate */
        Pfloat    y;    /* y coordinate */
        Pfloat    z;    /* z coordinate */
} Ppoint3;
```

**Execution**

When an input device in Event mode is triggered by the operator, an event report is added to the input event queue. The event report contains the identity and current measure of the device. AWAIT EVENT moves the measure of the oldest event in the input queue to the current event report in the PHIGS state list. If the event is a Locator event, GET LOCATOR or GET LOCATOR 3 is used to retrieve the contents of the event from the current event report.

The LOCATOR position is computed by mapping the DC workstation position selected by the operator to a WC point. The workstation transform is used to transform the DC position to a NPC position prior to transforming it to WC. See INITIALIZE LOCATOR and SET VIEW TRANSFORMATION INPUT PRIORITY for a description of how PHIGS determines the view representation to use to map the NPC position to WC.

See INITIALIZE LOCATOR 3 for a description of the available LOCATOR devices and how their measure values are determined.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

259 Ignoring function, the input device class of the current input report does not match the class being requested

**SEE ALSO**

INITIALIZE LOCATOR 3 (3P)
AWAIT EVENT (3P)

GET LOCATOR (3P)
SET VIEW TRANSFORMATION INPUT PRIORITY (3P)

**NAME**

GET PICK – retrieve the PICK event in the PHIGS current event report

**SYNOPSIS**

**C Syntax**

```
void
pget_pick ( depth, in_status, rpick )
Pint            depth;          depth of pick path to return
Pin_status      *in_status;     OUT pick status
Ppick_path      *rpick;         OUT pick path
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

GET PICK retrieves the current event report from the PHIGS state list when the event report contains a PICK event. A PICK event consists of a status and a pick path. The status indicates whether or not a pick by the operator successfully selected an ouput primitive. The pick path describes the location of the picked primitive, if any, in the Central Structure Store (CSS).

**C Input Parameter**

*depth* The maximum number of pick path elements to return. This may be more or less than the actual path depth in the current event report.

**C Output Parameters**

*in_status*

PHIGS sets the variable pointed to by *in_status* to the event's pick status. A value of PIN_STATUS_OK indicates that an output primitive was successfully selected by the operator. A value of PIN_STATUS_NONE indicates that a pick was attempted but no primitive was selected. Pin_status is defined in phigs.h as follows:

```
typedef enum {
        PIN_STATUS_NONE,
        PIN_STATUS_OK,
        PIN_STATUS_NO_IN
} Pin_status;
```

The value PIN_STATUS_NO_IN will not be returned by this function. That value is for use with REQUEST PICK.

*rpick* PHIGS sets the variable pointed to by *rpick* to the events's pick path. Ppick_path is defined in phigs.h as follows:

```
typedef struct {
        Pint                depth;      /* pick path_list depth */
        Ppick_path_elem     *path_list; /* pick path */
} Ppick_path;
```

A pick path is only returned if the status is PIN_STATUS_OK. The contents of *rpick* are not changed if the status is not PIN_STATUS_OK. The pick path depth indicates the number of elements in the event's pick path as stored in the current event report. It is not affected by the *depth* input parameter; therefore, the number of elements returned in *path_list* may be less than depth.

*path_list* is an array of references defining the location of the picked primitive in the CSS. **This array must be allocated by the calling program and the array pointer assigned to this field before calling this function.** The array must be at least long enough to hold the number of path elements indicated by the *depth* input parameter. Ppick_path_elem is defined in phigs.h as:

```
typedef struct {
        Pint    struct_id;    /* structure identifier */
        Pint    pick_id;      /* hierarchical pick identifier */
        Pint    elem_pos;     /* element sequence number */
} Ppick_path_elem;
```

The *struct_id, pick_id,* and *elem_pos* are the structure identifier, pick id, and element number, respectively, of each element in the path. Each element but the last indicates the location and current pick id of an EXECUTE STRUCTURE structure element in the path to the selected primitive. The last element in the path indicates the location and current pick id of the selected output primitive.

**Execution**

GET PICK retrieves the contents of a PICK event from the current event report in the PHIGS state list. If PIN_STATUS_OK is returned as the status, the returned pick path contains the portion of the event's pick path requested, as specified by the *depth* input parameter. The pick path will be in either top-first or bottom-first order depending upon the value specified when the device was initialized.

When an input device in EVENT mode is triggered by the operator, an event report is added to the input event queue if the queue is not full. The event report contains the identity and current measure of the triggered device. AWAIT EVENT moves the measure of the oldest event in the input queue to the current event report in the PHIGS state list. If the event was a PICK event, GET PICK is used to retrieve the contents of the event from the current event report.

If the pick event was generated by the operator's successfully selecting an output primitive, GET PICK returns the status PIN_STATUS_OK and the pick path to the primitive selected. If a primitive was not successfully selected, the status returned will be PIN_STATUS_NONE and no pick path is returned.

See INITIALIZE PICK 3 for a description of the available PICK devices and how their measure values are determined.

The *pick filter* of a PICK device controls which output primitives on the device's workstation are pickable. By default no output primitives are pickable. See SET PICK FILTER for more information on the pick filter.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

259 Ignoring function, the input device class of the current input report does not match the class being requested

**SEE ALSO**

INITIALIZE PICK (3P)
INITIALIZE PICK 3 (3P)
AWAIT EVENT (3P)
SET PICK FILTER (3P)

**NAME**

GET STRING– retrieve the STRING event in the PHIGS current event report

**SYNOPSIS**

**C Syntax**

```
void
pget_string ( string )
char            *string;        OUT pointer to string of bufsiz
                                for the device + 1 for the terminator
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

GET STRING retrieves the current event report from the PHIGS state list when the event report contains a STRING event. A STRING event consists of an array of characters reflecting the contents of a STRING input device's buffer when the event was generated.

**C Output Parameter**

*string* PHIGS copies the event's contents to the array pointed to by *string*. The string is null terminated. The array must be at least as large as the buffer of the STRING device that generated the event. This buffer size is set when the device is initialized.

**Execution**

GET STRING retrieves the contents of a STRING device from the current event report in the PHIGS state list.

When an input device in EVENT mode is triggered by the operator, an event report is added to the input event queue if the queue is not full. The event report contains the identity and current measure of the triggered device. AWAIT EVENT moves the contents of the oldest event in the input queue to the current event report in the PHIGS state list. If the event was a STRING event, GET STRING is used to retrieve the contents of the event from the current event report.

See INITIALIZE STRING 3 for a description of the available STRING devices and how events are generated from them.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

259 Ignoring function, the input device class of the current input report does not match the class being requested

**SEE ALSO**

INITIALIZE STRING (3P)
INITIALIZE STRING 3 (3P)
AWAIT EVENT (3P)

**NAME**

GET STROKE – retrieve the 2D components of the STROKE event in the PHIGS current event report

**SYNOPSIS**

**C Syntax**

```
void
pget_stroke ( view_ind, stroke )
Pint            *view_ind;          OUT view index
Ppoint_list     *stroke;            OUT stroke
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

GET STROKE retrieves the current event report from the PHIGS state list when the event report contains a STROKE event. Only the 2D components of the event are returned.

A STROKE event consists of a list of World Coordinate (WC) points and a view_index. The points correspond to positions on the workstation selected by the operator. The view_index is the index of the view used to transform those positions to World Coordinates.

**C Output Parameters**

*view_ind*

PHIGS sets the variable pointed to by *view_ind* to the index of the view representation used to transform the NPC stroke positions to WC points.

*stroke* PHIGS copies the number and list of stroke points selected by the operator to the variable pointed to by *stroke*. Ppoint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint    num_points;     /* Number of points in the list */
        Ppoint  *points;        /* List of points */
} Ppoint_list;
```

*num_points* is the number of points in *points*.

*points* is the array of 2D WC points. **This array must be allocated by the calling program and the array pointer assigned to this field before calling this function.** The array must be at least as large as the buffer of the STROKE device that generated the event. This buffer size is set when the device is initialized. Ppoint is defined in phigs.h as follows:

```
typedef struct {
        Pfloat  x;      /* x coordinate */
        Pfloat  y;      /* y coordinate */
} Ppoint;
```

**Execution**

GET STROKE retrieves the 2D components of the STROKE event from the current event report in the PHIGS state list. The *x* and *y* components are returned. The *z* component is discarded.

When an input device in EVENT mode is triggered by the operator an event report is added to the input event queue. The event report contains the identity and current measure of the triggering device. AWAIT EVENT moves the measure of the oldest event in the input queue to the current event report in the PHIGS state list. If the event was a STROKE event, GET STROKE 3 or GET STROKE is used to retrieve the 2D components of the event from the current event report. The STROKE positions are computed by mapping the DC workstation positions selected by the operator to WC points. The workstation transform is used to transform the DC positions to NPC positions prior to transforming them to WC. See INITIALIZE LOCATOR and SET VIEW TRANSFORMATION INPUT PRIORITY for a description of how PHIGS determines the view representation to use

to map the NPC positions to WC.

See INITIALIZE STROKE for a description of the available STROKE devices and how their measure values are determined.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

259 Ignoring function, the input device class of the current input report does not match the class being requested

**SEE ALSO**

INITIALIZE STROKE (3P)
AWAIT EVENT (3P)
SET VIEW TRANSFORMATION INPUT PRIORITY (3P)
GET STROKE 3 (3P)

**NAME**

GET STROKE 3 – retrieve the STROKE event in the PHIGS current event report

**SYNOPSIS**

**C Syntax**

```
void
pget_stroke3 ( view_ind, stroke )
Pint            *view_ind;          OUT view index
Ppoint_list3    *stroke;            OUT stroke
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

GET STROKE 3 retrieves the current event report from the PHIGS state list when the event report contains a STROKE event.

A STROKE event consists of a list of World Coordinate (WC) points and a view_index. The points correspond to positions on the workstation selected by the operator. The view_index is the index of the view used to transform those positions to World Coordinates.

**C Output Parameters**

*view_ind*

PHIGS sets the variable pointed to by *view_ind* to the index of the view representation used to transform the NPC stroke positions to WC points.

*stroke* PHIGS copies the number and list of stroke points selected by the operator to the variable pointed to by *stroke*. Ppoint_list3 is defined in phigs.h as follows:

```
typedef struct {
        Pint        num_points;     /* number of Ppoint3s in the list */
        Ppoint3     *points;        /* list of points */
} Ppoint_list3;
```

*num_points* is the number of points in *points*.

*points* is the array of Ppoint3 structures specifying the points in WC. **This array must be allocated by the calling program and the array pointer assigned to this field before calling this function.** The array must be at least as large as the buffer of the STROKE device that generated the event. This buffer size is set when the device is initialized. Ppoint3 is defined in phigs.h as follows:

```
typedef struct {
        Pfloat    x;    /* x coordinate */
        Pfloat    y;    /* y coordinate */
        Pfloat    z;    /* z coordinate */
} Ppoint3;
```

**Execution**

GET STROKE 3 retrieves the STROKE event from the current event report in the PHIGS state list.

When an input device in EVENT mode is triggered by the operator, an event report is added to the input event queue. The event report contains the identity and current measure of the triggering device. AWAIT EVENT moves the measure of the oldest event in the input queue to the current event report in the PHIGS state list. If the event was a STROKE event, GET STROKE 3 or GET STROKE is used to retrieve the event from the current event report. The STROKE positions are computed by mapping the DC workstation positions selected by the operator to WC points. The workstation transform is used to transform the DC positions to NPC positions prior to transforming them to WC. See INITIALIZE LOCATOR 3 and SET VIEW TRANSFORMATION

INPUT PRIORITY for a description of how PHIGS determines the view representation to use to map the NPC positions to WC.

See INITIALIZE STROKE 3 for a description of the available STROKE devices and how their measure values are determined.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

259 Ignoring function, the input device class of the current input report does not match the class being requested

**SEE ALSO**

INITIALIZE STROKE 3 (3P)
AWAIT EVENT (3P)
SET VIEW TRANSFORMATION INPUT PRIORITY (3P)
GET STROKE (3P)

**NAME**

GET VALUATOR – retrieve the VALUATOR event in the PHIGS current event report

**SYNOPSIS**

**C Syntax**

```
void
pget_val ( valuator )
Pfloat          *valuator;          OUT valuator value
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

GET VALUATOR retrieves the current event report from the PHIGS state list when the event report contains a VALUATOR event.

A VALUATOR event consists of a floating point number that the operator has selected.

**C Output Parameter**

*valuator*

PHIGS copies the event's contents to the variable pointed to by *valuator*.

**Execution**

GET VALUATOR retrieves the contents of a VALUATOR event from the current event report in the PHIGS state list.

When an input device that is set to EVENT mode is triggered by the operator, an event report is added to the input event queue. The event report contains the identity and current measure of the triggered device. AWAIT EVENT moves the measure of the oldest event in the input queue to the current event report in the PHIGS state list. If the event was a VALUATOR event, GET VALUATOR is used to retrieve the contents of the event from the current event report.

See INITIALIZE VALUATOR 3 for a description of the available VALUATOR devices and how their measure values are determined.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

259 Ignoring function, the input device class of the current input report does not match the class being requested

**SEE ALSO**

INITIALIZE VALUATOR (3P)
INITIALIZE VALUATOR 3 (3P)
AWAIT EVENT (3P)

**NAME**

INCREMENTAL SPATIAL SEARCH – search the structure network for the next occurrence of a graphical output structure element meeting the specified search criteria

**SYNOPSIS**

**C Syntax**

**void**
**pincr_spa_search ( ref, dist, sp, mclip_flag, ceil, norm, inv, len, st, error_ind, fp, tot_len )**

| | | |
|---|---|---|
| **Ppoint** | ***ref;** | *search reference point* |
| **Pfloat** | **dist;** | *search distance* |
| **Pelem_ref_list** | ***sp;** | *starting path list* |
| **Pclip_ind** | **mclip_flag;** | *model clip flag* |
| **Pint** | **ceil;** | *search ceiling index* |
| **Pfilter_list** | ***norm;** | *normal filter list* |
| **Pfilter_list** | ***inv;** | *inverted filter list* |
| **Pint** | **len;** | *length of application list* |
| **Pint** | **st;** | *starting position* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pelem_ref_list** | ***fp;** | *OUT found path* |
| **Pint** | ***tot_len;** | *OUT length of list in PHIGS* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

INCREMENTAL SPATIAL SEARCH searches a structure network for the next occurrence of a graphical output structure element that satisfies the specified search criteria.

**C Input Parameters**

*ref* A pointer to a Ppoint structure that specifies the *x* and *y* coordinates, in World Coordinates (WC), of the search reference point. The *z* coordinate is assumed to be zero. The Ppoint structure is defined in phigs.h as:

```
typedef struct {
        Pfloat    x;    /* x coordinate */
        Pfloat    y;    /* y coordinate */
} Ppoint;
```

*dist* A real value specifying the maximum distance that a selected primitive may be from the search reference point.

*sp* A pointer to a Pelem_ref_list structure that contains the starting search path. Pelem_ref_list is defined in phigs.h as:

```
typedef struct {
        Pint          num_elem_refs;    /* number of execute references */
        Pelem_ref     *elem_refs;       /* list of execute references */
} Pelem_ref_list;
```

*elem_refs* is a pointer to a list of *num_elem_refs* long of Pelem_ref structures containing the structure identifier and element number of each execute reference structure element in the execute reference list.

Pelem_ref is defined in phigs.h as:

```
typedef struct {
        Pint    struct_id;    /* structure identifier */
        Pint    elem_pos;     /* element number */
```

```
} Pelem_ref;
```

*mclip_flag*

Indicates whether model clipping should be performed during the incremental spatial search. Pclip_ind is defined in phigs.h as follows:

```
typedef enum {
        PIND_NO_CLIP, Do not perform model clipping
        PIND_CLIP       Perform model clipping
} Pclip_ind;
```

*ceil* The search ceiling index defines a position in the starting path list. The structure identifier at this position of the list defines a ceiling for the search.

*norm* A pointer to a Pfilter_list structure containing a set of normal filter lists. Pfilter_list is defined below.

*inv* A pointer to a Pfilter_list structure containing a set of inverted filter lists. Pfilter_list is defined in phigs.h as:

```
typedef struct {
        Pint      num_filters;    /* number of filters */
        Pfilter   *filters;       /* list of filters */
} Pfilter_list;
```

*filters* is a pointer to an array (num_filters) of Pfilter structures, that contains an inclusion set and an exclusion set of names.

Pfilter is defined in phigs.h as:

```
typedef struct {
        Pint_list   incl_set;   /* inclusion set */
        Pint_list   excl_set;   /* exclusion set */
} Pfilter;
```

The incl_set contains a set of names for the inclusion set. The excl_set contains a set of names for the exclusion set. Pint_list is defined in phigs.h as:

```
typedef struct {
        Pint    num_ints;   /* number of Pints in list */
        Pint    *ints;      /* list of integers */
} Pint_list;
```

*len* The maximum length of the portion of the found path to be returned.

*st* The starting position of the portion of the found path to be returned. The complete found path is returned if *st* is set to zero.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*fp* PHIGS stores the found path in the variable pointed to by this parameter. Pelem_ref_list is defined in phigs.h as:

```
typedef struct{
        Pint            num_elem_refs;  /* number of execute references */
        Pelem_ref       *elem_refs;     /* list of execute references */
} Pelem_ref_list;
```

*num_elem_refs* specifies the length of the structure path returned.

*elem_refs* is a pointer to an array of Pelem_ref structures fInum_elem_refs *long. All but the last of these structures contain the structure identifier and element number of each execute reference structure element in the path to the selected output primitive.*

The application must allocate memory for *len* elements in the list of *elem_refs*. If the length of the found path is greater than the *len* parameter, then no found path is returned. In this case, *tot_len* will be set to indicate the total length of the found path.

*tot_len* A pointer to an integer to contain the total length of the found path.

**Execution**

When INCREMENTAL SPATIAL SEARCH is called, a search begins at the element following the position specified by the starting path and continues until an ouput primitive meeting the search criteria is found or the search ceiling is reached. The search is conceptually a traversal with structure elements being examined sequentially and matched against the search criteria. Element position zero is allowed so that the search may start at the first element of the structure.

Search filters are applied to control which output primitive structure elements are considered. The filters are organized into two lists, the normal and the inverted filter lists, which operate in the opposite senses. A structure element is said to be *accepted* if it is declared eligible when NAME SET is applied to the filters. To be accepted by a filter, NAME SET must have at least one name in common with the inclusion set and no names in common with the exclusion set. For a structure element to be considered a candidate for the search it must be accepted by each of the filters in the normal filter list and rejected by each of the filters in the inverted filter list. If the normal filter list is empty, then all structure elements satisfy the acceptance criteria for normal filters. If the inverted filter list is empty, all structure elements satisfy the rejection criteria for inverted filters. Graphical output structure elements which satisfy the search filters are checked for proximity to the reference point.

For TEXT structure elements, the only proximity relationship used is closeness to the text origin. For ANNOTATION TEXT RELATIVE structure elements, the only proximity relationship used is closeness to the annotation reference point.

The search will continue until either a graphical output structure element matches the search criteria or the end of the structure identified by the search ceiling is reached. If a search is successful, the complete search path is returned as a found path. An unsuccessful search returns a null found path.

The function is incremental in that, having found a match, the search may be continued by invoking INCREMENTAL SPATIAL SEARCH again with the found path as the next starting path. This allows all elements matching the search criteria to be found for a given portion of a structure network.

**ERRORS**

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

203 Ignoring function, specified starting path not found in CSS

204 Ignoring function, specified search ceiling index out of range

**SEE ALSO**

ADD NAMES TO SET (3P)
REMOVE NAMES FROM SET (3P)
INCREMENTAL SPATIAL SEARCH 3 (3P)

**NAME**

INCREMENTAL SPATIAL SEARCH 3 – search the structure network for the next occurrence of a graphical output structure element meeting the specified search criteria

**SYNOPSIS**

**C Syntax**

**void**
**pincr_spa_search3 ( ref, dist, sp, mclip_flag, ceil, norm, inv, len, st, error_ind, fp, tot_len )**

| | | |
|---|---|---|
| **Ppoint3** | ***ref;** | *search reference point* |
| **Pfloat** | **dist;** | *search distance* |
| **Pelem_ref_list** | ***sp;** | *starting path list* |
| **Pclip_ind** | **mclip_flag;** | *model clip flag* |
| **Pint** | **ceil;** | *search ceiling index* |
| **Pfilter_list** | ***norm;** | *normal filter list* |
| **Pfilter_list** | ***inv;** | *inverted filter list* |
| **Pint** | **len;** | *length of application list* |
| **Pint** | **st;** | *starting position* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pelem_ref_list** | ***fp;** | *OUT found path* |
| **Pint** | ***tot_len;** | *OUT length of list in PHIGS* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

INCREMENTAL SPATIAL SEARCH 3 searches a structure network for the next occurrence of a graphical output structure element that satisfies the specified search criteria.

**C Input Parameters**

*ref* A pointer to a Ppoint structure that specifies the *x*, *y*, and *z* coordinates, in World Coordinates (WC), of the search reference point. The Ppoint3 structure is defined in phigs.h as:

```
typedef struct {
        Pfloat    x;    /* x coordinate */
        Pfloat    y;    /* y coordinate */
        Pfloat    z;    /* z coordinate */
} Ppoint3;
```

*dist* A real value specifying the maximum distance that a selected primitive may be from the search reference point.

*sp* A pointer to a Pelem_ref_list structure that contains the starting search path. Pelem_ref_list is defined in phigs.h as:

```
typedef struct {
        Pint          num_elem_refs;    /* number of element references */
        Pelem_ref     *elem_refs;       /* list of element references */
} Pelem_ref_list;
```

*elem_refs* is a pointer to a list of *num_elem_refs* long of Pelem_ref structures containing the structure identifier and element number of each execute reference structure element in the execute reference list.

Pelem_ref is defined in phigs.h as:

```
typedef struct {
        Pint    struct_id;    /* structure identifier */
        Pint    elem_pos;     /* element number */
```

```
} Pelem_ref;
```

*mclip_flag*
Indicates whether model clipping should be performed during the incremental spatial search. Pclip_ind is defined in phigs.h as follows:

```
typedef enum {
        PIND_NO_CLIP, Do not perform model clipping
        PIND_CLIP       Perform model clipping
} Pclip_ind;
```

*ceil* The search ceiling index defines a position in the starting path list. The structure identifier at this position of the list defines a ceiling for the search.

*norm* A pointer to a Pfilter_list structure containing a set of normal filter lists. Pfilter_list is defined below.

*inv* A pointer to a Pfilter_list structure containing a set of inverted filter lists. Pfilter_list is defined in phigs.h as:

```
typedef struct {
        Pint      num_filters;    /* number of filters */
        Pfilter   *filters;       /* list of filters */
} Pfilter_list;
```

*filters* is a pointer to an array (num_filters) of Pfilter structures, that contains an inclusion set and an exclusion set of names.

Pfilter is defined in phigs.h as:

```
typedef struct {
        Pint_list   incl_set;   /* inclusion set */
        Pint_list   excl_set;   /* exclusion set */
} Pfilter;
```

The incl_set contains a set of names for the inclusion set. The excl_set contains a set of names for the exclusion set. Pint_list is defined in phigs.h as:

```
typedef struct {
        Pint    num_ints;   /* number of Pints in list */
        Pint    *ints;      /* list of integers */
} Pint_list;
```

*len* The maximum length of the portion of the found path to be returned.

*st* The starting position of the portion of the found path to be returned. If *st* is set to zero, then the complete found path is returned.

**C Output Parameters**

*error_ind*
A pointer to the location to store the error number of any error detected by this function.

*fp* A pointer to a Pelem_ref_list structure containing the found path. Pelem_ref_list is defined in phigs.h as:

```
typedef struct{
        Pint        num_elem_refs;  /* number of element references */
        Pelem_ref   *elem_refs;     /* list of element references */
} Pelem_ref_list;
```

*num_elem_refs* specifies the length of the structure path returned.

*elem_refs* is a pointer to an array of Pelem_ref structures *num_elem_refs* long. All but the last of these structures contain the structure identifier and element number of each execute reference structure element in the path to the selected output primitive.

The application must allocate memory for *len* elements in the list of *elem_refs*. If the length of the found path is greater than the *len* parameter, then no found path is returned. In this case, *tot_len* will be set to indicate the total length of the found path.

*tot_len* A pointer to an integer which returns the total length of the found path.

**Execution**

When INCREMENTAL SPATIAL SEARCH 3 is called, a search begins at the element following the position specified by the starting path and continues until an ouput primitive meeting the search criteria is found or the search ceiling is reached. The search is conceptually a traversal with structure elements being examined sequentially and matched against the search criteria. Element position zero is permitted so that the search may start at the first element of the structure.

Search filters are applied to control which output primitives structure elements in the structures searched are considered. The filters are organized into two lists, the normal and the inverted filter lists which operate in the opposite senses. A structure element is said to be *accepted* if it is declared eligible when NAME SET is applied to the filters. To be accepted by a filter, NAME SET must have at least one name in common with the inclusion set and no names in common with the exclusion set. For a structure element to be considered a candidate for the spatial search it must be accepted by each of the filters in the normal filter list and rejected by each of the filters in the inverted filter list. If the normal filter list is empty, then all structure elements satisfy the acceptance criteria for normal filters. If the inverted filter list is empty, all structure elements satisfy the rejection criteria for inverted filters. Graphical output structure elements which satisfy the search filters are checked for proximity to the reference point.

For TEXT structure elements, the only proximity relationship used is closeness to the text origin. For ANNOTATION TEXT RELATIVE structure elements, the only proximity relationship used is closeness to the annotation reference point.

The search will continue until either a graphical output structure element matches the search criteria or the end of the structure identified by the search ceiling is reached. If a search is successful, the complete search path is returned as a found path. An unsuccessful search returns a null found path.

The function is incremental in that, having found a match, the search may be continued by invoking INCREMENTAL SPATIAL SEARCH 3 again with the found path as the next starting path. This allows all elements matching the search criteria to be found for a given portion of a structure network.

**ERRORS**

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

203 Ignoring function, specified starting path not found in CSS

204 Ignoring function, specified search ceiling index out of range

**SEE ALSO**

ADD NAMES TO SET (3P)
REMOVE NAMES FROM SET (3P)
INCREMENTAL SPATIAL SEARCH (3P)

**NAME**

INITIALIZE CHOICE – initialize a CHOICE input device using 2D data

**SYNOPSIS**

**C Syntax**

**void**
**pinit_choice ( ws, dev, istat, init, pet, echo_area, record )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **dev;** | *choice device number* |
| **Pin_status** | **istat;** | *initial choice status* |
| **Pint** | **init;** | *initial choice* |
| **Pint** | **pet;** | *prompt and echo type* |
| **Plimit** | ***echo_area;** | *echo area pointer* |
| **Pchoice_data** | ***record;** | *data record pointer* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

INITIALIZE CHOICE sets the initialization parameters of a CHOICE device. This function stores these parameters in the workstation state list of the workstation associated with the specified device. The specified **device must be in REQUEST mode** when this function is called.

**C Input Parameters**

*ws* The workstation identifier of the workstation associated with the device.

*dev* The device number of the CHOICE device to initialize. See the *AVAILABLE DEVICES* section below for a description of the available devices.

*istat* The CHOICE status of the initial measure. Pin_status is an enumerated type defined in phigs.h as follows:

```
typedef enum {
        PIN_STATUS_NONE,
        PIN_STATUS_OK,
        PIN_STATUS_NO_IN
} Pin_status;
```

*init* The initial choice number. PHIGS ignores this value if *istat* is not PIN_STATUS_OK.

*pet* The prompt/echo type desired. Those supported by each device are listed in the *AVAILABLE DEVICES* section below.

*echo_area*

A pointer to a Plimit structure defining the *x* and *y* components of the echo volume, in Device Coordinates. The *z* component in the workstation state list is left unchanged. Plimit is defined in phigs.h as follows:

```
typedef struct {
        Pfloat      x_min;      /* minimum x coordinate value */
        Pfloat      x_max;      /* maximum x coordinate value */
        Pfloat      y_min;      /* minimum y coordinate value */
        Pfloat      y_max;      /* maximum y coordinate value */
} Plimit;
```

*record* A pointer to a Pchoice_data union containing the data record information. The contents of the data record for each device and prompt/echo type are described below in the *AVAILABLE DEVICES*

section.

The members of the data record union correspond to the prompt/echo type being used. As an example, the appropriate member of the Pchoice_data union for prompt/echo type 1 is *pet_r1*.

Pchoice_data is defined in phigs.h as:

```
typedef struct {
    union {
        struct {
          Pint          unused;
        } pet_r1;
        struct {
          Pint          num_prompts;  /* number of alternatives*/
          Ppr_switch    *prompts;     /* array of prompts*/
        } pet_r2;
        struct {
          Pint          num_strings;  /* number of choice strings*/
          char          **strings;    /* array of choice strings*/
        } pet_r3;
        struct {
          Pint          num_strings;  /* number of alternatives*/
          char          **strings;    /* array of strings*/
        } pet_r4;
        struct {
          Pint          struct_id;    /* struct identifier*/
          Pint          num_pick_ids; /* number of alternatives*/
          Pint          *pick_ids;    /* array of pick identifiers*/
        } pet_r5;
    } pets;
} Pchoice_data;
```

For some prompt/echo types the data record is not used; however, the *record* parameter must still be supplied.

**Execution**

INITIALIZE CHOICE sets the initialization parameters of a CHOICE device. This function stores these parameters in the workstation state list of the workstation associated with the specified device. The specified device must be in REQUEST mode when this function is called.

The parameters that are initialized by this function are *initial choice*, *prompt/echo type*, *echo volume*, and *input data record*.

The *initial choice* is the logical input value the device will take on whenever it is enabled. The device's measure retains this value until operator input changes it. A device is enabled when REQUEST CHOICE is called, or when its input mode is set to SAMPLE or EVENT.

A CHOICE device measure consists of a *status* and a *choice number*. *Status* indicates whether or not one of the possible choice values on the device was selected. (It's possible to trigger some CHOICE devices without selecting one of the choices.) The *choice number* indicates the value selected, if any.

The *prompt/echo type* determines how the device will be presented to the operator and respond to his actions. Each device supports one or more prompt/echo types. Those supported by each device are listed in the device's description in the *AVAILABLE DEVICES* section below. All devices support prompt/echo type 1. Positive prompt/echo types are defined by the PHIGS standard. Negative types are implementation-dependent.

The *echo volume* defines the region of the display surface in which to echo the device. It is specified in Device Coordinates (DC). Devices that use the echo volume may restrict their display to this region. Some of these devices may still recognize operator input outside the region even though they do not display there. This function only specifies the *x* and *y* components of the echo volume. The existing *z* component in the workstation state list is left unchanged.

The *input data record* contains the prompt/echo type specific information that controls the device's appearance and characteristics. Not all of the data record contents are used by some devices. The device descriptions in the *AVAILABLE DEVICES* section below list the data record contents that each device recognizes.

All the initialization parameters must be specified properly or this function will generate an error. The *ERRORS* section below lists the possible error conditions.

The default initialization parameters and the list of prompt/echo types supported by a CHOICE input device can be inquired with the function INQUIRE DEFAULT CHOICE DEVICE DATA. The current state of the device can be inquired with the function INQUIRE CHOICE DEVICE STATE.

## AVAILABLE DEVICES

### Device 1 - Athena List Widget

This device is a collection of widgets from the Athena widget set. The set consists of a shell widget, a viewport widget, and a list widget. The list is the item manipulated by the operator to change the choice value.

The echo volume is not used by this device.

**Prompt/echo types supported**: 1, 3

PET 1 — Display a PHIGS-defined list of choice strings in the menu. The choice strings are the integers *1* through *10*.

The data record is not used for this PET.

PET 3 — Display an application-specified list of choice strings in the menu. The data record member used is *pet_r3*, which contains the number and array of choice strings.

The widgets used and their names, in decreasing hierarchy, are:

choice1 → *applicationShellWidgetClass*
viewport → *viewportWidgetClass*
list → *listWidgetClass*

Fallback resources for choice devices are:

*choice1*allowHoriz → *TRUE*

*choice1*allowVert → *TRUE*

*choice1*list.translations —
<Btn1Down>,<Btn1Up> → *Set() Notify() RequestSatisfied()*

The fully qualified name of all widgets is <appl_name>.workstation<ws_id>.choice<dev_id>.<widget_name>, where <appl_name> is the application name specified in the call to OPEN XPHIGS. (This will be "phigs" if OPEN XPHIGS was not called.) For example, phigs.workstation1.choice1.viewport is the name of the viewport widget of choice device 1 on workstation 1.

## ERRORS

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

251 Ignoring function, the function requires the input device to be in REQUEST mode

253 Warning, the specified prompt/echo type is not available on the specified workstation. Prompt/echo type one will be used in its place

254 Ignoring function, invalid echo area/volume; XMIN ≥ XMAX, YMIN ≥ YMAX, or ZMIN > ZMAX

255 Ignoring function, one of the echo area/volume boundary points is outside the range of the device

260 Ignoring function, one of the fields within the input device data record is in error

261 Ignoring function, initial value is invalid

**SEE ALSO**

SET CHOICE MODE (3P)
REQUEST CHOICE (3P)
SAMPLE CHOICE (3P)
GET CHOICE (3P)
INQUIRE CHOICE DEVICE STATE (3P)
INITIALIZE CHOICE 3 (3P)

**NAME**

INITIALIZE CHOICE 3 – initialize a CHOICE input device using 3D data

**SYNOPSIS**

**C Syntax**

**void**
**pinit_choice3 ( ws, dev, istat, init, pet, echo_volume, record )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **dev;** | *choice device number* |
| **Pin_status** | **istat;** | *initial choice status* |
| **Pint** | **init;** | *initial choice* |
| **Pint** | **pet;** | *prompt and echo type* |
| **Plimit3** | ***echo_volume;** | *echo volume pointer* |
| **Pchoice_data3** | ***record;** | *data record pointer* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

INITIALIZE CHOICE 3 sets the initialization parameters of a CHOICE device. This function stores these parameters in the workstation state list of the workstation associated with the specified device. The specified **device must be in REQUEST mode** when this function is called.

**C Input Parameters**

*ws* The workstation identifier of the workstation associated with the device.

*dev* The device number of the CHOICE device to initialize. See the *Available Devices* section below for a description of the available devices.

*istat* The CHOICE status of the initial measure. Pin_status is an enumerated data type defined in phigs.h as follows:

```
typedef enum {
        PIN_STATUS_NONE,
        PIN_STATUS_OK,
        PIN_STATUS_NO_IN
} Pin_status;
```

*init* The initial choice number. PHIGS ignores this value if *istat* is not PIN_STATUS_OK.

*pet* The prompt/echo type desired. Those supported by each device are listed in the *Available Devices* section below.

*echo_volume*

A pointer to a Plimit3 structure defining the *x*, *y*, and *z* components of the echo volume, in Device Coordinates. Plimit3 is defined in phigs.h as follows:

```
typedef struct {
        Pfloat      x_min;      /* minimum x coordinate value */
        Pfloat      x_max;      /* maximum x coordinate value */
        Pfloat      y_min;      /* minimum y coordinate value */
        Pfloat      y_max;      /* maximum y coordinate value */
        Pfloat      z_min;      /* minimum z coordinate value */
        Pfloat      z_max;      /* maximum z coordinate value */
} Plimit3;
```

*record* A pointer to a Pchoice_data3 union containing the data record information. The contents of the data record for each device and prompt/echo type are described below in the *Available Devices* section.

The members of the data record union correspond to the prompt/echo type being used. As an example, the appropriate member of the Pchoice_data3 union for prompt/echo type 1 is *pet_r1*.

For some prompt/echo types the data record is not used. The *record* parameter, however, must still be supplied.

Pchoice_data3 is defined in phigs.h as:

```
typedef struct {
    union {
        struct {
          Pint          unused;
        } pet_r1;
        struct {
          Pint          num_prompts;  /* number of alternatives*/
          Ppr_switch    *prompts;     /* array of prompts*/
        } pet_r2;
        struct {
          Pint          num_strings;  /* number of choice strings*/
          char          **strings;    /* array of choice strings*/
        } pet_r3;
        struct {
          Pint          num_strings;  /* number of alternatives*/
          char          **strings;    /* array of strings*/
        } pet_r4;
        struct {
          Pint          struct_id;    /* struct identifier*/
          Pint          num_pick_ids; /* number of alternatives*/
          Pint          *pick_ids;    /* array of pick identifiers*/
        } pet_r5;
    } pets;
} Pchoice_data3;
```

**Execution**

INITIALIZE CHOICE 3 sets the initialization parameters of a CHOICE device. This function stores these parameters in the workstation state list of the workstation associated with the specified device. The specified device must be in REQUEST mode when this function is called.

The parameters that are initialized by this function are *initial choice*, *prompt/echo type*, *echo volume*, and *input data record*.

The *initial choice* is the logical input value the device will be set to whenever it is enabled. The device's measure retains this value until operator input changes it. A device is enabled when the appropriate REQUEST function is called, or when its input mode is set to SAMPLE or EVENT.

A CHOICE device measure consists of a *status* and a *choice number*. *Status* indicates whether or not one of the possible choice values on the device was selected. (It is possible to trigger some CHOICE devices without selecting one of the choices.) The *choice number* indicates the value selected, if any.

The *prompt/echo type* determines how the device will be presented to the operator and respond to his actions. Each device supports one or more prompt/echo types. Those supported by each device are listed in the device's description in the *AVAILABLE DEVICES* section below. All devices support prompt/echo type 1. Positive prompt/echo types are defined by the PHIGS standard. Negative types are implementation-dependent.

The *echo volume* defines the region of the display surface in which to echo the device. It is specified in Device Coordinates (DC). Devices that use the echo volume may restrict their display to this region. Some of these devices may still recognize operator input outside the region even though they don't display there.

The *input data record* contains the prompt/echo type specific information that controls the device's appearance and characteristics. Not all of the data record contents are used by some devices. The device descriptions in the *AVAILABLE DEVICES* section below list the data record contents that each device recognizes.

All the initialization parameters must be specified properly or this function will generate an error. The *ERRORS* section below lists the possible error conditions.

The default initialization parameters and the list of prompt/echo types supported by a CHOICE input device can be inquired with the function INQUIRE DEFAULT CHOICE DEVICE DATA 3. The current state of the device can be inquired with the function INQUIRE CHOICE DEVICE STATE 3.

## AVAILABLE DEVICES

### Device 1 - Athena List Widget

This device is a collection of widgets from the Athena widget set. The set consists of a shell widget, a viewport widget, and a list widget. The list is the item manipulated by the operator to change the choice value.

The echo volume is not used by this device.

**Prompt/echo types supported**: 1, 3

PET 1 Display a PHIGS-defined list of choice strings in the menu. The choice strings are the integers *1* through *10*.

The data record is not used for this PET.

PET 3 Display an application-specified list of choice strings in the menu. The data record member used is *pet_r3*, which contains the number and array of choice strings.

The widgets used and their names, in decreasing hierarchy, are:

choice001 → *applicationShellWidgetClass*
viewport → *viewportWidgetClass*
list → *listWidgetClass*

Fallback resources for choice devices are:

∗choice001∗allowHoriz → *TRUE*

∗choice001∗allowVert → *TRUE*

∗choice001∗list.translations —
<Btn1Down>,<Btn1Up> → *Set() Notify() RequestSatisfied()*

## ERRORS

003 Ignoring function, function requires state (PHOP, WSOP, ∗, ∗)

054 Ignoring function, the specified workstation is not open

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

251 Ignoring function, the function requires the input device to be in REQUEST mode

253 Warning, the specified prompt/echo type is not available on the specified workstation. Prompt/echo type one will be used in its place

254 Ignoring function, invalid echo area/volume; XMIN ≥ XMAX, YMIN ≥ YMAX, or ZMIN > ZMAX

255 Ignoring function, one of the echo area/volume boundary points is outside the range of the device

260 Ignoring function, one of the fields within the input device data record is in error

261 Ignoring function, initial value is invalid

## SEE ALSO

SET CHOICE MODE (3P)

REQUEST CHOICE (3P)
SAMPLE CHOICE (3P)
GET CHOICE (3P)
INQUIRE CHOICE DEVICE STATE 3 (3P)
INITIALIZE CHOICE (3P)

**NAME**

INITIALIZE LOCATOR – initialize a LOCATOR input device using 2D data

**SYNOPSIS**

**C Syntax**

**void**
**pinit_loc ( ws, dev, init_view_ind, init_loc_pos, pet, echo_area, record )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **dev;** | *locator device number* |
| **Pint** | **init_view_ind;** | *initial view indicator* |
| **Ppoint** | ***init_loc_pos;** | *initial locator pointer* |
| **Pint** | **pet;** | *prompt and echo type* |
| **Plimit** | ***echo_area;** | *echo area pointer* |
| **Ploc_data** | ***record;** | *data record pointer* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INITIALIZE LOCATOR to set the initialization parameters of a LOCATOR device. This function stores these parameters in the workstation state list of the workstation associated with the specified device. The specified **device must be in REQUEST mode** when this function is called.

**C Input Parameters**

*ws* The workstation identifier of the workstation associated with the device.

*dev* The device number of the LOCATOR device to initialize. See the *AVAILABLE DEVICES* section below for a description of the available devices.

*init_view_ind*

The index of the view representation in the workstation's view table to use to map the specified initial position from World Coordinates (WC) to Normalized Projection Coordinates (NPC).

*init_loc_pos*

A Ppoint structure specifying the *x* and *y* WC coordinates of the initial locator position. Ppoint is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          x;              /* x coordinate */
        Pfloat          y;              /* y coordinate */
} Ppoint;
```

*pet* The prompt/echo type desired. Those supported by each device are listed in the *AVAILABLE DEVICES* section below.

*echo_area*

A pointer to a Plimit structure defining the *x* and *y* components of the echo volume, in Device Coordinates (DC). The *z* component in the workstation state list is left unchanged. Plimit is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          x_min;          /* minimum x coordinate value */
        Pfloat          x_max;          /* maximum x coordinate value */
        Pfloat          y_min;          /* minimum y coordinate value */
        Pfloat          y_max;          /* maximum y coordinate value */
} Plimit;
```

*record* A pointer to a Ploc_data union containing the data record information. The contents of the data

record for each device and prompt/echo type are described below in the *AVAILABLE DEVICES* section.

The members of the data record union correspond to the prompt/echo type being used. As an example, the appropriate member of the Ploc_data union for prompt/echo type 1 is *pet_r1*.

For some prompt/echo types the locator data record is not used; however, the *record* parameter must still be supplied.

**Execution**

INITIALIZE LOCATOR sets the initialization parameters of a LOCATOR device. This function stores these parameters in the workstation state list of the workstation associated with the specified device. The specified device must be in REQUEST mode when this function is called.

The state list values set by this function are *initial value*, *prompt/echo type*, *echo volume*, and *input data record*.

The *initial value* is the logical input value the device will be set to whenever it is enabled. The device's measure retains this value until operator input changes it. A device is enabled when the appropriate *REQUEST* function is called, or when its input mode is set to SAMPLE or EVENT.

A LOCATOR device measure consists of a *position* and a *view index*. *Position* is the World Coordinate (WC) point corresponding to the position on the workstation selected by the operator. The *view index* is the index of the view representation used to transform the locator position from Normalized Projection Coordinates (NPC) to WC. This view representation is determined by selecting the highest priority representation that contains the locator position within its NPC limits. See SET VIEW TRANSFORMATION INPUT PRIORITY for more information. The workstation transform is used to transform the operator-selected position from Device Coordinates (DC) to NPC.

The initial locator position is transformed to DC by applying the view-orientation and view-mapping transforms of the specified view representation, and then applying the workstation transformation. If the view index is invalid, an error is generated.

The *prompt/echo type* determines how the device will be presented to the operator and respond to his actions. Each device supports one or more prompt/echo types. Those supported by each device are listed in the device's description in the *AVAILABLE DEVICES* section below. All devices support prompt/echo type 1. Positive prompt/echo types are defined by the PHIGS standard. Negative types are implementation-dependent.

The *echo volume* defines the region of the display surface in which to echo the device. It is specified in DC. Devices that use the echo volume may restrict their display to this region. This function only specifies the *x* and *y* components of the echo volume. The existing *z* component in the workstation state list is left unchanged.

The *input data record* contains the prompt/echo type specific information that controls the device's appearance and characteristics. Not all of the data record contents are used by some devices. The device descriptions in the *AVAILABLE DEVICES* section below list the data record contents that each device recognizes.

All the initialization parameters must be specified properly or this function will generate an error. The *ERRORS* section below lists the possible error conditions.

The default initialization parameters and the list of prompt/echo types supported by a LOCATOR input device can be inquired with the function INQUIRE DEFAULT LOCATOR DEVICE DATA. The current state of the device can be inquired with the function INQUIRE LOCATOR DEVICE STATE.

## AVAILABLE DEVICES

**Devices 1 to 3 - Pointer devices**

These devices are associated with the X pointer device, usually the mouse and cursor. Devices 1, 2, and 3 are triggered by the left, middle, and right mouse button respectively. The locator position in the *LOCATOR* device's measure is the WC position corresponding to the pointer event position. PHIGS computes the WC position from the two-dimensional pointer position by transforming the pointer position to a 2D NPC point,

determining the highest priority view containing that point, setting the *Z* coordinate of the NPC point to the lower *Z* limit of that view, and transforming the NPC point to world coordinates. This function returns only the *X* and *Y* coordinates of the resulting position.

Not all prompt/echo types use the initial locator position. Only those that need an additional point to the current pointer position use the initial position. The pointer is not moved to the initial position when the device is enabled.

This device does not use the echo volume.

**Prompt/echo types supported**:

PET 1 Display the default cursor at the current locator position.

The data record is not used for this PET.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

114 Ignoring function, the view index value is less than zero

250 Ignoring function, the specified device is not available on the specified workstation

251 Ignoring function, the function requires the input device to be in REQUEST mode

253 Warning, the specified prompt/echo type is not available on the specified workstation. Prompt/echo type one will be used in its place

254 Ignoring function, invalid echo area/volume; XMIN ≥ XMAX, YMIN ≥ YMAX, or ZMIN > ZMAX

255 Ignoring function, one of the echo area/volume boundary points is outside the range of the device

260 Ignoring function, one of the fields within the input device data record is in error

261 Ignoring function, initial value is invalid

**SEE ALSO**

SET LOCATOR MODE (3P)
REQUEST LOCATOR (3P)
SAMPLE LOCATOR (3P)
GET LOCATOR (3P)
INQUIRE LOCATOR DEVICE STATE (3P)
SET VIEW TRANSFORMATION INPUT PRIORITY (3P)
INITIALIZE LOCATOR 3 (3P)

**NAME**

INITIALIZE LOCATOR 3 – initialize a LOCATOR input device using 3D data

**SYNOPSIS**

**C Syntax**

**void**
**pinit_loc3 ( ws, dev, init_view_ind, init_loc_pos, pet, echo_volume, record )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **dev;** | *locator device number* |
| **Pint** | **init_view_ind;** | *initial view indicator* |
| **Ppoint3** | ***init_loc_pos;** | *initial locator position* |
| **Pint** | **pet;** | *prompt and echo type* |
| **Plimit3** | ***echo_volume;** | *echo volume pointer* |
| **Ploc_data3** | ***record;** | *data record pointer* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INITIALIZE LOCATOR 3 to set the initialization parameters of a LOCATOR device. This function stores these parameters in the workstation state list of the workstation associated with the device. The specified **device must be in REQUEST mode** when this function is called.

**C Input Parameters**

*ws* The workstation identifier of the workstation associated with the device.

*dev* The device number of the LOCATOR device to initialize. See the *AVAILABLE DEVICES* section below for a description of the available devices.

*init_view_ind*

The index of the view representation in the workstation's view table to use to map the specified initial position from World Coordinates (WC) to Normalized Projection Coordinates (NPC).

*init_loc_pos*

A Ppoint3 structure specifying the initial locator position, in World Coordinates. Ppoint3 is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          x;              /* x coordinate */
        Pfloat          y;              /* y coordinate */
        Pfloat          z;              /* z coordinate */
} Ppoint3;
```

*pet* The prompt/echo type desired. Those supported by each device are listed in the *AVAILABLE DEVICES* section below.

*echo_volume*

A pointer to a Plimit3 structure specifying the echo volume, in Device Coordinates (DC). Plimit3 is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          x_min;          /* minimum x coordinate value */
        Pfloat          x_max;          /* maximum x coordinate value */
        Pfloat          y_min;          /* minimum y coordinate value */
        Pfloat          y_max;          /* maximum y coordinate value */
        Pfloat          z_min;          /* minimum z coordinate value */
        Pfloat          z_max;          /* maximum z coordinate value */
} Plimit3;
```

*record* A pointer to a Ploc_data3 union containing the data record information. The contents of the data record for each device and prompt/echo type are described below in the *AVAILABLE DEVICES* section.

The members of the data record union correspond to the prompt/echo type being used. As an example, the appropriate member of the Ploc_data3 union for prompt/echo type 1 is *pet_r1*.

For some prompt/echo types the locator data record is not used; however, the *record* parameter must still be supplied.

**Execution**

INITIALIZE LOCATOR 3 sets the initialization parameters of a LOCATOR device. This function stores these parameters in the workstation state list of the workstation associated with the specified device. The specified device must be in REQUEST mode when this function is called.

The parameters that are initialized by this function are *initial value*, *prompt/echo type*, *echo volume*, and *input data record*.

The *initial value* is the logical input value the device will be set to whenever it is enabled. The device's measure retains this value until operator input changes it. A device is enabled when the appropriate *REQUEST* function is called, or when its input mode is set to SAMPLE or EVENT.

A LOCATOR device measure consists of a *position* and a *view index*. *Position* is the World Coordinate (WC) point corresponding to the position on the workstation selected by the operator. The *view index* is the index of the view representation used to transform the locator position from Normalized Projection Coordinates (NPC) to WC. This view representation is determined by selecting the highest priority representation that contains the locator position within its NPC limits. See SET VIEW TRANSFORMATION INPUT PRIORITY for more information. The workstation transform is used to transform the operator-selected position from Device Coordinates (DC) to NPC.

The initial locator position is transformed to DC by applying the view-orientation and view-mapping transforms of the specified view representation, and then applying the workstation transformation. If the view index is invalid, an error is generated.

The *prompt/echo type* determines how the device will be presented to the operator and respond to his actions. Each device supports one or more prompt/echo types. Those supported by each device are listed in the device's description in the *AVAILABLE DEVICES* section below. All devices support prompt/echo type 1. Positive prompt/echo types are defined by the PHIGS standard. Negative types are implementation-dependent.

The *echo volume* defines the region of the display surface in which to echo the device. It is specified in DC. Devices that use the echo volume may restrict their display to this region. Some of these devices will still recognize operator input outside the region even though they don't display there.

The *input data record* contains the prompt/echo type specific information that controls the device's appearance and characteristics. Not all the data record contents are used by some devices. The device descriptions in the *AVAILABLE DEVICES* section below list the data record contents that each device recognizes.

All the initialization parameters must be specified properly or this function will generate an error. The *ERRORS* section below lists the possible error conditions.

The default initialization parameters and the list of prompt/echo types supported by a LOCATOR input device can be inquired with the function INQUIRE DEFAULT LOCATOR DEVICE DATA 3. The current state of the device can be inquired with the function INQUIRE LOCATOR DEVICE STATE 3.

## AVAILABLE DEVICES

**Devices 1 to 3 - Pointer devices**

These devices are associated with the X pointer device, usually the mouse and cursor. Devices 1, 2, and 3 are triggered by the left, middle, and right mouse buttons respectively. The locator position in the *LOCATOR* device's measure is the WC position corresponding to the pointer position in X pointer events. PHIGS computes the WC position from the two-dimensional pointer position by transforming the pointer position to a

2D NPC point, determining the highest priority view containing that point, setting the *Z* coordinate of the *NPC* point to the lower *Z* limit of that view, and transforming the *NPC* point to world coordinates.

Not all prompt/echo types use the initial locator position. Only those that need an additional point to the current pointer position use the initial position. The pointer is not moved to the initial position when the device is enabled.

This device does not use the echo volume.

**Prompt/echo types supported**:

PET 1 Display the default cursor at the current locator position.

The data record is not used for this PET.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

114 Ignoring function, the view index value is less than zero

250 Ignoring function, the specified device is not available on the specified workstation

251 Ignoring function, the function requires the input device to be in REQUEST mode

253 Warning, the specified prompt/echo type is not available on the specified workstation.

254 Ignoring function, invalid echo area/volume; XMIN ≥ XMAX, YMIN ≥ YMAX, or ZMIN > ZMAX

255 Ignoring function, one of the echo area/volume boundary points is outside the range of the device

260 Ignoring function, one of the fields within the input device data record is in error. Prompt/echo type one will be used in its place

261 Ignoring function, initial value is invalid

**SEE ALSO**

SET LOCATOR MODE (3P)
REQUEST LOCATOR 3 (3P)
SAMPLE LOCATOR 3 (3P)
GET LOCATOR 3 (3P)
INQUIRE LOCATOR DEVICE STATE 3 (3P)
SET VIEW TRANSFORMATION INPUT PRIORITY (3P)
INITIALIZE LOCATOR (3P)

**NAME**

INITIALIZE PICK – initialize a PICK input device using 2D data

**SYNOPSIS**

**C Syntax**

**void**
**pinit_pick ( ws, dev, istat, init, pet, echo_area, record, order )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **dev;** | *pick device number* |
| **Pin_status** | **istat;** | *initial pick status* |
| **Ppick_path** | ***init;** | *initial pick pointer* |
| **Pint** | **pet;** | *prompt and echo type* |
| **Plimit** | ***echo_area;** | *echo area pointer* |
| **Ppick_data** | ***record;** | *data record pointer* |
| **Ppath_order** | **order;** | *pick path order* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

INITIALIZE PICK sets the initialization parameters of a PICK device. This function stores these parameters in the workstation state list of the workstation associated with the specified device. The specified **device must be in REQUEST mode** when this function is called.

The *pick filter* of a PICK device controls which output primitives on the device's workstation are pickable. By default no output primitives are pickable. See SET PICK FILTER for more information on the pick filter.

**C Input Parameters**

*ws* The workstation identifier of the workstation associated with the device.

*dev* The device number of the PICK device to initialize. See the *AVAILABLE DEVICES* section below for a description of the available devices.

*istat* The PICK status of the initial measure. Pin_status is an enumerated type defined in phigs.h as follows:

```
typedef enum {
        PIN_STATUS_NONE,
        PIN_STATUS_OK,
        PIN_STATUS_NO_IN
} Pin_status;
```

*init* A pointer to a Ppick_path structure containing the initial pick path, if any. This value is ignored if *status* is not PIN_STATUS_OK. Ppick_path is defined in phigs.h as:

```
typedef struct {
        Pint            depth;      /* pick path_list depth */
        Ppick_path_elem *path_list; /* pick path */
} Ppick_path;
```

The depth indicates the number of elements in the path.

*path_list* is the array of path elements defining the location of the primitive in the CSS. Ppick_path_elem is defined in phigs.h as:

```
typedef struct {
        Pint        struct_id;      /* structure identifier */
        Pint        pick_id;        /* hierarchical pick identifier */
        Pint        elem_pos;       /* element sequence number */
} Ppick_path_elem;
```

The struct_id, pick_id, and elem_pos are the structure identifier, pick id, and element number, respectively, of each element in the path.

*pet* The prompt/echo type desired. Those supported by each device are listed in the *AVAILABLE DEVICES* section below.

*echo_area*

A pointer to a Plimit structure specifying the *x* and *y* components of the echo volume, in Device Coordinates. The *z* component in the workstation state list is left unchanged. Plimit is defined in phigs.h as follows:

```
typedef struct {
        Pfloat      x_min;          /* minimum x coordinate value */
        Pfloat      x_max;          /* maximum x coordinate value */
        Pfloat      y_min;          /* minimum y coordinate value */
        Pfloat      y_max;          /* maximum y coordinate value */
} Plimit;
```

*record* A pointer to a Ppick_data union containing the data record information. The contents of the data record for each device and prompt/echo type are described below in the *AVAILABLE DEVICES* section.

The members of the data record union correspond to the prompt/echo type being used. As an example, the appropriate member of the Ppick_data union for prompt/echo type 1 is *pet_r1*.

For some prompt/echo types the data record is not used; however, the record parameter must still be specified.

*order* The pick path order of paths returned by REQUEST PICK, GET PICK, and SAMPLE PICK. Ppath_order is an enumerated type defined in phigs.h as follows:

```
typedef enum {
        PORDER_TOP_FIRST,
        PORDER_BOTTOM_FIRST
} Ppath_order;
```

If the order is TOP FIRST, the structure specified in any pick path element is a parent of the structure specified in the subsequent pick path element. If the order is BOTTOM FIRST, the structure specified in any pick path element is a child of the structure specified in the subsequent pick path element.

**Execution**

INITIALIZE PICK sets the initialization parameters of a PICK device. This function stores these parameters in the workstation state list of the workstation associated with the specified device. The specified device must be in REQUEST mode when this function is called.

The parameters that are initialized by this function are the *initial path*, *prompt/echo type*, *echo volume*, and *input data record*.

The *initial path* is the logical input value the device will take on whenever it is enabled. The device's measure retains this value until operator input changes it. A device is enabled when the appropriate REQUEST function is called, or when its input mode is set to SAMPLE or EVENT.

A PICK device measure consists of a *status* and a *pick path*. Status indicates whether or not an initial pick path is specified. The pick path, if any, describes the location of a picked primitive in the Central Structure Store (CSS).

The *pick filter* of a PICK device controls which output primitives on the device's workstation are pickable. By default no output primitives are pickable. See SET PICK FILTER for more information on the pick filter.

The *prompt/echo type* determines how the device will be presented to the operator and respond to his actions. Each device supports one or more prompt/echo types. Those supported by each device are listed in the device's description in the *AVAILABLE DEVICES* section below. All devices support prompt/echo type 1. Positive prompt/echo types are defined by the PHIGS standard. Negative types are implementation-dependent.

The *echo volume* defines the region of the display surface in which to echo the device. It is specified in Device Coordinates (DC). Devices that use the echo volume may restrict their display to this region. Some of these devices may still recognize operator input outside the region even though they don't display there. This function only specifies the *x* and *y* components of the echo volume. The existing *z* component in the workstation state list is left unchanged.

The *input data record* contains the prompt/echo type specific information that controls the device's appearance and characteristics. Not all of the data record contents are used by some devices. The device descriptions in the *AVAILABLE DEVICES* section below list the data record contents that each device recognizes.

All the initialization parameters must be specified properly or this function will generate an error. The *ERRORS* section below lists the possible error conditions.

The default initialization parameters and the list of prompt/echo types supported by a PICK input device can be inquired with the function INQUIRE DEFAULT PICK DEVICE DATA. The current state of the device can be inquired with the function INQUIRE PICK DEVICE STATE.

## AVAILABLE DEVICES

### Device 1 - Pointer device

This device is associated with the X pointer device. The operator picks an output primitive by selecting a displayed primitive with the pointer.

When a pick is attempted by the operator, PHIGS searches the workstation's list of posted structures for visible and pickable output primitives that intersect a 3D region around the pointer. This region is called the pick *aperture*. The search starts at the root of the posted structure network with highest priority and continues until the entire network has been traversed. The **last** pickable primitive in this network that intersects the aperture is the picked primitive. If no such primitive is found, the search continues with the next highest priority structure network until a primitive is found or there are no more structure networks to search.

If no pickable primitive is found to intersect the aperture, the PICK device measure status is set to PIN_STATUS_NONE.

This device does not use the echo volume.

Only prompt/echo type 1 is supported.

Picking is only supported for X servers that support the full PEX extension.

## ERRORS

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

060 Ignoring function, specified workstation is not of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

251 Ignoring function, the function requires the input device to be in REQUEST mode

253 Warning, the specified prompt/echo type is not available on the specified workstation. Prompt/echo type one will be used in its place

254 Ignoring function, invalid echo area/volume; XMIN ≥ XMAX, YMIN ≥ YMAX, or ZMIN > ZMAX

255 Ignoring function, one of the echo area/volume boundary points is outside the range of the device

260 Ignoring function, one of the fields within the input device data record is in error

261 Ignoring function, initial value is invalid

**SEE ALSO**

SET PICK FILTER (3P)
SET PICK IDENTIFIER (3P)
SET PICK MODE (3P)
REQUEST PICK (3P)
SAMPLE PICK (3P)
GET PICK (3P)
INQUIRE PICK DEVICE STATE (3P)
INITIALIZE PICK 3 (3P)

**NAME**

INITIALIZE PICK 3 – initialize a PICK input device using 3D data.

**SYNOPSIS**

**C Syntax**

**void**
**pinit_pick3 ( ws, dev, istat, init, pet, echo_volume, record, order )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **dev;** | *pick device number* |
| **Pin_status** | **istat;** | *initial pick status* |
| **Ppick_path** | ***init;** | *initial pick pointer* |
| **Pint** | **pet;** | *prompt and echo type* |
| **Plimit3** | ***echo_volume;** | *echo volume pointer* |
| **Ppick_data3** | ***record;** | *data record pointer* |
| **Ppath_order** | **order;** | *pick path order* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INITIALIZE PICK 3 to set the initialization parameters of a PICK device. This function stores these parameters in the workstation state list of the workstation associated with the specified device. The specified **device must be in REQUEST mode** when this function is called.

The *pick filter* of a PICK device controls which output primitives on the device's workstation are pickable. By default no output primitives are pickable. See SET PICK FILTER for more information on the pick filter.

**C Input Parameters**

*ws* The workstation identifier of the workstation associated with the device.

*dev* The device number of the PICK device to initialize. See the *AVAILABLE DEVICES* section below for a description of the available devices.

*istat* The PICK status of the initial measure. Pin_status is an enumerated type defined in phigs.h as follows:

```
typedef enum {
        PIN_STATUS_NONE,
        PIN_STATUS_OK,
        PIN_STATUS_NO_IN
} Pin_status;
```

*init* A pointer to a Ppick_path structure containing the initial pick path, if any. This value is ignored if *status* is not PIN_STATUS_OK. Ppick_path is defined in phigs.h as:

```
typedef struct {
        Pint            depth;      /* pick path_list depth */
        Ppick_path_elem *path_list; /* pick path */
} Ppick_path;
```

The depth indicates the number of elements in the path.

*path_list* is the array of path elements defining the location of the primitive in the CSS. Ppick_path_elem is defined in phigs.h as:

```
typedef struct {
        Pint        struct_id;      /* structure identifier */
        Pint        pick_id;        /* hierarchical pick identifier */
        Pint        elem_pos;       /* element sequence number */
} Ppick_path_elem;
```

The struct_id, pick_id, and elem_pos are the structure identifier, pick id, and element number, respectively, of each element in the path.

*pet* The prompt/echo type desired. Those supported by each device are listed in the *AVAILABLE DEVICES* section below.

*echo_volume*

A pointer to a Plimit3 structure spedifying the echo volume, in Device Coordinates. Plimit3 is defined in phigs.h as follows:

```
typedef struct {
        Pfloat      x_min;          /* minimum x coordinate value */
        Pfloat      x_max;          /* maximum x coordinate value */
        Pfloat      y_min;          /* minimum y coordinate value */
        Pfloat      y_max;          /* maximum y coordinate value */
        Pfloat      z_min;          /* minimum z coordinate value */
        Pfloat      z_max;          /* maximum z coordinate value */
} Plimit3;
```

*record* A pointer to a Ppick_data3 union containing the data record information. The contents of the data record for each device and prompt/echo type are described below in the *AVAILABLE DEVICES* section.

The members of the data record union correspond to the prompt/echo type being used. As an example, the appropriate member of the Ppick_data3 union for prompt/echo type 1 is *pet_r1*.

For some prompt/echo types the data record is not used; however, the record parameter must still be specified.

*order* The pick path order of paths returned by REQUEST PICK, GET PICK, and SAMPLE PICK. Ppath_order is an enumerated type defined in phigs.h as follows:

```
typedef enum {
        PORDER_TOP_FIRST,
        PORDER_BOTTOM_FIRST
} Ppath_order;
```

If the order is TOP FIRST, the structure specified in any pick path element is a parent of the structure specified in the subsequent pick path element. If the order is BOTTOM FIRST, the structure specified in any pick path element is a child of the structure specified in the subsequent pick path element.

**Execution**

INITIALIZE PICK 3 sets the initialization parameters of a PICK device. This function stores these parameters in the workstation description table of the workstation associated with the specified device. The specified device must be in REQUEST mode when this function is called.

The parameters that are initialized by this function are the *initial path*, *prompt/echo type*, *echo volume*, and *input data record*.

The *initial path* is the logical input value the device will take on whenever it is enabled. The device's measure retains this value until operator input changes it. A device is enabled when the appropriate REQUEST function is called, or when its input mode is set to SAMPLE or EVENT.

A PICK device measure consists of a *status* and a *pick path*. Status indicates whether or not an initial pick path is specified. The pick path, if any, describes the location of a picked primitive in the Central Structure Store (CSS).

The *pick filter* of a PICK device controls which output primitives on the device's workstation are pickable. By default no output primitives are pickable. See SET PICK FILTER for more information on the pick filter.

The *prompt/echo type* determines how the device will be presented to the operator and respond to his actions. Each device supports one or more prompt/echo types. Those supported by each device are listed in the device's description in the *AVAILABLE DEVICES* section below. All devices support prompt/echo type 1. Positive prompt/echo types are defined by the PHIGS standard. Negative types are implementation-dependent.

The *echo volume* defines the region of the display surface in which to echo the device. It is specified in Device Coordinates (DC). Devices that use the echo volume may restrict their display to this region. Some of these devices may still recognize operator input outside the region even though they don't display there.

The *input data record* contains the prompt/echo type specific information that controls the device's appearance and characteristics. Not all the data record contents are used by some devices. The device descriptions in the *AVAILABLE DEVICES* section below list the data record contents that each device recognizes.

All the initialization parameters must be specified properly or this function will generate an error. The *ERRORS* section below lists the possible error conditions.

The default initialization parameters and the list of prompt/echo types supported by a PICK input device can be inquired with the function INQUIRE DEFAULT PICK DEVICE DATA 3. The current state of the device can be inquired with the function INQUIRE PICK DEVICE STATE 3.

## AVAILABLE DEVICES

### Device 1 - Pointer device

This device is associated with the X pointer device. The operator picks an output primitive by selecting a displayed primitive with the pointer.

When a pick is attempted by the operator, PHIGS searches the workstation's list of posted structures for visible and pickable output primitives that intersect a 3D region around the pointer. This region is called the pick *aperture*. The search starts at the root of the posted structure network with highest priority and continues until the entire network has been traversed. The **last** pickable primitive in this network that intersects the aperture is the picked primitive. If no such primitive is found, the search continues with the next highest priority structure network until a primitive is found or there are no more structure networks to search.

If no pickable primitive is found to intersect the aperture, the PICK device measure status is set to PIN_STATUS_NONE.

This device does not use the echo volume.

Only prompt/echo type 1 is supported.

Picking is only supported for X servers that support the full PEX extension.

## ERRORS

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

060 Ignoring function, specified workstation is not of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

251 Ignoring function, the function requires the input device to be in REQUEST mode

253 Warning, the specified prompt/echo type is not available on the specified workstation. Prompt/echo type one will be used in its place

254 Ignoring function, invalid echo area/volume; XMIN ≥ XMAX, YMIN ≥ YMAX, or ZMIN > ZMAX

255 Ignoring function, one of the echo area/volume boundary points is outside the range of the device

260 Ignoring function, one of the fields within the input device data record is in error

261 Ignoring function, initial value is invalid

**SEE ALSO**

SET PICK FILTER (3P)
SET PICK IDENTIFIER (3P)
SET PICK MODE (3P)
REQUEST PICK (3P)
SAMPLE PICK (3P)
GET PICK (3P)
INQUIRE PICK DEVICE STATE 3 (3P)
INITIALIZE PICK (3P)

**NAME**

INITIALIZE STRING – initialize a STRING input device using 2D data

**SYNOPSIS**

**C Syntax**

**void**
**pinit_string ( ws, dev, init, pet, echo_area, record )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **dev;** | *string device number* |
| **char** | ***init;** | *initial string* |
| **Pint** | **pet;** | *prompt and echo type* |
| **Plimit** | ***echo_area;** | *echo area pointer* |
| **Pstring_data** | ***record;** | *data record pointer* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

INITIALIZE STRING sets the initialization parameters of a STRING device. This function stores these parameters in the workstation state list of the workstation associated with the specified device. The specified **device must be in REQUEST mode** when this function is called.

**C Input Parameters**

*ws* The workstation identifier of the workstation associated with the device.

*dev* The device number of the STRING device to initialize. See the *AVAILABLE DEVICES* section below for a description of the available devices.

*init* A pointer to the initial string. *init* can be NULL, that is, (char*)0.

*pet* The prompt/echo type desired. Those supported by each device are listed in the *AVAILABLE DEVICES* section below.

*echo_area*

A pointer to a Plimit structure defining the *x* and *y* components of the echo volume, in Device Coordinates (DC). The *z* component in the workstation state list is left unchanged. Plimit is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          x_min;          /* minimum x coordinate value */
        Pfloat          x_max;          /* maximum x coordinate value */
        Pfloat          y_min;          /* minimum y coordinate value */
        Pfloat          y_max;          /* maximum y coordinate value */
} Plimit;
```

*record* A pointer to a Pstring_data structure containing the data record information. The contents of the data record for each device and prompt/echo type are described below in the *Available Devices* section.

The members of the data record union correspond to the prompt/echo type being used. As an example, the appropriate member of the Pstring_data structure for prompt/echo type 1 is *pet_r1*.

For some prompt/echo types of some STRING devices the data record is not used. The *record* parameter, however, must still be supplied.

Pstring_data3 is defined in phigs.h as follows:

```
typedef struct {
        Pint    buffer_size;        /* input buffer size*/
        Pint    init_pos;           /* initial editing position*/
        union {
                struct {
                        Pint    unused;
                } pet_r1;
        } pets;
} Pstring_data;
```

**Execution**

INITIALIZE STRING sets the initialization parameters of a STRING device. This function stores these parameters in the workstation state list of the workstation associated with the specified device. The specified device must be in REQUEST mode when this function is called.

The parameters that are initialized by this function are *initial string*, *prompt/echo type*, *echo volume*, and *input data record*.

The *initial string* is the logical input value the device will take on whenever it is enabled. The device's measure retains this value until operator input changes it. A device is enabled when the appropriate REQUEST function is called, or when its input mode is set to SAMPLE or EVENT.

A STRING device measure consists of a character string.

The *prompt/echo type* determines how the device will be presented to the operator and respond to his actions. Each device supports one or more prompt/echo types. Those supported by each device are listed in the device's description in the *AVAILABLE DEVICES* section below. All devices support prompt/echo type 1. Positive prompt/echo types are defined by the PHIGS standard. Negative types are implementation-dependent.

The *echo volume* defines the region of the display surface in which to echo the device. It is specified in Device Coordinates (DC). Devices that use the echo volume may restrict their display to this region. Some of these devices may still recognize operator input outside the region even though they do not display there. This function only specifies the *x* and *y* components of the echo volume. The existing *z* component in the workstation state list is left unchanged.

The *input data record* contains the prompt/echo type specific information that controls the device's appearance and characteristics. Not all of the data record contents are used by some devices. The device descriptions in the *AVAILABLE DEVICES* section below list the data record contents that each device recognizes.

All the initialization parameters must be specified properly, or this function will generate an error. The *ERRORS* section below lists the possible error conditions.

The default initialization parameters and the list of prompt/echo types supported by a STRING input device can be inquired with the function INQUIRE DEFAULT STRING DEVICE DATA. The current state of the device can be inquired with the function INQUIRE STRING DEVICE STATE.

**AVAILABLE DEVICES**

**Device 1 - Athena Ascii Text Widget**

The string device realization is a collection of widgets from the Athena widget set. It consists of a Shell widget, a Viewport widget, and an AsciiText widget. The AsciiText widget is the item manipulated by the operator to change the *string value*.

String device events are generated by a carriage return. A carriage return also invokes the *RequestSatisfied* action. The device's measure corresponds to all the characters added to the widget by the operator since the device was last enabled or the last STRING event was generated. The string device has the following translation defined:

<Key>Return → *newline() StringEvent() RequestSatisfied()*

The single prompt/echo type supported is 1.

The widgets used and their names, in decreasing hierarchy, are:

string1 → *applicationShellWidgetClass*
viewport → *viewportWidgetClass*
text → *asciiTextWidgetClass*

Fallback resources for string devices are:

*string1*Text*width → *400*

*string1*Text*height → *50*

*string1*Text*editType → *edit*

*string1*Text*scrollVertical → *never*

*string1*Text*scrollHorizontal → *never*

*string1*Text*autoFill → *on*

The fully qualified name of all widgets is <appl_name>.workstation<ws_id>.string<dev_id>.<widget_name>, where <appl_name> is the application name specified in the call to OPEN XPHIGS. (This will be "phigs" if OPEN XPHIGS was not called.) For example, phigs.workstation1.string.viewport is the name of the pane widget of string device 1 on workstation 1.

This device does not use the PHIGS echo area. Applications or users can specify the position of the devices (subject to window manager control) by specifying the appropriate resource values in a resource file.

The echo volume is ignored by this device. The *buffer_size* and *init_pos* fields of the data record are used by this device.

**Prompt/echo types supported**: 1

PET 1 Display the text in the widget.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

251 Ignoring function, the function requires the input device to be in REQUEST mode

253 Warning, the specified prompt/echo type is not available on the specified workstation. Prompt/echo type one will be used in its place

254 Ignoring function, invalid echo area/volume; XMIN ≥ XMAX, YMIN ≥ YMAX, or ZMIN > ZMAX

255 Ignoring function, one of the echo area/volume boundary points is outside the range of the device

260 Ignoring function, one of the fields within the input device data record is in error

261 Ignoring function, initial value is invalid

263 Ignoring function, length of the initial string is greater than the buffer size

**SEE ALSO**

SET STRING MODE (3P)
REQUEST STRING (3P)
SAMPLE STRING (3P)
GET STRING (3P)
INQUIRE STRING DEVICE STATE (3P)
INITIALIZE STRING 3 (3P)

**NAME**

INITIALIZE STRING 3 – initialize a STRING input device using 3D data

**SYNOPSIS**

**C Syntax**

```
void
pinit_string3 ( ws, dev, init, pet, echo_volume, record )
Pint            ws;             workstation identifier
Pint            dev;            string device number
char            *init;          initial string
Pint            pet;            prompt and echo type
Plimit3         *echo_volume;   echo volume pointer
Pstring_data3   *record;        data record pointer
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

INITIALIZE STRING 3 sets the initialization parameters of a STRING device. This function stores these parameters in the workstation state list of the workstation associated with the specified device. **The specified device must be in REQUEST mode when this function is called.**

**C Input Parameters**

*ws* The workstation identifier of the workstation associated with the device.

*dev* The device number of the STRING device to initialize. See the *AVAILABLE DEVICES* section below for a description of the available devices.

*init* A pointer to the initial string. *init* can be NULL, that is, (char*)0.

*pet* The prompt/echo type desired. Those supported by each device are listed in the *AVAILABLE DEVICES* section below.

*echo_volume*

A pointer to a Plimit3 structure defining the *x*, *y*, and *z* components of the echo volume, in Device Coordinates (DC). Plimit3 is defined in phigs.h as follows:

```
typedef struct {
        Pfloat      x_min;      /* minimum x coordinate value */
        Pfloat      x_max;      /* maximum x coordinate value */
        Pfloat      y_min;      /* minimum y coordinate value */
        Pfloat      y_max;      /* maximum y coordinate value */
        Pfloat      z_min;      /* minimum z coordinate value */
        Pfloat      z_max;      /* maximum z coordinate value */
} Plimit3;
```

*record* A pointer to a Pstring_data3 structure containing the data record information. The contents of the data record for each device and prompt/echo type are described below in the *AVAILABLE DEVICES* section.

The members of the data record union correspond to the prompt/echo type being used. As an example, the appropriate member of the Pstring_data3 structure for prompt/echo type 1 is *pet_r1*.

For some prompt/echo types of some STRING devices, the data record is not used. The *record* parameter, however, must still be supplied.

Pstring_data3 is defined in phigs.h as follows:

```
typedef struct {
        Pint    buffer_size;        /* input buffer size*/
        Pint    init_pos;           /* initial editing position*/
        union {
                struct {
                        Pint    unused;
                } pet_r1;
        } pets;
} Pstring_data3;
```

**Execution**

INITIALIZE STRING 3 sets the initialization parameters of a STRING device. This function stores these parameters in the workstation state list of the workstation associated with the specified device. The specified device must be in REQUEST mode when this function is called.

The parameters that are initialized by this function are *initial string*, *prompt/echo type*, *echo volume*, and *input data record*.

The *initial string* is the logical input value the device will be set to whenever it is enabled. The device's measure retains this value until operator input changes it. A device is enabled when the appropriate *REQUEST* function is called, or when its input mode is set to SAMPLE or EVENT.

A STRING device measure consists of a character string.

The *prompt/echo type* determines how the device will be presented to the operator and respond to his actions. Each device supports one or more prompt/echo types. Those supported by each device are listed in the device's description in the *AVAILABLE DEVICES* section below. All devices support prompt/echo type 1. Positive prompt/echo types are defined by the PHIGS standard. Negative types are implementation-dependent.

The *echo volume* defines the region of the display surface in which to echo the device. It is specified in Device Coordinates (DC). Devices that use the echo volume may restrict their display to this region. Some of these devices may still recognize operator input outside the region even though they do not display there.

The *input data record* contains the prompt/echo type specific information that controls the device's appearance and characteristics. Not all of the data record contents are used by some devices. The device descriptions in the *AVAILABLE DEVICES* section below list the data record contents that each device recognizes.

All the initialization parameters must be properly specified, or this function will generate an error. The *ERRORS* section below lists the possible error conditions.

The default initialization parameters and the list of prompt/echo types supported by a STRING input device can be inquired with the function INQUIRE DEFAULT STRING DEVICE DATA 3. The current state of the device can be inquired with the function INQUIRE STRING DEVICE STATE 3.

**AVAILABLE DEVICES**

**Device 1 - Athena Ascii Text Widget**

The string device realization is a collection of widgets from the Athena widget set. It consists of a Shell widget, a Viewport widget, and an AsciiText widget. The AsciiText widget is the item manipulated by the operator to change the *string value*.

String device events are generated by a carriage return. A carriage return also invokes the *RequestSatisfied* action. The device's measure corresponds to all the characters added to the widget by the operator since the device was last enabled or the last STRING event was generated. The string device has the following translation defined:

<Key>Return → *newline() StringEvent() RequestSatisfied()*

The single prompt/echo type supported is 1.

The widgets used and their names, in decreasing hierarchy, are:

string001 → *applicationShellWidgetClass*
viewport → *viewportWidgetClass*
text → *asciiTextWidgetClass*

Fallback resources for string devices are:

∗string001∗Text∗width → *400*

∗string001∗Text∗height → *50*

∗string001∗Text∗editType → *edit*

∗string001∗Text∗scrollVertical → *never*

∗string001∗Text∗scrollHorizontal → *never*

∗string001∗Text∗autoFill → *on*

This device does not use the PHIGS echo area. Applications or users can specify the position of the devices (subject to window manager control) by specifying the appropriate resource values in a resource file.

The echo volume is ignored by this device. The *buffer_size* and *init_pos* fields of the data record are used by this device.

**Prompt/echo types supported**: 1

PET 1 Display the text in the widget.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, ∗, ∗)

054 Ignoring function, the specified workstation is not open

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

251 Ignoring function, the function requires the input device to be in REQUEST mode

253 Warning, the specified prompt/echo type is not available on the specified workstation. Prompt/echo type one will be used in its place

254 Ignoring function, invalid echo area/volume; XMIN ≥ XMAX, YMIN ≥ YMAX, or ZMIN > ZMAX

255 Ignoring function, one of the echo area/volume boundary points is outside the range of the device

260 Ignoring function, one of the fields within the input device data record is in error

261 Ignoring function, initial value is invalid

263 Ignoring function, length of the initial string is greater than the buffer size

**SEE ALSO**

SET STRING MODE (3P)
REQUEST STRING (3P)
SAMPLE STRING (3P)
GET STRING (3P)
INQUIRE STRING DEVICE STATE 3 (3P)
INITIALIZE STRING (3P)

**NAME**

INITIALIZE STROKE – initialize a stroke input device using 2D data

**SYNOPSIS**

**C Syntax**

**void**
**pinit_stroke ( ws, dev, init_view_ind, init_stroke, pet, echo_area, record )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **dev;** | *stroke device number* |
| **Pint** | **init_view_ind;** | *initial view indicator* |
| **Ppoint_list** | ∗**init_stroke;** | *initial stroke pointer* |
| **Pint** | **pet;** | *prompt and echo type* |
| **Plimit** | ∗**echo_area;** | *echo area pointer* |
| **Pstroke_data** | ∗**record;** | *data record pointer* |

**Required PHIGS Operating States**

(PHOP, WSOP, ∗, ∗)

**DESCRIPTION**

**Purpose**

Use INITIALIZE STROKE to set the initialization parameters of a STROKE device. This function stores these parameters in the workstation state list of the workstation associated with the specified device. The specified **device must be in REQUEST mode** when this function is called.

**C Input Parameters**

*ws* The workstation identifier of the workstation associated with the device.

*dev* The device number of the STROKE device to initialize. See the *AVAILABLE DEVICES* section below for a description of the available devices.

*init_view_ind*

The index of the view representation in the workstation's view table to use to map the specified initial positions from World Coordinates (WC) to Normalized Projection Coordinates (NPC).

*init_stroke*

A pointer to a Ppoint_list structure containing the initial stroke. The Ppoint_list structure is defined in phigs.h as follows:

```
typedef struct {
        Pint    num_points;
        Ppoint  *points;
} Ppoint_list;
```

*num_points* is the number of points in the initial stroke. This must be less than or equal to the buffer size specified in the input data record, *record*.

*points* is an array of Ppoint structures specifying the *x* and *y* WC of the initial stroke points. Ppoint is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          x;              /* x coordinate */
        Pfloat          y;              /* y coordinate */
} Ppoint;
```

*pet* The prompt/echo type desired. Those supported by each device are listed in the *AVAILABLE DEVICES* section below.

*echo_area*

A pointer to a Plimit structure defining the *x* and *y* components of the echo area, in Device Coordinates (DC). The *z* component in the workstation state list is left unchanged. Plimit is defined in

phigs.h as follows:

```
typedef struct {
        Pfloat          x_min;          /* minimum x coordinate value */
        Pfloat          x_max;          /* maximum x coordinate value */
        Pfloat          y_min;          /* minimum y coordinate value */
        Pfloat          y_max;          /* maximum y coordinate value */
} Plimit;
```

*record* A pointer to a Pstroke_data structure containing the data record information. The contents of the data record for each device and prompt/echo type are described below in the *AVAILABLE DEVICES* section. Pstroke_data is defined in phigs.h as:

Some fields of the record correspond to the prompt/echo type being used.

```
typedef struct {
        Pint            buffer_size;            /* input buffer size */
        Pint            init_pos;               /* initial editing position */
        Pfloat          x_interval;             /* x interval */
        Pfloat          y_interval;             /* y interval */
        Pfloat          time_interval;          /* time interval */
        union {
                struct {
                        Pint            unused;
                } pet_r1;
                struct {
                        Pint            unused;
                } pet_r2;
                struct {
                        Pmarker_attrs   marker_attrs;   /* marker attributes */
                } pet_r3;
                struct {
                        Pline_attrs     line_attrs;     /* line attributes */
                } pet_r4;
        } pets;
} Pstroke_data;
```

**Execution**

INITIALIZE STROKE sets the initialization parameters of a STROKE device. This function stores these parameters in the workstation state list of the workstation associated with the specified device. The specified device must be in REQUEST mode when this function is called.

The parameters that are initialized by this function are *initial stroke*, *prompt/echo type*, *echo volume*, and *input data record*.

The *initial stroke* is the logical input value the device will be set to whenever it is enabled. The device's measure will retain this value until operator input changes it. A device is enabled when the appropriate REQUEST function is called, or when its input mode is set to SAMPLE or EVENT.

A STROKE device measure consists of *points* and a *view index*. The *points* are World Coordinate (WC) points corresponding to the positions on the workstation selected by the operator. The *view index* is the index of the view representation used to transform the stroke positions from Normalized Projection Coordinates (NPC) to WC. This view representation is determined by selecting the highest priority representation that contains all the stroke positions within its NPC limits. See SET VIEW TRANSFORMATION INPUT PRIORITY for more information. The workstation transform is used to transform the operator-selected positions from Device Coordinates (DC) to NPC.

The initial stroke points are transformed to DC by applying the view orientation and view mapping transforms of the specified view representation, and then applying the workstation transformation. If the view index is invalid, an error is generated.

The *prompt/echo type* determines how the device will be presented to the operator and respond to his actions. Each device supports one or more prompt/echo types. Those supported by each device are listed in the device's description in the *AVAILABLE DEVICES* section below. All devices support prompt/echo type 1. Positive prompt/echo types are defined by the PHIGS standard. Negative types are implementation-dependent.

The *echo volume* defines the region of the display surface in which to echo the device. It is specified in DC. Devices that use the echo volume may restrict their display to this region. Some of these devices may still recognize operator input outside the region even though they do not display there. This function specifies only the *x* and *y* components of the echo volume. The existing *z* component in the workstation state list is left unchanged.

The *input data record* contains the prompt/echo type specific information that controls the device's appearance and characteristics. Not all of the data record contents are used by some devices. The device descriptions in the *AVAILABLE DEVICES* section below list the data record contents that each device recognizes.

Stroke devices contain a buffer in which to store the points selected by the operator. Points are added to the buffer starting at the edit position. Both the buffer size and the edit position are specified in the input data record.

All the initialization parameters must be specified properly, or this function will generate an error. The *ERRORS* section below lists the possible error conditions.

The default initialization parameters and the list of prompt/echo types supported by a STROKE input device can be inquired with the function INQUIRE DEFAULT STROKE DEVICE DATA. The current state of the device can be inquired with the function INQUIRE STROKE DEVICE STATE.

## AVAILABLE DEVICES

### Device 1 - Pointer device

This device is associated with the X pointer device. The stroke points in the STROKE device's measure are the WC points corresponding to the positions specified in an operator-generated series of X pointer events. The operator adds points to the buffer by positioning the cursor to the desired location and depressing the left pointer button. Once all the desired points are selected, the operator presses both the shift key and the left mouse button at the same time. This triggers the stroke. Points can be removed from the buffer by pressing the CONTROL key and pointer button simultaneously. Only points after the *edit position* can be removed in this way.

**Prompt/echo types supported**: 1

PET 1 — Display the stroke points by drawing a marker at each stroke position. The marker used is of marker type 3 (asterisk), size 1.0, and uses colour index 1.

This device uses the *buffer_size* and *init_pos* fields of the stroke data record, Pstroke_data.

## ERRORS

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

251 Ignoring function, the function requires the input device to be in REQUEST mode

253 Warning, the specified prompt/echo type is not available on the specified workstation. Prompt/echo type one will be used in its place

254 Ignoring function, invalid echo area/volume; XMIN ≥ XMAX, YMIN ≥ YMAX, or ZMIN > ZMAX

255 Ignoring function, one of the echo area/volume boundary points is outside the range of the device

260 Ignoring function, one of the fields within the input device data record is in error

253 Warning, the specified prompt/echo type is not available on the specified workstation. Prompt/echo type one will be used in its place

261 Ignoring function, initial value is invalid

262 Ignoring function, number of points in the initial stroke is greater than the buffer size

114 Ignoring function, the view index value is less than zero

**SEE ALSO**

SET STROKE MODE (3P)
REQUEST STROKE (3P)
SAMPLE STROKE (3P)
GET STROKE (3P)
INQUIRE STROKE DEVICE STATE (3P)
SET VIEW TRANSFORMATION INPUT PRIORITY (3P)
INITIALIZE STROKE 3 (3P)

**NAME**

INITIALIZE STROKE 3 – initialize a stroke input device using 3D data

**SYNOPSIS**

**C Syntax**

**void**
**pinit_stroke3 ( ws, dev, init_view_ind, init_stroke, pet, echo_volume, rec )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **dev;** | *stroke device number* |
| **Pint** | **init_view_ind;** | *initial view indicator* |
| **Ppoint_list3** | ***init_stroke;** | *initial stroke pointer* |
| **Pint** | **pet;** | *prompt and echo type* |
| **Plimit3** | ***echo_volume;** | *echo volume pointer* |
| **Pstroke_data3** | ***rec;** | *data record pointer* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INITIALIZE STROKE 3 to set the initialization parameters of a STROKE device. This function stores these parameters in the workstation state list of the workstation associated with the specified device. The specified **device must be in REQUEST mode** when this function is called.

**C Input Parameters**

*ws* The workstation identifier of the workstation associated with the device.

*dev* The device number of the STROKE device to initialize. See the *AVAILABLE DEVICES* section below for a description of the available devices.

*init_view_ind*

The index of the view representation in the workstation's view table to use to map the specified initial positions from World Coordinates (WC) to Normalized Projection Coordinates (NPC).

*init_stroke*

A pointer to a Ppoint_list3 structure containing the initial stroke. The Ppoint_list3 structure is defined in phigs.h as follows:

```
typedef struct {
        Pint        num_points;
        Ppoint3     *points;
} Ppoint_list3;
```

*num_points* is the number of points in the initial stroke. This must be less than or equal to the buffer size specified in the input data record, *rec*.

*points* is an array of Ppoint3 structures specifying the *x*, *y*, and *z* WC of the initial stroke points. Ppoint3 is defined in phigs.h as follows:

```
typedef struct {
        Pfloat      x;      /* x coordinate */
        Pfloat      y;      /* y coordinate */
        Pfloat      z;      /* z coordinate */
} Ppoint3;
```

*pet* The prompt/echo type desired. Those supported by each device are listed in the *AVAILABLE DEVICES* section below.

*echo_volume*

A pointer to a Plimit3 structure defining the *x*, *y*, and *z* components of the echo volume, in Device

Coordinates (DC). Plimit3 is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          x_min;          /* minimum x coordinate value */
        Pfloat          x_max;          /* maximum x coordinate value */
        Pfloat          y_min;          /* minimum y coordinate value */
        Pfloat          y_max;          /* maximum y coordinate value */
        Pfloat          z_min;          /* minimum z coordinate value */
        Pfloat          z_max;          /* maximum z coordinate value */
} Plimit3;
```

*rec* A pointer to a Pstroke_data3 structure containing the data record information. The contents of the data record for each device and prompt/echo type are described below in the *AVAILABLE DEVICES* section.

Some members of the data record correspond to the prompt/echo type being used.

Pstroke_data3 is defined in phigs.h as follows:

```
typedef struct {
        Pint            buffer_size;            /* input buffer size */
        Pint            init_pos;               /* initial editing position */
        Pfloat          x_interval;             /* x interval */
        Pfloat          y_interval;             /* y interval */
        Pfloat          z_interval;             /* z interval */
        Pfloat          time_interval;          /* time interval */
        union {
                struct {
                        Pint            unused;
                } pet_r1;
                struct {
                        Pint            unused;
                } pet_r2;
                struct {
                        Pmarker_attrs   marker_attrs;   /* marker attributes */
                } pet_r3;
                struct {
                        Pline_attrs     line_attrs;     /* line attributes */
                } pet_r4;
        } pets;
} Pstroke_data3;
```

**Execution**

INITIALIZE STROKE 3 sets the initialization parameters of a STROKE device. This function stores these parameters in the workstation state list of the workstation associated with the specified device. The specified device must be in REQUEST mode when this function is called.

The parameters that are initialized by this function are *initial stroke*, *prompt/echo type*, *echo volume*, and *input data record*.

The *initial stroke* is the logical input value the device will be set to whenever it is enabled. The device's measure will retain this value until operator input changes it. A device is enabled when the appropriate REQUEST function is called, or when its input mode is set to SAMPLE or EVENT.

A STROKE device measure consists of *points* and a *view index*. The *points* are World Coordinate (WC) points corresponding to the positions on the workstation selected by the operator. The *view index* is the index of the view representation used to transform the stroke positions from Normalized Projection Coordinates (NPC) to WC. This view representation is determined by selecting the highest priority representation

that contains all the stroke positions within its NPC limits. See SET VIEW TRANSFORMATION INPUT PRIORITY for more information. The workstation transform is used to transform the operator-selected positions from Device Coordinates (DC) to NPC.

The initial stroke points are transformed to DC by applying the view orientation and view mapping transforms of the specified view representation, and then applying the workstation transformation. If the view index is invalid, an error is generated.

The *prompt/echo type* determines how the device will be presented to the operator and respond to his actions. Each device supports one or more prompt/echo types. Those supported by each device are listed in the device's description in the *AVAILABLE DEVICES* section below. All devices support prompt/echo type 1. Positive prompt/echo types are defined by the PHIGS standard. Negative types are implementation-dependent.

The *echo volume* defines the region of the display surface in which to echo the device. It is specified in Device Coordinates (DC). Devices that use the echo volume may restrict their display to this region. Some of these devices may still recognize operator input outside the region even though they do not display there.

The *input data record* contains the prompt/echo type specific information that controls the device's appearance and characteristics. Not all of the data record contents are used by some devices. The device descriptions in the *AVAILABLE DEVICES* section below list the data record contents that each device recognizes.

Stroke devices contain a buffer in which to store the points selected by the operator. Points are added to the buffer starting at the edit position. Both the buffer size and the edit position are specified in the input data record.

All the initialization parameters must be specified properly, or this function will generate an error. The *ERRORS* section below lists the possible error conditions.

The default initialization parameters and the list of prompt/echo types supported by a STROKE input device can be inquired with the function INQUIRE DEFAULT STROKE DEVICE DATA 3. The current state of the device can be inquired with the function INQUIRE STROKE DEVICE STATE 3.

## AVAILABLE DEVICES

### Device 1 - Pointer device

This device is associated with the X pointer device. The stroke points in the STROKE device's measure are the WC points corresponding to the positions specified in an operator-generated series of X pointer events. The operator adds points to the buffer by positioning the cursor to the desired location and depressing the left pointer button. Once all the desired points are selected, the operator presses both the shift key and the left mouse button at the same time. This triggers the stroke. Points can be removed from the buffer by pressing the CONTROL key and pointer button simultaneously. Only points after the *edit position* can be removed in this way.

**Prompt/echo types supported**: 1

PET 1 — Display the stroke points by drawing a marker at each stroke position. The marker used is of marker type 3 (asterisk), size 1.0, and uses colour index 1.

This device uses the *buffer_size* and *init_pos* fields of the stroke data record, Pstroke_data.

## ERRORS

003 — Ignoring function, function requires state (PHOP, WSOP, *, *)

054 — Ignoring function, the specified workstation is not open

061 — Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 — Ignoring function, the specified device is not available on the specified workstation

251 — Ignoring function, the function requires the input device to be in REQUEST mode

253 — Warning, the specified prompt/echo type is not available on the specified workstation. Prompt/echo type one will be used in its place

254 Ignoring function, invalid echo area/volume; XMIN ≥ XMAX, YMIN ≥ YMAX, or ZMIN > ZMAX

255 Ignoring function, one of the echo area/volume boundary points is outside the range of the device

260 Ignoring function, one of the fields within the input device data record is in error

253 Warning, the specified prompt/echo type is not available on the specified workstation. Prompt/echo type one will be used in its place

261 Ignoring function, initial value is invalid

262 Ignoring function, number of points in the initial stroke is greater than the buffer size

114 Ignoring function, the view index value is less than zero

**SEE ALSO**

SET STROKE MODE (3P)
REQUEST STROKE 3 (3P)
SAMPLE STROKE 3 (3P)
GET STROKE 3 (3P)
INQUIRE STROKE DEVICE STATE 3 (3P)
SET VIEW TRANSFORMATION INPUT PRIORITY (3P)
INITIALIZE STROKE (3P)

**NAME**

INITIALIZE VALUATOR – initialize a valuator input device using 2D data

**SYNOPSIS**

**C Syntax**

```
void
pinit_val ( ws, dev, init, pet, echo_area, record )
Pint        ws;          workstation identifier
Pint        dev;         valuator device number
Pfloat      init;        initial value
Pint        pet;         prompt and echo type
Plimit      *echo_area;  echo area pointer
Pval_data   *record;     data record pointer
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INITIALIZE VALUATOR to set the initialization parameters of a VALUATOR device. This function stores these parameters in the workstation state list of the workstation associated with the specified device. The specified **device must be in REQUEST mode** when this function is called.

**C Input Parameters**

*ws* The workstation identifier of the workstation associated with the device.

*dev* The device number of the VALUATOR device to initialize. See the *AVAILABLE DEVICES* section below for a description of the available devices.

*init* The initial valuator value.

*pet* The prompt/echo type desired. Those supported by each device are listed in the *AVAILABLE DEVICES* section below.

*echo_area*

A pointer to a Plimit structure defining the *x* and *y* components of the echo volume, in Device Coordinates (DC). The existing *z* component in the workstation state list is left unchanged. Plimit is defined in phigs.h as follows:

```
typedef struct {
        Pfloat      x_min;      /* minimum x coordinate value */
        Pfloat      x_max;      /* maximum x coordinate value */
        Pfloat      y_min;      /* minimum y coordinate value */
        Pfloat      y_max;      /* maximum y coordinate value */
} Plimit;
```

*record* A pointer to a Pval_data structure containing the data record information. The contents of the data record for each device and prompt/echo type are described below in the *Available Devices* section.

Some members of the data record correspond to the prompt/echo type being used. As an example, the appropriate member of the Pval_data structure for prompt/echo type 1 is *pet_r1*.

For some prompt/echo types the data record is not used. The *record* parameter, however, must still be passed.

Pval_data is defined in phigs.h as:

```
typedef struct {
        Pfloat                  low;        /* low range limit */
        Pfloat                  high;       /* high range limit */
        union {
          struct {
              Pint              unused;
          } pet_r1;
          struct {
              Pchar             *label;           /* the device label widget string */
              Pchar             *format;          /* the current value widget format string */
              Pchar             *low_label;       /* the low value widget format string */
              Pchar             *high_label;      /* the low value widget format string */
          } pet_u1;
        } pets;
} Pval_data;
```

**Execution**

INITIALIZE VALUATOR sets the initialization parameters of a VALUATOR device. This function stores these parameters in the workstation state list of the workstation associated with the specified device. The specified device must be in REQUEST mode when this function is called.

The parameters that are initialized by this function are *initial value*, *prompt/echo type*, *echo volume*, and *input data record*.

The *initial value* is the logical input value the device will take on whenever it is enabled. The device's measure will retain this value until operator input changes it. A device is enabled when the appropriate REQUEST function is called, or when its input mode is set to SAMPLE or EVENT.

A VALUATOR device measure consists of a floating point number. This number is between the device's low value and high value, inclusive.

The *prompt/echo type* determines how the device will be presented to the operator and respond to his actions. Each device supports one or more prompt/echo types. Those supported by each device are listed in the device's description in the *AVAILABLE DEVICES* section below. All devices support prompt/echo type 1. Positive prompt/echo types are defined by the PHIGS standard. Negative types are implementation-dependent.

The *echo volume* defines the region of the display surface in which to echo the device. It is specified in DC. Devices that use the echo volume may restrict their display to this region. Some of these devices may still recognize operator input outside the region even though they do not display there. This function specifies only the *x* and *y* components of the echo volume. The existing *z* component in the workstation state list is left unchanged.

The *input data record* contains the prompt/echo type specific information that controls the device's appearance and characteristics. Not all of the data record contents are used by some devices. The device descriptions in the *AVAILABLE DEVICES* section below list the data record contents that each device recognizes.

All the initialization parameters must be specified properly, or this function will generate an error. The *ERRORS* section below lists the possible error conditions.

The default initialization parameters and the list of prompt/echo types supported by a VALUATOR input device can be inquired with the function INQUIRE DEFAULT VALUATOR DEVICE DATA. The current state of the device can be inquired with the function INQUIRE VALUATOR DEVICE STATE.

## AVAILABLE DEVICES

### Device 1 - Athena Scrollbar Widget

The valuator device realization is a collection of widgets from the Athena widget set. It consists of a Shell widget, a Pane widget, 4 Label widgets and a Scrollbar widget. The Scrollbar is the item manipulated by the operator to change the valuator value. The four Label widgets display a label for the device and labels

for each of the current value, low value and high value of the device. The device label, current value and low value are all displayed (by default) to the left of the Scrollbar. The high value is displayed to the right. The Pane, Label, and Scrollbar widgets are children of the Shell widget.

Each of the four Label widgets has a character string associated with it that indicates what to display in that widget. The strings for the current value, low value and high value widgets may contain printf(3) format descriptors. PEX-SI will expand these descriptors with the corresponding device value so that the value appears in the widget. For example, if the low value widget string is ‘‘the low value is %f’’ and the low value is 5, PEX-SI will display the widget as ‘‘the low value is 5.0’’.

The echo volume is not used for this device.

**Prompt/echo types supported**: 1, -1

PET 1 Display the valuator item as specified above. The string for the label widget is ‘‘value.’’ The strings for the current value, low value, and high value strings are all *%8.3g*. (Use PET -1 if these values are not as desired.) No data record is used with PET 1.

PET -1 Display the valuator item as specified for PET 1 but with application-specified widget strings. The label, low value, and high value strings are specified in the data record.

The widgets used and their names, in decreasing hierarchy, are:

valuator1 → *applicationShellWidgetClass*
pane → *panedWidgetClass*
label → *labelWidgetClass*
readout → *labelWidgetClass*
low_label → *labelWidgetClass*
scrollbar → *scrollbarWidgetClass*
high_label → *labelWidgetClass*

Fallback resources for valuator devices are specified in the API. They are:

∗valuator1.pane.orientation → *horizontal*

∗valuator1∗scrollbar.orientation → *horizontal*

∗valuator1∗scrollbar.length → *50*

∗valuator1∗scrollbar.translations —
<Btn1Down> → *StartScroll(Continuous) MoveThumb() NotifyThumb()*
<Btn1Motion> → *MoveThumb() NotifyThumb()*
<Btn1Up> → *NotifyScroll(Proportional) RequestSatisfied() EndScroll()*

∗valuator1∗scrollbar.skipAdjust → *FALSE*

∗valuator1∗skipAdjust → *TRUE*

∗valuator1∗showGrip → *FALSE*

The fully qualified name of all widgets is <appl_name>.workstation<ws_id>.valuator<dev_id>.<widget_name>, where <appl_name> is the application name specified in the call to OPEN XPHIGS. (This will be "phigs" if OPEN XPHIGS was not called.) For example, phigs.workstation1.valuator1.pane is the name of the pane widget of valuator device 1 on workstation 1.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, ∗, ∗)

054 Ignoring function, the specified workstation is not open

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

251 Ignoring function, the function requires the input device to be in REQUEST mode

253 Warning, the specified prompt/echo type is not available on the specified workstation. Prompt/echo type one will be used in its place

254 Ignoring function, invalid echo area/volume; XMIN ≥ XMAX, YMIN ≥ YMAX, or ZMIN > ZMAX

255 Ignoring function, one of the echo area/volume boundary points is outside the range of the device

260 Ignoring function, one of the fields within the input device data record is in error

261 Ignoring function, initial value is invalid

**SEE ALSO**

SET VALUATOR MODE (3P)
REQUEST VALUATOR (3P)
SAMPLE VALUATOR (3P)
GET VALUATOR (3P)
INQUIRE VALUATOR DEVICE STATE (3P)
INITIALIZE VALUATOR 3 (3P)

**NAME**

INITIALIZE VALUATOR 3 – initialize a valuator input device using 3D data

**SYNOPSIS**

**C Syntax**

```
void
pinit_val3 ( ws, dev, init, pet, echo_volume, record )
Pint          ws;          workstation identifier
Pint          dev;         valuator device number
Pfloat        init;        initial value
Pint          pet;         prompt and echo type
Plimit3       *echo_volume; echo volume pointer
Pval_data3    *record;     data record pointer
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INITIALIZE VALUATOR 3 to set the initialization parameters of a VALUATOR device. This function stores these parameters in the workstation state list of the workstation associated with the specified device. **The specified device must be in REQUEST mode when this function is called.**

**C Input Parameters**

*ws* The workstation identifier of the workstation associated with the device.

*dev* The device number of the VALUATOR device to initialize. See the *AVAILABLE DEVICES* section below for a description of the available devices.

*init* The initial valuator value.

*pet* The prompt/echo type desired. Those supported by each device are listed in the *AVAILABLE DEVICES* section below.

*echo_volume*

A pointer to a Plimit3 structure defining the *x*, *y*, and *z* components of the echo volume, in Device Coordinates (DC). Plimit3 is defined in phigs.h as follows:

```
typedef struct {
        Pfloat      x_min;      /* minimum x coordinate value */
        Pfloat      x_max;      /* maximum x coordinate value */
        Pfloat      y_min;      /* minimum y coordinate value */
        Pfloat      y_max;      /* maximum y coordinate value */
        Pfloat      z_min;      /* minimum z coordinate value */
        Pfloat      z_max;      /* maximum z coordinate value */
} Plimit3;
```

*record* A pointer to a Pval_data3 structure containing the data record information. The contents of the data record for each device and prompt/echo type are described below in the *AVAILABLE DEVICES* section.

Some members of the data record correspond to the prompt/echo type being used. As an example, the appropriate member of the Pval_data3 structure for prompt/echo type 1 is *pet_r1*.

For some prompt/echo types the data record is not used. The *record* parameter, however, must still be passed.

Pval_data3 is defined in phigs.h as:

```
typedef struct {
        Pfloat          low;            /* low range limit */
        Pfloat          high;           /* high range limit */
        union {
          struct {
              Pint      unused;
          } pet_r1;
          struct {
              Pchar     *label;         /* the device label widget string */
              Pchar     *format;        /* the current value widget format string */
              Pchar     *low_label;     /* the low value label widget format string */
              Pchar     *high_label;    /* the high value label widget format string */
          } pet_u1;
        } pets;
} Pval_data3;
```

**Execution**

INITIALIZE VALUATOR 3 sets the initialization parameters of a VALUATOR device. This function stores these parameters in the workstation state list of the workstation associated with the specified device. The specified device must be in REQUEST mode when this function is called.

The parameters that are initialized by this function are the *initial value*, *prompt/echo type*, *echo volume*, and *input data record*.

The *initial value* is the logical input value the device will take on whenever it is enabled. The device's measure retains this value until operator input changes it. A device is enabled when the appropriate REQUEST function is called, or when its input mode is set to SAMPLE or EVENT.

A VALUATOR device measure consists of a floating point number. This number is between the device's low value and high value, inclusive.

The *prompt/echo type* determines how the device will be presented to the operator and respond to his actions. Each device supports one or more prompt/echo types. Those supported by each device are listed in the device's description in the *AVAILABLE DEVICES* section below. All devices support prompt/echo type 1. Positive prompt/echo types are defined by the PHIGS standard. Negative types are implementation-dependent.

The *echo volume* defines the region of the display surface in which to echo the device. It is specified in DC. Devices that use the echo volume may restrict their display to this region. Some of these devices may still recognize operator input outside the region even though they do not display there.

The *input data record* contains the prompt/echo type specific information that controls the device's appearance and characteristics. Not all the data record contents are used by some devices. The device descriptions in the *AVAILABLE DEVICES* section below list the data record contents that each device recognizes.

All the initialization parameters must be specified properly, or this function will generate an error. The *ERRORS* section below lists the possible error conditions.

The default initialization parameters and the list of prompt/echo types supported by a VALUATOR input device can be inquired with the function INQUIRE DEFAULT VALUATOR DEVICE DATA 3. The current state of the device can be inquired with the function INQUIRE VALUATOR DEVICE STATE 3.

**AVAILABLE DEVICES**

**Device 1 - Athena Scrollbar Widget**

The valuator device realization is a collection of widgets from the Athena widget set. It consists of a Shell widget, a Pane widget, 4 Label widgets and a Scrollbar widget. The Scrollbar is the item manipulated by the operator to change the valuator value. The four Label widgets display a label for the device and labels for each of the current value, low value and high value of the device. The device label, current value and low value are all displayed (by default) to the left of the Scrollbar. The high value is displayed to the right.

The Pane, Label, and Scrollbar widgets are children of the Shell widget.

Each of the four Label widgets has a character string associated with it that indicates what to display in that widget. The strings for the current value, low value and high value widgets may contain printf(3) format descriptors. PEX-SI will expand these descriptors with the corresponding device value so that the value appears in the widget. For example, if the low value widget string is ‘‘the low value is %f’’ and the low value is 5, PEX-SI will display the widget as ‘‘the low value is 5.0’’.

The echo volume is not used for this device.

**Prompt/echo types supported**: 1, -1

PET 1 Display the valuator item as specified above. The string for the label widget is ‘‘value’’. The strings for the current value, low value and high value widgets are all *%8.3g*. (Use PET -1 if you want to specify your own strings.) No data record is ued with PET 1.

PET -1 Display the valuator item as specified for PET 1 but with application-specified widget strings. The label, current value, low value and high value widget strings are specified in the data record.

The widgets used and their names, in decreasing hierarchy, are:

valuator001 → *applicationShellWidgetClass*
pane → *panedWidgetClass*
label → *labelWidgetClass*
readout → *labelWidgetClass*
low_label → *labelWidgetClass*
scrollbar → *scrollbarWidgetClass*
high_label → *labelWidgetClass*

Fallback resources for valuator devices are specified in the API. They are:

∗valuator001.pane.orientation → *horizontal*

∗valuator001∗scrollbar.orientation → *horizontal*

∗valuator001∗scrollbar.length → *50*

∗valuator001∗scrollbar.translations —
<Btn1Down> → *StartScroll(Continuous) MoveThumb() NotifyThumb()*
<Btn1Motion> → *MoveThumb() NotifyThumb()*
<Btn1Up> → *NotifyScroll(Proportional) RequestSatisfied() EndScroll()*

∗valuator001∗scrollbar.skipAdjust → *FALSE*

∗valuator001∗skipAdjust → *TRUE*

∗valuator001∗showGrip → *FALSE*

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, ∗, ∗)

054 Ignoring function, the specified workstation is not open

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

251 Ignoring function, the function requires the input device to be in REQUEST mode

253 Warning, the specified prompt/echo type is not available on the specified workstation. Prompt/echo type one will be used in its place

254 Ignoring function, invalid echo area/volume; XMIN ≥ XMAX, YMIN ≥ YMAX, or ZMIN > ZMAX

255 Ignoring function, one of the echo area/volume boundary points is outside the range of the device

260 Ignoring function, one of the fields within the input device data record is in error

261 Ignoring function, initial value is invalid

**SEE ALSO**

SET VALUATOR MODE (3P)
REQUEST VALUATOR (3P)
SAMPLE VALUATOR (3P)
GET VALUATOR (3P)
INQUIRE VALUATOR DEVICE STATE 3 (3P)
INITIALIZE VALUATOR (3P)

**NAME**

INQUIRE ALL CONFLICTING STRUCTURES – inquire all conflicting structures in the Central Structure Store and an archive file

**SYNOPSIS**

**C Syntax**

**void**
**pinq_all_conf_structs ( ar_id, length, start, error_ind, ids, total_length )**

| | | |
|---|---|---|
| **Pint** | **ar_id;** | *archive identifier* |
| **Pint** | **length;** | *length of application list* |
| **Pint** | **start;** | *starting position* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pint_list** | ***ids;** | *OUT list of conflicting structure ids* |
| **Pint** | ***total_length;** | *OUT length of list in PHIGS* |

**Required PHIGS Operating States**

(PHOP, *, *, AROP)

**DESCRIPTION**

**Purpose**

INQUIRE ALL CONFLICTING STRUCTURES obtains a list of the identifiers of all structures which exist in both the Central Structure Store and the specified open archive file.

**C Input Parameters**

*ar_id* The archive identifier specifying the open archive file to use.

*length* The number of *ints* items in the *ids* output parameter for which the application has allocated memory. *length* is the number of list elements (structure identifiers) that the system can return in *ints*. If a value of 0 is used here, no data will be returned in the *ints* list, but the total number of conflicting structures will be returned in *total_length*.

*start* Starting position in the list of identifiers of conflicting structures at which to begin the inquiry. The elements of the list of structure identifiers, beginning with the item number specified by *start*, are copied sequentially into *ints* until *ints* is full or all the structure identifiers have been copied.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*ids* A pointer to a Pint_list structure in which the list of identifiers of conflicting structures is returned. The Pint_list structure is defined in phigs.h as follows:

```
typedef struct {
        Pint    num_ints;       /* number of Pints in list */
        Pint    *ints;          /* list of integers */
} Pint_list;
```

The *num_ints* component specifies the number of structure identifiers in the list. The *ints* component is a pointer to a list, *num_ints* long, of the structure identifiers.

The application must allocate memory for *length* elements in the *ints* list.

*total_length*

A pointer to an integer in which the total number of conflicting structures is returned. This is the value required for *length* if all structure identifiers are to be returned.

**ERRORS**

007 Ignoring function, function requires state (PHOP, *, *, AROP)

404 Ignoring function, the specified archive file is not open

**SEE ALSO**

INQUIRE CONFLICTING STRUCTURES IN NETWORK (3P)

**NAME**

INQUIRE ANNOTATION FACILITIES – inquire annotation text facilities on workstation type

**SYNOPSIS**

**C Syntax**

**void**
**pinq_anno_facs ( type, length, start, error_ind, styles, total_length, num_char_hts, min_char_ht, max_char_ht )**

| | | |
|---|---|---|
| **Pint** | **type;** | *workstation type* |
| **Pint** | **length;** | *length of application list* |
| **Pint** | **start;** | *starting position* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pint_list** | ***styles;** | *OUT list annotation styles* |
| **Pint** | ***total_length;** | *OUT length of list in PHIGS* |
| **Pint** | ***num_char_hts;** | *OUT number of character heights* |
| **Pfloat** | ***min_char_ht;** | *OUT minimum character height* |
| **Pfloat** | ***max_char_ht;** | *OUT maximum character height* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

INQUIRE ANNOTATION FACILITIES obtains a list of the annotation text facilities available on the specified workstation type.

**C Input Parameters**

*type* Workstation type.

*length* The number of *ints* items in the *styles* output parameter for which the application has allocated memory. The *length* is the number of list elements (annotation styles) that the system can return in the list of integers in the Pint_list structure under *styles*. If a value of zero is used here, no data will be returned in this list, but the total number of annotation styles available on the specified workstation type will be returned in *total_length*.

*start* Starting position in the workstation type list of available annotation styles at which the inquiry is to begin. The elements of the list of annotation styles, beginning with the item number specified by *start*, are copied sequentially into the list of integers until it is full or all the annotation styles have been copied.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*styles* A pointer to a Pint_list structure in which the system returns the list of annotation styles available on the specified type of workstation. Pint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint    num_ints;       /* number of Pints in list */
        Pint    *ints;          /* list of integers */
} Pint_list;
```

The *num_ints* component specifies the number of annotation styles in the list. The *ints* component is a pointer to a list, *num_ints* long, of the annotation styles.

The application must allocate memory for *length* elements in the list of *ints*.

The defined values for the annotation styles are:

1 PANNO_STYLE_UNCONNECTED

2 PANNO_STYLE_LEAD_LINE

*total_length*

A pointer to an integer in which the total number of elements in the specified workstation type list of annotation styles is returned. This is the value required for *length* if all annotation styles are to be returned.

*num_char_hts*

A pointer to an integer in which the number of character heights is returned.

*min_char_ht*

A pointer to an integer in which the minimum character height is returned.

*max_char_ht*

A pointer to an integer in which the maximum character height is returned.

**ERRORS**

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

052 Ignoring function, workstation type not recognized by the implementation

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

059 Ignoring function, the specified workstation does not have output capability (i.e., the workstation category is neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

**SEE ALSO**

ANNOTATION TEXT RELATIVE (3P)
ANNOTATION TEXT RELATIVE 3 (3P)
PHIGS WORKSTATION DESCRIPTION TABLE (7P)

**NAME**

INQUIRE ARCHIVE FILES – inquire the currently open archive file identifiers and names

**SYNOPSIS**

**C Syntax**

```
void
pinq_ar_files ( store, error_ind, ar_files )
Pstore          store;          handle to Store object
Pint            *error_ind;     OUT error indicator
Par_file_list   **ar_files;     OUT list of archive file
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE ARCHIVE FILES to determine the names and identifiers of all currently open archive files.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing data associated with the device state. This buffer is passed as the *store* argument.

The store buffer is a data area managed by PHIGS. While the application is responsible for creating the inital buffer through a call to STORE, PHIGS manages this area such that there is sufficient memory for the specific inquiry. The data record within the store buffer is accessed by the pointer pointed to by *ar_files*.

*store* The memory buffer PHIGS is to use for storing the information returned. This buffer must exist prior to calling this function (see CREATE STORE (3P)).

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*ar_files* A pointer to a structure Par_file_list, containing a list of the currently open archive file identifiers and names. Par_file_list is defined in phigs.h as follows:

```
typedef struct{
        Pint            num_ar_files;   /* number of archive files */
        Par_file        *ar_files;      /* list of archive files */
} Par_file_list;
```

The *num_ar_files* component specifies the number of currently open archive files. The *ar_files* component is a pointer to a list, *num_ar_files* long, of Par_file structs listing the identifiers and names of currently open archive files. Par_file is defined in phigs.h as follows:

```
typedef struct{
        Pint            id;             /* archive file identifier */
        char            *name;          /* archive file name */
} Par_file;
```

The *id* component is the archive file identifier and the *name* component is a pointer to a character string giving the file name of the archive file with identifier *id*.

**ERRORS**

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

**SEE ALSO**

OPEN ARCHIVE FILE (3P)
CLOSE ARCHIVE FILE (3P)
INQUIRE ARCHIVE STATE VALUE (3P)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

INQUIRE ARCHIVE STATE VALUE – inquire state value of archive file

**SYNOPSIS**

**C Syntax**

```
void
pinq_ar_st ( archive_state )
Par_st          *archive_state;          OUT archive state
```

**Required PHIGS Operating States**

(*, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE ARCHIVE STATE VALUE to determine the current state of the archive file. The state value may be either *archive open* (AROP ) or *archive closed* (ARCL).

**C Output Parameter**

*archive_state*

A pointer to a Par_st enumerated variable in which the current state of the archive file is returned. Archive state values are defined in phigs.h as follows:

| | |
|---|---|
| PST_ARCL | (archive closed) |
| PST_AROP | (archive open) |

**ERRORS**

No Error

**SEE ALSO**

OPEN ARCHIVE FILE (3P)
CLOSE ARCHIVE FILE (3P)
INQUIRE ARCHIVE FILES (3P)

**NAME**

INQUIRE CHOICE DEVICE STATE – inquire state of a choice device

**SYNOPSIS**

**C Syntax**

**void**
**pinq_choice_st ( ws, dev, store, err, op_mode, echo_switch, init_status, init_choice, prompt_echo, echo_area, choice_data )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **dev;** | *choice device number* |
| **Pstore** | **store;** | *OUT handle to Store object* |
| **Pint** | ***err;** | *OUT error indicator* |
| **Pop_mode** | ***op_mode;** | *OUT operating mode* |
| **Pecho_switch** | ***echo_switch;** | *OUT echo switch* |
| **Pin_status** | ***init_status;** | *OUT initial choice status* |
| **Pint** | ***init_choice;** | *OUT initial choice* |
| **Pint** | ***prompt_echo;** | *OUT prompt/echo type* |
| **Plimit** | ***echo_area;** | *OUT echo area* |
| **Pchoice_data** | ****choice_data;** | *OUT data record* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE CHOICE DEVICE STATE to determine the current state of a choice device.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing data associated with the device state. This buffer is passed as the *store* argument.

The store buffer is a data area managed by PHIGS. While the application is responsible for creating the initial buffer through a call to CREATE STORE, PHIGS manages this area such that there is sufficient memory for the specific inquiry. The choice device data record within the store buffer is accessed via the pointer pointed to by choice_data.

*ws* Workstation identifier. An integer specifying the workstation with which the specified choice device is associated.

*dev* The device number of the choice device. See the *Available Devices* section of INITIALIZE CHOICE for a description of the available devices.

*store* The memory buffer PHIGS is to use for storing the information returned. This buffer must exist prior to calling this function (see CREATE STORE (3P)).

**C Output Parameters**

*err* The error indicator. See the *Execution* section below for a description of its use. See the *Errors* section below for the possible values it may return.

*op_mode*

A pointer to a variable of type Pop_mode, which contains the current operating mode of the device. Pop_mode is enumerated in phigs.h as follows:

```
typedef enum {
        POP_REQ,
        POP_SAMPLE,
        POP_EVENT
} Pop_mode;
```

*echo_switch*

A pointer to a variable of type Pecho_switch, which contains the state of the device's echo switch. The value returned for *echo_switch* will be either PSWITCH_ECHO or PSWITCH_NO_ECHO.

*init_status*

A pointer to a variable of type Pin_status, which contains the initial choice status of the device. Possible values for *init_status* are PIN_STATUS_OK or PIN_STATUS_NONE.

*init_choice*

A pointer to an integer that contains the value of the initial choice.

*prompt_echo*

A pointer to an integer that contains the value of the prompt/echo type.

*echo_area*

A pointer to a variable of type Plimit that contains the echo area of the device. Plimit is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          x_min;          /* x min */
        Pfloat          x_max;          /* x max */
        Pfloat          y_min;          /* y min */
        Pfloat          y_max;          /* y max */
} Plimit;
```

*choice_data*

A pointer to a Pchoice_data pointer. PHIGS assigns this pointer to the location in the Pstore structure that contains the device's data record contents. Pchoice_data is defined in phigs.h as follows:

```
typedef struct {
    union {
        struct {
                Pint            unused;
        } pet_r1;
        struct {
                Pint            num_prompts;    /* number of alternatives */
                Ppr_switch      *prompts;       /* array of prompts */
        } pet_r2;
        struct {
                Pint            num_strings;    /* number of choice strings */
                char            **strings;      /* array of choice strings */
        } pet_r3;
        struct {
                Pint            num_strings;    /* number of alternatives */
                char            **strings;      /* array of strings  */
        } pet_r4;
        struct {
                Pint            struct_id;      /* struct identifier  */
                Pint            num_pick_ids;   /* number of alternatives */
                Pint            *pick_ids;      /* array of pick identifiers */
        } pet_r5;
    } pets;
} Pchoice_data;
```

Ppr_switch is an enumerated type with the following values:

```
typedef enum {
        PPR_OFF
        PPR_ON
```

```
} Ppr_switch;
```

**Execution**

INQUIRE CHOICE DEVICE STATE returns the current state of the specified choice device, which is stored in the workstation state list of the workstation associated with the device. The current state includes the operating mode, echo switch, initial choice, prompt/echo type, echo area, and data record. See SET CHOICE MODE for a description of the operating mode and the echo switch, and how to set these values. See INITIALIZE CHOICE for a description of the initial choice, prompt/echo type, echo area, and data record contents, and how to set these values.

In the event of an error, the *error indicator* indicates the error number of the error detected, and no other output data is returned. If no error is detected, the *error indicator* is set to zero, and the inquired information will be available in the output parameters. Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation Buffer overflow in input or inquiry function

**SEE ALSO**

SET CHOICE MODE (3P)
INITIALIZE CHOICE (3P)
INQUIRE CHOICE DEVICE STATE 3 (3P)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

INQUIRE CHOICE DEVICE STATE 3 – inquire state of a choice device

**SYNOPSIS**

**C Syntax**

**void**
**pinq_choice_st3 ( ws, dev, store, err, op_mode, echo_switch, init_status, init_choice, prompt_echo, echo_vol, choice_data )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **dev;** | *choice device number* |
| **Pstore** | **store;** | *OUT pointer to buffer* |
| **Pint** | ***err;** | *OUT error indicator* |
| **Pop_mode** | ***op_mode;** | *OUT operating mode* |
| **Pecho_switch** | ***echo_switch;** | *OUT echo switch* |
| **Pin_status** | ***init_status;** | *OUT initial choice status* |
| **Pint** | ***init_choice;** | *OUT initial choice* |
| **Pint** | ***prompt_echo;** | *OUT prompt/echo type* |
| **Plimit3** | ***echo_vol;** | *OUT echo volume* |
| **Pchoice_data3** | ****choice_data;** | *OUT data record* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE CHOICE DEVICE STATE 3 to retrieve the current state of the specified choice device.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing data associated with the device state. This buffer is passed as the *store* argument.

The store buffer is a data area managed by the PHIGS. While the application is responsible for creating the initial buffer through a call to CREATE STORE, PHIGS manages this area such that there is sufficient memory for the specific inquiry. The choice device data record within the store buffer is accessed via the pointer pointed to by choice_data.

*ws* Workstation identifier. An integer specifying the workstation with which the specified choice device is associated.

*dev* The device number of the choice device. See the *Available Devices* section of INITIALIZE CHOICE 3 for a description of the available devices.

*store* The memory buffer PHIGS is to use for storing the information returned. This buffer must exist prior to calling this function (see CREATE STORE (3P)).

*err* The error indicator. See the *Execution* section below for a description of its use. See the *Errors* section below for the possible values it may return.

*op_mode*

A pointer to a variable of type Pop_mode, which will contain the current operating mode of the device. Pop_mode is enumerated in phigs.h as follows:

```
typedef enum {
        POP_REQ,
        POP_SAMPLE,
        POP_EVENT
} Pop_mode;
```

*echo_switch*

A pointer to a variable of type Pecho_switch, which will contain the state of the device's echo

switch. The value returned for *echo_switch* will be either PSWITCH_ECHO or PSWITCH_NO_ECHO.

*init_status*

A pointer to a variable of type Pin_status, which contains the initial choice status of the device. Possible values for *init_status* are PIN_STATUS_OK and PIN_STATUS_NONE.

*init_choice*

A pointer to an integer that contains the value of the initial choice.

*prompt_echo*

A pointer to an integer that contains the value of the prompt/echo type.

*echo_vol*

A pointer to a Plimit3 structure defining the *x*, *y*, and *z* components of the echo volume, in Device Coordinates. Plimit3 is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          x_min;          /* minimum x coordinate value */
        Pfloat          x_max;          /* maximum x coordinate value */
        Pfloat          y_min;          /* minimum y coordinate value */
        Pfloat          y_max;          /* maximum y coordinate value */
        Pfloat          z_min;          /* minimum z coordinate value */
        Pfloat          z_max;          /* maximum z coordinate value */
} Plimit3;
```

*choice_data*

A pointer to a Pchoice_data3 pointer. PHIGS assigns this pointer to the location in the Pstore structure that contains the device's data record contents. Pchoice_data3 is defined in phigs.h as follows:

```
typedef struct {
    union {
        struct {
                        Pint            unused;
        } pet_r1;
        struct {
                        Pint            num_prompts;    /* number of alternatives */
                        Ppr_switch      *prompts;       /* array of prompts */
        } pet_r2;
        struct {
                        Pint            num_strings;    /* number of choice strings */
                        char            **strings;      /* array of choice strings */
        } pet_r3;
        struct {
                        Pint            num_strings;    /* number of alternatives */
                        char            **strings;      /* array of strings  */
        } pet_r4;
        struct {
                        Pint            struct_id;      /* struct identifier  */
                        Pint            num_pick_ids;   /* number of alternatives */
                        Pint            *pick_ids;      /* array of pick identifiers */
        } pet_r5;
    } pets;
} Pchoice_data3;
```

Ppr_switch is an enumerated type with the following values:

```
typedef enum {
```

```
        PPR_OFF,
        PPR_ON
} Ppr_switch;
```

**Execution**

INQUIRE CHOICE DEVICE STATE 3 returns the current state of the specified choice device, which is stored in the workstation state list of the workstation associated with the device. The current state includes the operating mode, echo switch, initial choice, prompt/echo type, echo volume, and data record. See SET CHOICE MODE for a description of the operating mode and the echo switch, and how to set these values. See INITIALIZE CHOICE 3 for a description of the initial choice, prompt/echo type, echo volume, and data record contents, and how to set these values.

In the event of an error, the *error indicator* indicates the error number of the error detected, and no other output data is returned. If no error is detected, the *error indicator* is set to zero, and the inquired information will be available in the output parameters. Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

| | |
|---|---|
| 003 | Ignoring function, function requires state (PHOP, WSOP, *, *) |
| 054 | Ignoring function, the specified workstation is not open |
| 061 | Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN |
| 250 | Ignoring function, the specified device is not available on the specified workstation |

**SEE ALSO**

SET CHOICE MODE (3P)
INITIALIZE CHOICE 3 (3P)
INQUIRE CHOICE DEVICE STATE (3P)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

INQUIRE COLOUR FACILITIES – inquire a workstation type's colour facilities

**SYNOPSIS**

**C Syntax**

```
void
pinq_colr_facs ( type, error_ind, facilities )
Pint        type;          workstation type
Pint        *error_ind;    OUT error indicator
Pcolr_facs  *facilities;   OUT colour facilities
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE COLOUR FACILITIES to obtain a description of colour facilities for the specified type of workstation.

**C Input Parameter**

*type* Get the colour facilities for this workstation type.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*facilities*

A pointer to the structure in which to store the information, defined as:

```
typedef struct {
        Pint        num_colrs;      /* number of colours */
        Pcolr_avail colr_avail;     /* colour availability */
        Pint        num_pred_inds;  /* number of predefined bundles */
        Pcieluv     prim_colrs;     /* primary colours */
} Pcolr_facs;
```

*colr_avail* returns the colour availability, one of the following enumeration values:

| | |
|---|---|
| PAVAIL_COLR | *Colour* |
| PAVAIL_MONOCHR | *Monochrome* |

*num_pred_inds* returns the number of predefined colour bundle indices.

*prim_colrs* returns the three primary colours CIE (*Commission Internationale de l'Eclairage*'s universal colour definition system) chromaticity coefficients, defined as:

```
typedef struct {
        Pfloat      cieluv_x;       /* x coefficient */
        Pfloat      cieluv_y;       /* y coefficient */
        Pfloat      cieluv_y_lum;   /* y luminance */
} Pcieluv;
```

The *z* primary chromaticity coefficients may be calculated by $1.0 - x - y$.

**Note:** PEX-SI does not support the CIE colour model, so 0.0 is returned for all the chromaticity coefficients.

**ERRORS**

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

052 Ignoring function, workstation type not recognized by the implementation

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

**SEE ALSO**

INQUIRE WORKSTATION CONNECTION AND TYPE (3P)
WORKSTATION TYPE SET (3P)
INQUIRE PREDEFINED COLOUR REPRESENTATION (3P)
PHIGS WORKSTATION DESCRIPTION TABLE (7P)

**NAME**

INQUIRE COLOUR MAPPING FACILITIES – inquire colour mapping facilities for the specified workstation type

**SYNOPSIS**

**C Syntax**

**void**
**pinq_colr_map_facs ( type, length, start, error_ind, colr_map_facs, total_length )**

| | | |
|---|---|---|
| **Pint** | **type;** | *workstation type* |
| **Pint** | **length;** | *length of application list* |
| **Pint** | **start;** | *starting position* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pcolr_map_facs** | ***colr_map_facs;** | *OUT colour mapping facilities* |
| **Pint** | ***total_length;** | *OUT total length of list* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

INQUIRE COLOUR MAPPING FACILITIES returns the available colour mapping facilities for a specified workstation type.

**C Input Parameter**

*type* The workstation type.

*length* The number of Pint entries in the array the application has allocated and assigned to *colr_map_facs→methods.ints*. If zero is specified then the list of available colour mapping methods will not be returned, but the number of available colour mapping methods will be returned in the *total_length* argument.

*start* Starting position of inquiry. The elements in the list, beginning with the item number specified by *start*, are copied sequentially into *colr_map_facs→methods.ints* until *colr_maps_facs→methods.ints* is full or all the elements have been copied.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*colr_map_facs*

A pointer to to the variable in which to store the colour mapping facilities. Pcolr_map_facs is defined in phigs.h as:

```
typedef struct {
        Pint_list       meths;          /* available colour mapping methods */
        Pint            num_pred_inds;  /* number of predefined indices */
 } Pcolr_map_facs;
```

*num_pred_ind* returns the number of predfined colour mapping representations.

*meths* returns the number and list of available colour mapping methods for the specified workstation type. Pint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint            num_ints;       /* number of Pints in list */
        Pint            *ints;          /* list of integers        */
} Pint_list;
fi
```

The *num_ints* component specifies the number of elements in the list.

The *ints* component is a pointer to a list *num_ints* long.

The application must allocate memory for *length* elements in the list of *ints*.

The pre-defined colour mapping methods in phigs.h are:

1 PCOLR_MAP_TRUE
2 PCOLR_MAP_PSEUDO
3 PCOLR_MAP_PSEUDO_N

*total_length*

A pointer to the integer in which to return the length of the list. This is the value required for *length* if all the items in the list are to be returned.

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the following error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**Errors**

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

051 Ignoring function, this information is not yet available for this generic workstation type; open a workstation of this type and use the specific workstation type

052 Ignoring function, workstation type not recognized by the implementation

059 Ignoring function, the specified workstation does not have output capability (i.e. the workstation category is neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

**SEE ALSO**

SET COLOUR MAPPING REPRESENTATION (3P+)
SET COLOUR MAPPING INDEX (3P+)

**NAME**

INQUIRE COLOUR MAPPING METHOD FACILITIES – inquire the available colour mapping facilities specific to a colour mapping method.

**SYNOPSIS**

**C Syntax**

```
void
pinq_colr_map_method_facs ( type, map_method, error_ind, map_st )
Pwstype         type;          workstation type
Pint            map_method;    mapping method
Pint            *error_ind;    OUT error indicator
Pcolr_map_st    *map_st;       OUT method-specific facilities
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE COLOUR MAPPING METHOD FACILITIES to obtain the available colour mapping facilities specific to a specified colour mapping method.

**C Input Parameters**

*type* The workstation type for which to get the colour mapping method facilities.

*map_method*

The colour mapping method of interest. The colour mapping methods defined in phigs.h are as follows:

1 PCOLR_MAP_TRUE
2 PCOLR_MAP_PSUEDO
3 PCOLR_MAP_PSEUDO_N

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*map_st* A pointer to a Pcolr_map_st structure in which to store the information. Pcolr_map_st is defined in phigs.h as:

```
typedef union {
        Pint        int_data;       for map_method 1 and 2
} Pcolr_map_st;
```

For colour mapping method PCOLR_MAP_TRUE, *int_data* will contain the number of available true colours. For colour mapping method PCOLR_MAP_PSEUDO, *int_data* will contain the maximum number of available pseudo colour entries. There is no information returned for colour mapping method PCOLR_MAP_PSEUDO_N.

**Execution**

If the inquired information is available, the error indicator is returned as zero and the requested information is returned.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the following error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

051 Ignoring function, this information is not yet available for this workstation type; open a

workstation of this type and use the specific workstation type

052 Ignoring function, workstation type not recognized by the implementation

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT nor OUTIN)

062 Ignoring function, this information is not available for this MO workstation type

126 Ignoring function, the specified colour mapping method is not available on the specified workstation.

**SEE ALSO**

INQUIRE COLOUR MAPPING FACILITIES (3P+)
PHIGS WORKSTATION DESCRIPTION TABLE (7P)

**NAME**

INQUIRE COLOUR MAPPING REPRESENTATION – inquire a colour mapping representation for a given workstation

**SYNOPSIS**

**C Syntax**

**void**
**pinq_colr_map_rep ( ws, index, type, store, error_ind, map_method, map_data )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **index;** | *colour mapping index* |
| **Pinq_type** | **type;** | *type of returned value* |
| **Pstore** | **store;** | *handle to Store object* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pint** | ***map_method;** | *OUT mapping method* |
| **Pcolr_map_data** | ****map_data;** | *OUT mapping method data record* |

**Required PHIGS Operating States**

(PHOP,WSOP, *, *)

**DESCRIPTION**

**Purpose**

INQUIRE COLOUR MAPPING REPRESENTATION returns a specified colour mapping table entry in a workstation's state list.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing data associated with the device state. This buffer is passed as the *store* argument.

The store buffer is a data area managed by PHIGS. While the application is responsible for creating the inital buffer through a call to STORE, PHIGS manages this area such that there is sufficient memory for the specific inquiry. The data record within the store buffer is accessed by the pointer pointed to by *map_data*.

*ws* The workstation identifier.

*index* The colour mapping table index of the entry to return.

*type* An enumerated value specifying whether the values to be returned are those originally specified by the application (PSET), or those resulting after PHIGS mapped them to ones available on the workstation (PREALIZED). A Pinq_type structure is defined in phigs.h as:

```
typedef enum {
        PSET,
        PREALIZED
} Pinq_type;
```

*store* The memory buffer PHIGS is to use for storing the information returned. This buffer must exist prior to calling this function (see CREATE STORE (3P)).

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*map_method*

A pointer to the location to store the colour mapping method of the represenation. The colour mapping methods are defined in phigs.h as follows:

| | |
|---|---|
| 1 | PCOLR_MAP_TRUE |
| 2 | PCOLR_MAP_PSEUDO |
| 3 | PCOLR_MAP_PSEUDO_N |

*map_data*

A pointer to a pointer to a Pcolr_map_data structure that contains the information specific to the data mapping method. PHIGS will allocate space for the structure in the Pstore object and return the pointer to that allocated space in this parameter. Pcolr_map_data is defined in phigs.h as follows:

```
typedef union {
        struct {
                Pint            colr_model;
                Pfloat_list     weights;
                Pcolr_rep_list  colrs;
        } method_r2
        struct {
                Pint            colr_model;
                Pfloat_list_list colr_lists;
        } method_r3
} Pcolr_map_data;
```

Pfloat_list is defined in phigs.h as follows:

```
typedef struct {
        Pint            num_floats;     /* number of Pfloats in list */
        Pfloat          *floats;        /* list of floats *
} Pfloat_list;
```

Pfloat_list_list is defined in phigs.h as follows:

```
typedef struct {
        Pint            num_lists;      /* number of lists in list */
        Pfloat_list     *lists;         /* list of float lists */
} Pfloat_list_list;
```

Pcolr_rep_list is defined in phigs.h as follows:

```
typedef struct {
        Pint            num_colr_reps;  /* number of colours */
        Pcolr_rep       *colr_reps;     /* array of colours */
} Pcolr_rep_list;
```

Pcolr_rep is defined in phigs.h as follows:

```
typedef union {
        Prgb            rgb;            /* Red Green Blue colour specification */
        Pcieluv         cieluv;         /* CIE L*U*V* colour specification */
        Phls            hls;            /* Hue Lightness Saturation colour specification */
        Phsv            hsv;            /* Hue Saturation Value colour specification */
        Pdata           unsupp;         /* Colour in unsupported colour model */
} Pcolr_rep;
```

Prgb is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          red;            /* red, hue, etc */
        Pfloat          green;          /* green, saturation, lightness, etc */
        Pfloat          blue;           /* blue, value, saturation, etc */
} Prgb;
```

Pcieluv is defined in phigs.h as follows:

```
typedef struct {
```

```
        Pfloat      cieluv_x;       /* x coefficient */
        Pfloat      cieluv_y;       /* y coefficient */
        Pfloat      cieluv_y_lum;   /* y luminance */
} Pcieluv;
```

Phsv is defined in phigs.h as follows:

```
typedef struct {
        Pfloat      hue;            /* hue */
        Pfloat      satur;          /* saturation */
        Pfloat      value;          /* value */
} Phsv;
```

Phls is defined in phigs.h as follows:

```
typedef struct {
        Pfloat      hue;            /* hue */
        Pfloat      lightness;      /* lightness */
        Pfloat      satur;          /* saturation */
} Phls;
```

Pdata is defined in phigs.h as follows:

```
typedef struct {
        size_t      size;           /* size of data */
        char        *data           /*pointer to data */
} Pdata;
```

There is no data associated with mapping method PCOLR_MAP_TRUE. The data for method PCOLR_MAP_PSEUDO is a list of weights and a list of direct colours. *colr_model* specifies the type of colours given in the list of colours, *colrs*. The colour models are defined in phigs.h as follows:

1 PMODEL_RGB
2 PMODEL_CIELUV
3 PMODEL_HSV
4 PMODEL_HLS

The data for mapping method PCOLR_MAP_PSEUDO_N is *N* lists of direct colours. These colours may be of the same types as for PCOLR_MAP_PSEUDO.

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the specified colour mapping index is not present in the colour mapping table on the workstation and the specified type of returned value is *Realized*, the representation for the colour mapping index zero is returned.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the following error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

003 Ignoring function, function requires state (PHOP,WSOP,*,*)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT nor OUTIN)

101 Ignoring function, the specified representation has not been defined

121 Ignoring function, the colour mapping index is less than zero

**SEE ALSO**

SET COLOUR MAPPING REPRESENTATION (3P+)
SET COLOUR MAPPING INDEX (3P+)
INQUIRE LIST OF COLOUR MAPPING INDICES (3P+)
INQUIRE PREDEFINED COLOUR MAPPING REPRESENTATION (3P+)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

INQUIRE COLOUR MAPPING STATE – inquire the current colour mapping state for a specified workstation

**SYNOPSIS**

**C Syntax**

```
void
pinq_colr_map_st ( ws, map_method, error_ind, map_st )
Pint            ws;             workstation identifier
Pint            map_method;     mapping method
Pint            *error_ind;     OUT error indicator
Pcolr_map_st    *map_st;        OUT method-specific state
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE COLOUR MAPPING STATE to determine the currently available colour mapping facilities for a specified colour mapping method and a specified workstation.

**C Input Parameters**

*ws* The workstation identifier.

*map_method*
The colour mapping method.

**C Output Parameters**

*error_ind*
A pointer to the location to store the error number of any error detected by this function.

*map_st* A pointer to a location in which the system returns the current colour mapping state for the specified colour mapping method. Pcolr_map_st is defined in phigs.h as:

```
typedef union {
        Pint        int_data;       /* for map_method 1 and 2 */
} Pcolr_map_st;
```

For colour mapping method PCOLR_MAP_TRUE, *int_data* will contain the number of available true colours. For colour mapping method PCOLR_MAP_PSEUDO, *int_data* will contain the number of pseudo colour entries available. There is no information returned for colour mapping method PCOLR_MAP_PSEUDO_N.

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the following error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is not OUTPUT or OUTIN)

126 Ignoring function, the specified colour mapping method is not available on the specified workstation.

**SEE ALSO**

SET COLOUR MAPPING REPRESENTATION (3P)
INQUIRE COLOUR MAPPING METHOD FACILITIES (3P)

**NAME**

INQUIRE COLOUR MODEL – inquire a workstation's current colour model

**SYNOPSIS**

**C Syntax**

```
void
pinq_colr_model ( ws, error_ind, model )
Pint        ws;          workstation identifier
Pint        *error_ind;  OUT error indicator
Pint        *model;      OUT current colour model
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE COLOUR MODEL to determine the current colour model from a specified PHIGS workstation state list.

**C Input Parameters**

*ws* The workstation identifier of the workstation whose workstation state list is queried.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*model* A pointer to a location in which the system returns the current colour model. The possible colour models are as follows:

| | |
|---|---|
| 0 | PINDIRECT |
| 1 | PMODEL_RGB |
| 2 | PMODEL_CIELUV |
| 3 | PMODEL_HSV |
| 4 | PMODEL_HLS |

**ERRORS**

| | |
|---|---|
| 003 | Ignoring function, function requires state (PHOP, WSOP, *, *) |
| 054 | Ignoring function, the specified workstation is not open |
| 059 | Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT, OUTIN, nor MO) |

**SEE ALSO**

INQUIRE COLOUR MODEL FACILITIES (3P)
SET COLOUR MODEL (3P)

**NAME**

INQUIRE COLOUR MODEL FACILITIES – inquire list of workstation colour model facilities

**SYNOPSIS**

**C Syntax**

**void**
**pinq_colr_model_facs ( type, length, start, error_ind, models, total_length, def )**

| | | |
|---|---|---|
| **Pint** | **type;** | *workstation type* |
| **Pint** | **length;** | *length of application list* |
| **Pint** | **start;** | *starting position* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pint_list** | ***models;** | *OUT list of colour models* |
| **Pint** | ***total_length;** | *OUT length of list in PHIGS* |
| **Pint** | ***def;** | *OUT default colour model* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE COLOUR MODEL FACILITIES to obtain a list of the colour model facilities for the specified type of workstation from the workstation description table.

**C Input Parameters**

*type* Get the colour model facilities for this workstation type.

*length* The number of elements for which memory is allocated in the output parameter *models*. Zero may be specified in order to get the *total_length* of the list.

*start* Starting position in the list at which to begin the inquiry.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*models* A pointer to a Pint_list which returns the portion of the list of available colour models from the workstation description table, starting with *start*. Pint_list is defined in phigs.h as:

```
typedef struct {
        Pint    num_ints;   /* number of integers */
        Pint    *ints;      /* list of integers */
} Pint_list;
```

The pointer to the list of integers must be initialized to an array of *length* Pint elements.

*total_length*

A pointer to an integer in which to return the total length of the list. This is the value required for *length* if all the items in the list are to be returned.

*def* A pointer to an integer in which to return the default colour model. Possible colour models are as follows:

| | |
|---|---|
| 0 | PINDIRECT |
| 1 | PMODEL_RGB |
| 2 | PMODEL_CIELUV |
| 3 | PMODEL_HSV |
| 4 | PMODEL_HLS |

**ERRORS**

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

051 Ignoring function, this information is not yet available for this workstation type; open a

workstation of this type and use the specific workstation type

052 Ignoring function, workstation type not recognized by the implementation

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

**SEE ALSO**

SET COLOUR MODEL (3P)
INQUIRE COLOUR FACILITIES (3P)
PHIGS WORKSTATION DESCRIPTION TABLE (7P)

**NAME**

INQUIRE COLOUR REPRESENTATION – inquire a colour representation on a workstation

**SYNOPSIS**

**C Syntax**

```
void
pinq_colr_rep ( ws, index, type, error_ind, rep )
Pint            ws;             workstation identifier
Pint            index;          colour index
Pinq_type       type;           type of returned value
Pint            *error_ind;     OUT error indicator
Pcolr_rep       *rep;           OUT colour representation
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE COLOUR REPRESENTATION to determine the current representation in a specified colour table entry from a specified workstation's state list.

**C Input Parameters**

*ws* The workstation identifier of the workstation whose state list is queried.

*index* Entry to be returned from the workstation's colour table; if this entry is not present in the table and the *type of returned value* parameter is REALIZED, the representation for colour index 1 is returned.

*type* An enumerated variable specifying whether the value desired is the value specified by the application program or the available value used by the workstation. Valid values are:

| Value | Meaning |
|---|---|
| PINQ_SET | Return the values exactly as they are specified in the application program. |
| PINQ_REALIZED | Return the values as they are used by the workstation when the values specified in the application program are mapped to values available on the workstation. |

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*rep* A pointer to a structure in which the system returns the colour representation at entry *index*. Pcolr_rep is defined in phigs.h as follows:

```
typedef union {
        Prgb        rgb;        /* Red Green Blue colour specification */
        Pcieluv     cieluv;     /* CIE L*U*V* colour specification */
        Phls        hls;        /* Hue Lightness Saturation colour specification */
        Phsv        hsv;        /* Hue Saturation Value colour specification */
        Pdata       unsupp;     /* Colour in unsupported colour model */
} Pcolr_rep;
```

Prgb is defined in phigs.h as follows:

```
typedef struct {
        Pfloat      red;        /* red, hue, etc */
        Pfloat      green;      /* green, saturation, lightness, etc */
        Pfloat      blue;       /* blue, value, saturation, etc */
} Prgb;
```

Pcieluv is defined in phigs.h as follows:

```
typedef struct {
```

```
        Pfloat          cieluv_x;       /* x coefficient */
        Pfloat          cieluv_y;       /* y coefficient */
        Pfloat          cieluv_y_lum;   /* y luminance */
} Pcieluv;
```

Phsv is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          hue;            /* hue */
        Pfloat          satur;          /* saturation */
        Pfloat          value;          /* value */
} Phsv;
```

Phls is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          hue;            /* hue */
        Pfloat          lightness;      /* lightness */
        Pfloat          satur;          /* saturation */
} Phls;
```

Pdata is defined in phigs.h as follows:

```
typedef struct {
        size_t          size;           /* size of data */
        char            *data           /* pointer to data */
} Pdata;
```

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (i.e., the workstation category is neither OUTPUT, OUTIN, nor MO)

113 Ignoring function, the colour index value is less than zero

101 Ignoring function, the specified representation has not been defined

**SEE ALSO**

INQUIRE COLOUR FACILITIES (3P)
SET COLOUR REPRESENTATION (3P)
INQUIRE PREDEFINED COLOUR REPRESENTATION (3P)

**NAME**

INQUIRE CONFLICT RESOLUTION – inquire current values of conflict resolution flags

**SYNOPSIS**

**C Syntax**

```
void
pinq_conf_res ( error_ind, archive, retrieval )
Pint            *error_ind;        OUT error indicator
Pconf_res       *archive;          OUT archival resolution
Pconf_res       *retrieval;        OUT retrieval resolution
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE CONFLICT RESOLUTION to determine the current values of the archival and retrieval conflict resolution flags. The flag values may be MAINTAIN, ABANDON, or UPDATE.

See the description of the subroutine SET CONFLICT RESOLUTION for information on the meaning of these values.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*archive* A pointer to a Pconf_res enumerated type in which the system returns the current value of the archival conflict resolution flag.

*retrieval*

A pointer to a Pconf_res enumerated type in which the system returns the current value of the retrieval conflict resolution flag. Values for the Pconf_res enumerated type are defined in phigs.h as follows:

PRES_MAINTAIN
PRES_ABANDON
PRES_UPD

**ERRORS**

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

**SEE ALSO**

SET CONFLICT RESOLUTION (3P)

**NAME**

INQUIRE CONFLICTING STRUCTURES IN NETWORK – inquire conflicting structures in a network in the Central Structure Store and an archive file

**SYNOPSIS**

**C Syntax**

**void**
**pinq_conf_structs_net ( ar_id, struct_id, src, length, start, error_ind, ids, total_length )**

| | | |
|---|---|---|
| **Pint** | **ar_id;** | *archive identifier* |
| **Pint** | **struct_id;** | *structure identifier* |
| **Pstruct_net_source** | **src;** | *structure network source* |
| **Pint** | **length;** | *length of application list* |
| **Pint** | **start;** | *starting position* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pint_list** | ***ids;** | *OUT conflicting struct id list* |
| **Pint** | ***total_length;** | *OUT length of list in PHIGS* |

**Required PHIGS Operating States**

(PHOP, *, *, AROP)

**DESCRIPTION**

**Purpose**

Use INQUIRE CONFLICTING STRUCTURES IN NETWORK to obtain a list of the identifiers of all structures in a specified network that exist in both the Central Structure Store and the specified open archive file.

**C Input Parameters**

*ar_id* The archive identifier specifying the open archive file to use.

*struct_id*
The identifier of the root structure of the network.

*src* The structure network source, defined in phigs.h as:

PNET_CSS *Central Structure Store*
PNET_AR *Archive*

*length* The number of *integers* items in the *ids* output parameter for which the application has allocated memory. *length* is the number of list elements (structure identifiers) that the system can return in *ids–>integers*. If a value of 0 is used here, no data will be returned in the *ids–>integers* list, but the total number of conflicting structures will be returned in *total_length*.

*start* Starting position in the list of identifiers of conflicting structures at which to begin the inquiry. The elements of the list of structure identifiers, beginning with the item number specified by *start*, are copied sequentially into *ids–>integers* until *ids–>integers* is full or all the structure identifiers have been copied.

**C Output Parameters**

*error_ind*
A pointer to the location to store the error number of any error detected by this function.

*ids* A pointer to a Pint_list structure in which the list of identifiers of conflicting structures is returned. The Pint_list structure is defined in phigs.h as follows:

```
typedef struct {
        Pint    num_ints;    /* number of Pints in list */
        Pint    *ints;       /* list of integers */
} Pint_list;
```

The *num_ints* component specifies the number of structure identifiers in the list. The *ints*

component is a pointer to a list, *num_ints* long, of the structure identifiers.

The application must allocate memory for *length* elements in the *ids–>integers* list.

*total_length*

A pointer to an integer in which the total number of conflicting structures is returned. This is the value required for *length* if all structure identifiers are to be returned.

**Execution**

If the structure network source is the Central Structure Store, the identifiers of all structures in the network defined by the root *structure identifier* in the Central Structure Store are compared to the identifiers of *all* structures in the specified archive file, and those which are in both are returned as conflicting structures. If the structure network source is the archive file, the identifiers of all structures in the network defined by the root *structure identifier* in the specified archive file are compared to the identifiers of *all* structures in the Central Structure Store, and those which are in both are returned as conflicting structures.

**ERRORS**

007 Ignoring function, function requires state (PHOP, *, *, AROP)

201 Ignoring function, the specified structure does not exist

404 Ignoring function, the specified archive file is not open

**SEE ALSO**

INQUIRE ALL CONFLICTING STRUCTURES (3P)

**NAME**

INQUIRE CURRENT ELEMENT CONTENT – inquire the contents of the current element

**SYNOPSIS**

**C Syntax**

```
void
pinq_cur_elem_content (store, error_ind, data )
Pstore          store;          handle to Store object
Pint            *error_ind;     OUT error indicator
Pelem_data      **data;         OUT data record
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

INQUIRE CURRENT ELEMENT CONTENT returns the contents of the current element.

**C Input Parameter**

Applications using the C binding must create a buffer to be used by this function as memory space for storing data associated with the device state. This buffer is passed as the *store* argument.

The store buffer is a data area managed by PHIGS. While the application is responsible for creating the inital buffer through a call to STORE, PHIGS manages this area such that there is sufficient memory for the specific inquiry. The data record within the store buffer is accessed by the pointer pointed to by *data*.

*store* The memory buffer PHIGS is to use for storing the information returned. This buffer must exist prior to calling this function (see CREATE STORE (3P)).

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*data* A pointer to a Pelem_data union, in which the contents of the current element will be returned, as appropriate for the element type. Pelem_data is defined in phigs.h as follows:

```
typedef union {
        Pint                    int_data;           /* integer valued data */
        Pfloat                  float_data;         /* float valued data */
        Ppoint_list3            point_list3;        /* list of 3d points */
        Ppoint_list             point_list;         /* list of 2d points */
        Ppoint_list_list3   point_list_list3;   /* list of 3d point lists */
        Ppoint_list_list    point_list_list;    /* list of 2d point lists */
        struct {
                Ppoint3         pos;            /* text pt */
                Pvec3           dir[2];         /* direction vectors */
                char            *char_string;   /* text string */
        } text3;
        struct {
                Ppoint          pos;            /* text pt */
                char            *char_string;   /* text string */
        } text;
        struct {
                Ppoint3         ref_point;      /* reference pt */
                Pvec3           offset;         /* anno. pt/offset */
                char            *char_string;   /* text string */
        } anno_text_rel3;
        struct {
                Ppoint          ref_point;      /* reference pt */
```

```
        Pvec            offset;         /* anno. pt/offset */
        char            *char_string;   /* text string */
} anno_text_rel;
struct {
        Pparal          paral;          /* parallelogram */
 Ppat_rep       colr_array;     /* colour array */
} cell_array3;
struct {
        Prect           rect;           /* rectangle */
 Ppat_rep       colr_array;     /* colour array */
} cell_array;
struct {
        Pint            id;             /* GDP3 id */
        Ppoint_list3    point_list;     /* pts */
        Pgdp_data3      data;           /* data record */
} gdp3;
struct {
        Pint            id;             /* GDP id */
        Ppoint_list     point_list;     /* pts */
        Pgdp_data       data;           /* data record */
} gdp;
Ptext_prec              text_prec;      /* text precision */
Pvec                    char_up_vec;    /* char up vector */
Ptext_path              text_path;      /* text path */
Ptext_align             text_align;     /* text alignment */
Pint_style              int_style;      /* interior style */
Pedge_flag              edge_flag;      /* edge flag */
Ppoint                  pat_ref_point;  /* pat ref pt */
Pfloat_size             pat_size;       /* pattern size */
struct {
        Ppoint3         ref_point;      /* pattern ref. pt */
        Pvec3           ref_vec[2];     /* vectors */
} pat_ref_point_vecs;
Pint_list               names;          /* name sets */
struct {
        Paspect         id;             /* attribute id */
        Pasf            source;         /* asf */
} asf;
struct {
        Pcompose_type   compose_type;   /* composition type */
        Pmatrix3        matrix;         /* matrix */
} local_tran3;
struct {
        Pcompose_type   compose_type;   /* composition type */
        Pmatrix             matrix;     /* matrix */
} local_tran;
Pmatrix3                global_tran3;   /* global transform3 */
Pmatrix                 global_tran;    /* global transform */
struct {
        Pint             op;            /* operator */
        Phalf_space_list3 half_spaces;  /* half-space list */
} model_clip3;
struct {
```

```
        Pint                op;                 /* operator */
        Phalf_space_list    half_spaces;        /* half-space list */
} model_clip;
Pclip_ind                   clip_ind;           /* clipping indicator */
Pdata                       appl_data;          /* application data */
struct {
        Pint                id;                 /* GSE id */
        Pgse_data           data;               /* GSE data record */
} gse;
struct {
        Pint                order;
        Pfloat_list         knots;
        Prational           rationality;
        Ppoint_list34       cpts;
        Pfloat              min;
        Pfloat              max;
} nurb_curve;                                   /* NURBS curve */
struct {
        Pint                type;
        Pfloat              value;
} curv_approx;                                  /* curve approximation criteria */
struct {
        Pint                u_order;
        Pint                v_order;
        Prational           rationality;
        Pfloat_list         uknots;
        Pfloat_list         vknots;
        Ppoint_grid34       grid;
        Pint                num_trim_loops;
        Ptrimcurve_list     *trim_loops;
} nurb_surf;                                    /* NURBS surface */
struct {
        Pint                type;
        Pfloat              u_val;
        Pfloat              v_val;
} surf_approx;                                  /* surface approximation criteria */
struct {
        Pint                vflag;
        Pint                colr_model;
        Pint                npl;
        Pline_vdata_list3   *vdata;
} plsd3;
struct {
        Pint                fflag;
        Pint                eflag;
        Pint                vflag;
        Pint                colr_model;
        Pfacet_data3        fdata;
        Pint                nfa;
        Pedge_data_list     *edata;
        Pfacet_vdata_list3  *vdata;
} fasd3;                                        /* fill area set with data */
struct {
```

```
            Pint                    fflag;
            Pint                    vflag;
            Pint                    colr_model;
            Pint                    nv;
            Pfacet_data_arr3        fdata;
            Pfacet_vdata_arr3       vdata;
        } tsd3;                                     /* triangle strip with data */
        struct {
            Pint                    fflag;
            Pint                    vflag;
            Pint                    colr_model;
            Pint_size               dim;
            Pfacet_data_arr3        fdata;
            Pfacet_vdata_arr3       vdata;
        } qmd3;                                     /* quad mesh */
        struct {
            Pint                    fflag;
            Pint                    eflag;
            Pint                    vflag;
            Pint                    colr_model;
            Pint                    num_sets;
            Pfacet_data_arr3        fdata;
            Pedge_data_list_list    *edata;
            Pint_list_list          *vlist;
            Pfacet_vdata_list3      vdata;
        } sofas3;                   /* set of fill area set */
        struct {
            Pparal                  paral;          /* parallelogram */
            Ppat_rep_plus   colr_array;     /* colour array */
        } cell_array_plus;
        Pgcolr                      colr;
        Prefl_props                 props;          /*surface relectance properties */
        struct {
            Pint_list               activation;
            Pint_list               deactivation;
        } lss;                      /* light source state */
        Pcull_mode                  cull_mode;      /* culling mode */
        Pdisting_mode               disting_mode;   /* distinguishing mode */
        struct {
            Pint                    type;
            Ppara_surf_characs      data;
        } para_surf_charac;         /* parametric surface characteristics */
} Pelem_data;
```

**Execution**

INQUIRE CURRENT ELEMENT CONTENT is the second function called in a two function, or step, process. First INQUIRE CURRENT ELEMENT TYPE AND SIZE must be called to determine the type of element to return for the INQUIRE CURRENT ELEMENT CONTENT request. Depending on the element type, the corresponding structure in the Pelem_data union is returned, as described below.

ADD NAMES TO SET (PEL_ADD_NAMES_TO_SET)

A Pint_list structure provided in Pelem_data as:

```
Pint_list names             /* name sets */
```

where

```
typedef struct {
        Pint    num_ints;       /* number of Pints in list */
        Pint    *ints;          /* list of integers */
} Pint_list;
```

ALL (PELEM_ALL)

No data involved

ANNOTATION STYLE (PELEM_ANNO_STYLE)

An integer provided in Pelem_data as:

```
        Pint            int_data;           /* integer valued data */
```

ANNOTATION TEXT ALIGNMENT (PELEM_ANNO_ALIGN)

A Ptext_align structure provided in Pelem_data as:

```
        Ptext_align     text_align;         /* text alignment */
```

where

```
typedef struct {
        Phor_text_align  hor;               /* horizontal component */
        Pvert_text_align vert;              /* vertical component */
} Ptext_align;
```

ANNOTATION TEXT CHARACTER HEIGHT (PELEM_ANNO_CHAR_HT)

A Pfloat floating point data provided in Pelem_data as:

```
        Pfloat          float_data;         /* float valued data */
```

ANNOTATION TEXT CHARACTER UP VECTOR (PELEM_ANNO_CHAR_UP_VEC)

A Pvec structure provided in Pelem_data as:

```
        Pvec            char_up_vec;        /* char up vector */
```

where

```
typedef struct {
        Pfloat          delta_x;            /* x magnitude */
        Pfloat          delta_y;            /* y magnitude */
} Pvec;
```

ANNOTATION TEXT PATH (PELEM_ANNO_PATH)

A Ptext_path data structure provided in Pelem_data as:

```
        Ptext_path      text_path;          /* text path */
```

where

```
typedef enum {
        PPATH_RIGHT,
        PPATH_LEFT,
        PPATH_UP,
        PPATH_DOWN
} Ptext_path;
```

ANNOTATION TEXT RELATIVE (PELEM_ANNO_TEXT_REL)

An anno_text_rel structure defined in Pelem_data as:

```
struct {
        Ppoint          ref_point;          /* reference pt */
        Pvec            offset;             /* annotation offset */
        char            *char_string;       /* annotation text string */
} anno_text_rel;
```

ANNOTATION TEXT RELATIVE 3 (PELEM_ANNO_TEXT_REL3)

An anno_text_rel3 structure defined in Pelem_data as:

```
struct {
        Ppoint3         ref_point;          /* reference pt */
        Pvec3           offset;             /* annotation offset */
        char            *char_string;       /* annotation text string */
} anno_text_rel3;
```

APPLICATION DATA (PELEM_APPL_DATA)

A Pdata structure provided in Pelem_data as:

```
        Pdata           appl_data;          /* application data */
where
typedef struct {
        size_t          size;               /* size of data */
        char            *data;              /* pointer to data */
} Pdata;
```

AREA PROPERTIES (PELEM_REFL_PROPS)

A Prefl_props structure provided in Pelem_data as:

```
        Prefl_props     properties;         /* area properties */
where
typedef struct {
        Pfloat          ambient_coef;       /* ambient reflectance coefficient */
        Pfloat          diffuse_coef;       /* diffuse reflectance coefficient */
        Pfloat          specular_coef;      /* specular reflectance coefficient */
        Pgcolr          specular_colr;      /* specular colour */
        Pfloat          specular_exp;       /* specular exponent */
        Pfloat          transpar_coef;      /* transparency coefficient */
} Prefl_props;
```

BACK AREA PROPERTIES (PELEM_BACK_REFL_PROPS)

A Prefl_props structure provided in Pelem_data as:

```
        Prefl_props     properties;         /* area properties */
where
typedef struct {
        Pfloat          ambient_coef;       /* ambient reflectance coefficient */
        Pfloat          diffuse_coef;       /* diffuse reflectance coefficient */
        Pfloat          specular_coef;      /* specular reflectance coefficient */
        Pgcolr          specular_colr;      /* specular colour */
        Pfloat          specular_exp;       /* specular exponent */
        Pfloat          transpar_coef;      /* transparency coefficient */
} Prefl_props;
```

BACK INTERIOR COLOUR (PELEM_BACK_INT_COLR)

A Pgcolr structure provided in Pelem_data as:

```
        Pgcolr          colr;               /* extended colour model */
where
typedef struct {
        Pint    type;   /* indirect, RGB, CIE, HSV, HLS */
        union {
                Pint    ind;        /* index in workstation colour bundle table */
                struct {
```

```
                    Pfloat    x;        /* red, hue, etc */
                    Pfloat    y;        /* green, saturation, lightness, etc */
                    Pfloat    z;        /* blue, value, saturation, etc */
                } general;
            } val;
        } Pgcolr;
```

BACK INTERIOR REFLECTANCE EQUATION (PELEM_BACK_INT_REFL_EQN)

An integer provided in Pelem_data as:

```
        Pint        int_data;        /* integer valued data */
```

The predefined reflectance equation values are:

| | | |
|---|---|---|
| 1 | PREFL_NONE | *No Reflectance Calculation Performed* |
| 2 | PREFL_AMBIENT | *Use Ambient Term* |
| 3 | PREFL_AMB_DIFF | *Use Ambient and Diffuse Terms* |
| 4 | PREFL_AMB_DIFF_SPEC | *Use Ambient, Diffuse, and Specular Terms* |

BACK INTERIOR SHADING METHOD (PELEM_BACK_INT_SHAD_METH)

An integer provided in Pelem_data as:

```
        Pint        int_data;        /* integer valued data */
```

The predefined shading method values are:

| | | |
|---|---|---|
| 1 | PSD_NONE | *No Shading* |
| 2 | PSD_COLOUR | *Colour Interpolation Shading* |
| 3 | PSD_DOT_PRODUCT | *Dot Product Interpolation Shading* |
| 4 | PSD_NORMAL | *Normal Interpolation Shading* |

BACK INTERIOR STYLE (PELEM_BACK_INT_STYLE)

An Pint_style structure provided in Pelem_data as:

```
        Pint_style      int_style;      /* back interior style */
```

where

typedef enum {

```
        PSTYLE_HOLLOW,
        PSTYLE_SOLID,
        PSTYLE_PAT,
        PSTYLE_HATCH,
        PSTYLE_EMPTY
```

} Pint_style;

BACK INTERIOR STYLE INDEX (PELEM_BACK_INT_STYLE_IND)

An integer provided in Pelem_data as:

```
        Pint        int_data;        /* integer valued data */
```

CELL ARRAY (PELEM_CELL_ARRAY)

A cell_array structure defined in Pelem_data as:

struct {

```
        Prect       rect;           /* rectangle */
        Pint_size   dim;            /* dimension */
        Pint        *colr;          /* colour array */
```

} cell_array;

CELL ARRAY 3 (PELEM_CELL_ARRAY3)

A cell_array3 structure defined in Pelem_data as:

struct {

```
        Pparal      paral;          /* parallelogram */
```

```
        Pint_size       dim;            /* dimension */
        Pint            *colr;          /* colour array */
} cell_array3;
```

CHARACTER EXPANSION FACTOR (PELEM_CHAR_EXPAN)

A floating point provided in Pelem_data as:

```
        Pfloat          float_data;     /* float valued data */
```

CHARACTER HEIGHT (PELEM_CHAR_HT)

A floating point provided in Pelem_data as:

```
        Pfloat          float_data;     /* float valued data */
```

CHARACTER SPACING (PELEM_CHAR_SPACE)

A float_data floating point data provided in Pelem_data as:

```
        Pfloat          float_data;     /* float valued data */
```

CHARACTER UP VECTOR (PELEM_CHAR_UP_VEC)

A Pvec structure provided in Pelem_data as:

```
        Pvec            char_up_vec;    /* char up vector */
```

where

```
typedef struct {
        Pfloat          delta_x;        /* x magnitude */
        Pfloat          delta_y;        /* y magnitude */
} Pvec;
```

COLOUR MAPPING INDEX (PEX_COLR_MAPPING_IND)

An integer provided in Pelem_data as:

```
        Pint            int_data;       /* integer valued data */
```

CURVE APPROXIMATION CRITERIA (PELEM_CURVE_APPROX_CRIT)

A curv_approx structure defined in Pelem_data as:

```
struct {
        Pint            type;           /* approximation type */
        Pfloat          value;          /* approximation value */
} curv_approx;
```

DEPTH CUE INDEX (PELEM_DCUE_IND)

An integer provided in Pelem_data as:

```
        Pint            int_data;       /* integer valued data */
```

EDGE COLOUR (PELEM_EDGE_COLR)

A Pgcolr structure provided in Pelem_data as:

```
        Pgcolr          colr;           /* extended colour model */
```

where

```
typedef struct {
        Pint    type;   /* indirect, RGB, CIE, HSV, HLS */
        union {
                Pint    ind;    /* index in workstation colour bundle table */
                struct {
                        Pfloat  x;      /* red, hue, etc */
                        Pfloat  y;      /* green, saturation, lightness, etc */
                        Pfloat  z;      /* blue, value, saturation, etc */
                } general;
        } val;
```

```
} Pgcolr;
```

EDGE COLOUR INDEX (PELEM_EDGE_COLR_IND)

An integer provided in Pelem_data as:

```
Pint        int_data;        /* integer valued data */
```

EDGE FLAG (PELEM_EDGE_FLAG)

A Pedge_flag data structure provided in Pelem_data as:

```
Pedge_flag  edge_flag;       /* edge flag */
```

where

```
typedef enum {
        PEDGE_OFF,
        PEDGE_ON
} Pedge_flag;
```

EDGE INDEX (PELEM_EDGE_IND)

An integer provided in Pelem_data as:

```
Pint        int_data;        /* integer valued data */
```

EDGETYPE (PELEM_EDGETYPE)

An integer provided in Pelem_data as:

```
Pint        int_data;        /* integer valued data */
```

The predefined line type values are:

| | |
|---|---|
| 1 | PLINE_SOLID |
| 2 | PLINE_DASH |
| 3 | PLINE_DOT |
| 4 | PLINE_DOT_DASH |

EDGEWIDTH SCALE FACTOR (PELEM_EDGEWIDTH)

A floating point provided in Pelem_data as:

```
Pfloat      float_data;      /* float valued data */
```

EXECUTE STRUCTURE (PELEM_EXEC_STRUCT)

An integer provided in Pelem_data as:

```
Pint        int_data;        /* integer valued data */
```

CELL ARRAY 3 PLUS (PELEM_CELL_ARRAY3_PLUS)

An cell_array_plus structure provided in Pelem_data as:

```
struct {
        Pparal      paral;       /* parallelogram */
        Pint_size   dim;         /* dimension */
        Pint        colr_model;  /* colour model */
        Pcoval      *colr;       /* colour array */
} cell_array_plus;
```

FACE CULLING MODE (PELEM_FACE_CULL_MODE)

An Pcull_mode data structure provided in Pelem_data as:

```
Pcull_mode  cull_mode;       /* culling mode */
```

where

```
typedef enum {
        PCULL_NONE,
        PCULL_BACKFACE,
        PCULL_FRONTFACE
} Pcull_mode;
```

FACE DISTINGUISHING MODE (PELEM_FACE_DISTING_MODE)

An Pdisting_mode data structure provided in Pelem_data as:

```
        Pdisting_mode    disting_mode;        /* distinguishing mode */
```

where

```
typedef enum {
        PDISTING_NO,
        PDISTING_YES
} Pdisting_mode;
```

FILL AREA (PELEM_FILL_AREA)

A Ppoint_list structure provided in Pelem_data as:

```
        Ppoint_list      point_list;          /* list of 2d points */
```

where

```
typedef struct {
        Pint             num_points;          /* number of Ppoints in the list */
        Ppoint           *points;             /* list of points */
} Ppoint_list;
```

FILL AREA SET (PELEM_FILL_AREA_SET)

A fill_area_set structure defined in Pelem_data as:

```
struct {
        Pint             num_point_lists;     /* number of sets */
        Ppoint_list      *point_lists;        /* list of sets */
} fill_area_set;
```

FILL AREA SET 3 (PELEM_FILL_AREA_SET3)

A fill_area_set3 structure defined in Pelem_data as:

```
struct {
        Pint             num_point_lists;     /* number of sets */
        Ppoint_list3     *point_lists;        /* list of sets */
} fill_area_set3;
```

FILL AREA SET 3 WITH DATA (PELEM_FILL_AREA_SET3_DATA)

A fasd3 structure defined in Pelem_data as:

```
struct {
        Pint             fflag;               /* data specified per facet */
        Pint             eflag;               /* edge visibility status */
        Pint             vflag;               /* data per vertex flag */
        Pint             colr_model;          /* colour type */
        Pfacet_data3     fdata;               /* facet data */
        Pint             nfa;                 /* number of fill areas in the set */
        Pedge_data_list  *edata;              /* edge data */
        Pfacet_vdata_arr3 *vdata;             /* vertex data */
} fasd3;
```

FILL AREA 3 (PELEM_FILL_AREA3)

A Ppoint_list3 structure provided in Pelem_data as:

```
        Ppoint_list3     point_list3;         /* list of 3d points */
```

where

```
typedef struct {
        Pint             num_points;          /* number of Ppoint3s in the list*/
        Ppoint3          *points;             /* list of points */
```

```
} Ppoint_list3;
```

GDP (PELEM_GDP)

A gdp structure defined in Pelem_data as:

```
struct {
        Pint            id;             /* GDP id */
        Ppoint_list     point_list;     /* pts */
        Pgdp_data       data;           /* data record */
} gdp;
```

GDP 3 (PELEM_GDP3)

A gdp3 structure defined in Pelem_data as:

```
struct {
        Pint            id;             /* GDP3 id */
        Ppoint_list3    point_list;     /* pts */
        Pgdp_data3      data;           /* data record */
} gdp3;
```

GLOBAL MODELLING TRANSFORMATION (PELEM_GLOBAL_MODEL_TRAN)

A Pmatrix structure provided in Pelem_data as:

```
        Pmatrix         global_tran;    /* global transform */
where
typedef  Pfloat Pmatrix[3][3];
```

GLOBAL MODELLING TRANSFORMATION 3 (PELEM_GLOBAL_MODEL_TRAN3)

A Pmatrix3 structure provided in Pelem_data as:

```
        Pmatrix3        global_tran3;   /* global transform3 */
where
  typedef Pfloat Pmatrix3[4][4];
```

GSE (PELEM_GSE)

A gse structure defined in Pelem_data as:

```
struct {
        Pint            id;             /* GSE id */
        Pgse_data       data;           /* GSE data record */
} gse;
```

HLHSR IDENTIFIER (PELEM_HLHSR_ID)

An integer provided in Pelem_data as:

```
        Pint            int_data;       /* integer valued data */
```

INDIVIDUAL ASF (PELEM_INDIV_ASF)

An asf structure defined in Pelem_data as:

```
struct {
        Paspect         id;             /* attribute id */
        Pasf            source;         /* asf */
} asf;
```

where

```
typedef enum {
   PASPECT_LINETYPE,
   PASPECT_LINEWIDTH,
   PASPECT_LINE_COLR_IND,
```

```
    PASPECT_MARKER_TYPE,
    PASPECT_MARKER_SIZE,
    PASPECT_MARKER_COLR_IND,
    PASPECT_TEXT_FONT,
    PASPECT_TEXT_PREC,
    PASPECT_CHAR_EXPAN,
    PASPECT_CHAR_SPACE,
    PASPECT_TEXT_COLR_IND,
    PASPECT_INT_STYLE,
    PASPECT_INT_STYLE_IND,
    PASPECT_INT_COLR_IND,
    PASPECT_EDGE_FLAG,
    PASPECT_EDGETYPE,
    PASPECT_EDGEWIDTH,
    PASPECT_EDGE_COLR_IND,
    PASPECT_CURVE_APPROX_CRIT,
    PASPECT_SURF_APPROX_CRIT,
    PASPECT_LINE_SHAD_METH,
    PASPECT_REFL_PROPS,
    PASPECT_INT_REFL_EQN,
    PASPECT_INT_SHAD_METH,
    PASPECT_BACK_INT_STYLE,
    PASPECT_BACK_INT_STYLE_IND,
    PASPECT_BACK_INT_COLR,
    PASPECT_BACK_REFL_PROPS,
    PASPECT_BACK_INT_REFL_EQN,
    PASPECT_BACK_INT_SHAD_METH
} Paspect;
```

and

```
typedef enum {
    PASF_BUNDLED,
    PASF_INDIV
} Pasf;
```

INTERIOR COLOUR (PELEM_INT_COLR)

A Pgcolr structure provided in Pelem_data as:

```
    Pgcolr      colur;      /* extended colour model */
```

where

```
typedef struct {
    Pint    type;   /* indirect, RGB, CIE, HSV, HLS */
    union {
        Pint    ind;    /* index in workstation colour bundle table */
        struct {
            Pfloat  x;      /* red, hue, etc */
            Pfloat  y;      /* green, saturation, lightness, etc */
            Pfloat  z;      /* blue, value, saturation, etc */
        } general;
    } val;
} Pgcolr;
```

INTERIOR COLOUR INDEX (PELEM_INT_COLR_IND)

An integer provided in Pelem_data as:

```
    Pint        int_data;       /* integer valued data */
```

INTERIOR INDEX (PELEM_INT_IND)

An integer provided in Pelem_data as:

```
Pint        int_data;        /* integer valued data */
```

INTERIOR REFLECTANCE EQUATION (PELEM_INT_REFL_EQN)

An integer provided in Pelem_data as:

```
Pint        int_data;        /* integer valued data */
```

The predefined reflectance equation values are:

| | | |
|---|---|---|
| 1 | PREFL_NONE | *No Reflectance Calculation Performed* |
| 2 | PREFL_AMBIENT | *Use Ambient Term* |
| 3 | PREFL_AMB_DIFF | *Use Ambient and Diffuse Terms* |
| 4 | PREFL_AMB_DIFF_SPEC | *Use Ambient, Diffuse, and Specular Terms* |

INTERIOR SHADING METHOD (PELEM_INT_SHAD_METH)

An integer provided in Pelem_data as:

```
Pint        int_data;        /* integer valued data */
```

The predefined shading method values are:

| | | |
|---|---|---|
| 1 | PSD_NONE | *No Shading* |
| 2 | PSD_COLOUR | *Colour Interpolation Shading* |
| 3 | PSD_DOT_PRODUCT | *Dot Product Interpolation Shading* |
| 4 | PSD_NORMAL | *Normal Interpolation Shading* |

INTERIOR STYLE (PELEM_INT_STYLE)

An Pint_style data structure provided in Pelem_data as:

```
Pint_style  int_style;       /* interior style */
```

where

```
typedef enum {
        PSTYLE_HOLLOW,
        PSTYLE_SOLID,
        PSTYLE_PAT,
        PSTYLE_HATCH,
        PSTYLE_EMPTY,
} Pint_style;
```

INTERIOR STYLE INDEX (PELEM_INT_STYLE_IND)

An integer provided in Pelem_data as:

```
Pint        int_data;        /* integer valued data */
```

LABEL (PELEM_LABEL)

An integer provided in Pelem_data as:

```
Pint        int_data;        /* integer valued data */
```

LIGHT SOURCE STATE (PELEM_LIGHT_SRC_STATE)

An lss structure defined in Pelem_data as:

```
struct {
        Pint_list   activation;     /* activation list */
        Pint_list   deactivation;   /* deactivation list */
} lss;
```

LINETYPE (PELEM_LINETYPE)

An integer provided in Pelem_data as:

```
Pint        int_data;        /* integer valued data */
```

The predefined line type values are:

1 PLINE_SOLID
2 PLINE_DASH
3 PLINE_DOT
4 PLINE_DOT_DASH

LINEWIDTH SCALE FACTOR (PELEM_LINEWIDTH)
A floating point provided in Pelem_data as:

```
Pfloat      float_data;     /* float valued data */
```

LOCAL MODELLING TRANSFORMATION (PELEM_LOCAL_MODEL_TRAN)
A local_tran structure defined in Pelem_data as:

```
struct {
    Pcompose_type   comp;       /* composition type */
    Pmatrix         tran;       /* matrix */
} local_tran;
```

where

```
typedef enum {
    PTYPE_PRECONCAT,
    PTYPE_POSTCONCAT,
    PTYPE_REPLACE
} Pcompose_type;
```

LOCAL MODELLING TRANSFORMATION 3 (PELEM_LOCAL_MODEL_TRAN3)
A local_tran3 structure defined in Pelem_data as:

```
struct {
    Pcompose_type   comp;       /* composition type */
    Pmatrix3        tran;       /* matrix */
} local_tran3;
```

where

```
typedef enum {
   PTYPE_PRECONCAT,
   PTYPE_POSTCONCAT,
   PTYPE_REPLACE
} Pcompose_type;
```

MARKER COLOUR INDEX (PELEM_MARKER_COLR_IND)
An integer provided in Pelem_data as:

```
Pint        int_data;       /* integer valued data */
```

MARKER SIZE SCALE FACTOR (PELEM_MARKER_SIZE)
A floating point provided in Pelem_data as:

```
Pfloat      float_data;     /* float valued data */
```

MARKER TYPE (PELEM_MARKER_TYPE)
An integer provided in Pelem_data as:

```
Pint        int_data;       /* integer valued data */
```

The predefined marker type values are:

1 PMARKER_DOT
2 PMARKER_PLUS
3 PMARKER_ASTERISK
4 PMARKER_CIRCLE
5 PMARKER_CROSS

MODELLING CLIPPING INDICATOR (PELEM_MODEL_CLIP_IND)

A Pclip_ind structure provided in Pelem_data as:

```
        Pclip_ind       clip_ind;       /* clipping indicator */
```

where

```
typedef enum {
        PIND_NO_CLIP,
        PIND_CLIP
} Pclip_ind;
```

MODELLING CLIPPING VOLUME (PELEM_MODEL_CLIP_VOL)

A model_clip structure defined in Pelem_data as:

```
struct {
        Pint                op;             /* operator */
        Phalf_space_list    half_spaces;    /* half-space list */
} model_clip;
```

MODELLING CLIPPING VOLUME 3 (PELEM_MODEL_CLIP_VOL3)

A model_clip3 structure defined in Pelem_data as:

```
struct {
        Pint                op;             /* operator */
        Phalf_space_list3   half_spaces;    /* half-space list */
} model_clip3;
```

NIL (PELEM_NIL)

No data involved

NON-UNIFORM B-SPLINE CURVE (PELEM_NUNI_BSP_CURVE)

A nurb_curve structure defined in Pelem_data as:

```
struct {
        Pint            order;          /* curve order */
        Pfloat_list     knots;          /* knots */
        Prational       rationality;    /* rationality specifier */
        Ppoint_list34   cpts;           /* control points */
        Pfloat          min;            /* curve min */
        Pfloat          max;            /* curve max */
} nurb_curve;
```

NON-UNIFORM B-SPLINE SURFACE (PELEM_NUNI_BSP_SURF)

A nurb_surf structure defined in Pelem_data as:

```
struct {
        Pint            u_order;        /* order of u surface */
        Pint            v_order;        /* order of v surface */
        Prational       rationality;    /* rationality specifier */
        Pfloat_list     uknots;         /* knots in u dimension */
        Pfloat_list     vknots;         /* knots in v dimension */
        Ppoint_grid34   grid;           /* grid of control points */
        Pint            num_trim_loops; /* number of trim curve loops */
        Ptrimcurve_list *trim_loops;    /* trim curve loops */
} nurb_surf;
```

NUM_EL_TYPES (PELEM_NUM_EL_TYPES)

No data involved

PARAMETRIC SURFACE CHARACTERISTICS (PEL_PARA_SURF_CHARACTERISITICS)

A para_surf_characs structure provided in Pelem_data as:

```
struct {
        Pint                type;
        Ppara_surf_characs  data;
} para_surf_characs;        /* parametric surface characteristics */
```

where

```
typedef union {
    struct {
        Pint                unused;
    } psc_1;
    struct {
        Pint                unused;
    } psc_2;
    struct {
        Pcurve_placement    placement;
        Pint                u_count;
        Pint                v_count;
    } psc_3;
    struct {
        Ppoint3             origin;
        Pvec3               direction;
        Pfloat_list         params;
    } psc_4;
    struct {
        Ppoint3             origin;
        Pvec3               direction;
        Pfloat_list         params;
    } psc_5;
} Ppara_surf_characs;
```

PATTERN REFERENCE POINT (PELEM_PAT_REF_POINT)

A Ppoint structure provided in Pelem_data as:

```
Ppoint      point;          /* pattern reference pt */
```

where

```
typedef struct {
    Pfloat      x;          /* x coordinate */
    Pfloat      y;          /* y coordinate */
} Ppoint;
```

PATTERN REFERENCE POINT AND VECTORS (PELEM_PAT_REF_POINT_VECS)

A pat_ref_point_vecs structure defined in Pelem_data as:

```
struct {
    Ppoint3     ref_point;      /* pattern ref. pt */
    Pvec3       ref_vec[2];     /* vectors */
} pat_ref_point_vecs;
```

PATTERN SIZE (PELEM_PAT_SIZE)

A Ppoint structure provided in Pelem_data as:

```
Ppoint      point;          /* pattern size */
```

where

```
typedef struct {
    Pfloat      x;          /* x coordinate */
```

```
        Pfloat          y;              /* y coordinate */
    } Ppoint;
```

PICK ID (PELEM_PICK_ID)

An integer provided in Pelem_data as:

```
        Pint            int_data;       /* integer valued data */
```

POLYLINE (PELEM_POLYLINE)

A Ppoint_list structure provided in Pelem_data as:

```
        Ppoint_list     point_list;     /* list of 2d points */
```

where

```
typedef struct {
        Pint            num_points;     /* number of Ppoints */
        Ppoint          *points;        /* array of points */
} Ppoint_list;
```

POLYLINE COLOUR (PELEM_LINE_COLR)

A Pgcolr structure provided in Pelem_data as:

```
        Pgcolr          colr;           /* extended colour model */
```

where

```
typedef struct {
        Pint    type;   /* indirect, RGB, CIE, HSV, HLS */
        union {
            Pint    ind;    /* index in workstation colour bundle table */
            struct {
                Pfloat  x;      /* red, hue, etc */
                Pfloat  y;      /* green, saturation, lightness, etc */
                Pfloat  z;      /* blue, value, saturation, etc */
            } general;
        } val;
} Pgcolr;
```

POLYLINE COLOUR INDEX (PELEM_LINE_COLR_IND)

An integer provided in Pelem_data as:

```
        Pint            int_data;       /* integer valued data */
```

POLYLINE INDEX (PELEM_LINE_IND)

An integer provided in Pelem_data as:

```
        Pint            int_data;       /* integer valued data */
```

POLYLINE SET 3 WITH DATA (PELEM_POLYLINE_SET3_DATA)

A plsd3 structure defined in Pelem_data as:

```
    struct {
        Pint                vflag;      /* data per vertex flag */
        Pint                colr_model; /* colour type */
        Pint                npl;        /* number of polylines in set */
        Pline_vdata_list3   *vdata;     /* per line vertex data list */
    } plsd3;
```

POLYLINE SHADING METHOD (PELEM_LINE_SHAD_METH)

An integer provided in Pelem_data as:

```
        Pint            int_data;       /* integer valued data */
```

The predefined shading method values are:

| | | |
|---|---|---|
| 1 | PSD_NONE | *No Shading* |
| 2 | PSD_COLOUR | *Colour Interpolation Shading* |

POLYLINE 3 (PELEM_POLYLINE3)

A Ppoint_list3 structure provided in Pelem_data as:

```
        Ppoint_list3    pts3;           /* list of 3d points */
where
typedef struct {
        Pint            num_points;     /* number of Ppoint3s */
        Ppoint3         *points;        /* array of points */
} Ppoint_list3;
```

POLYMARKER (PELEM_POLYMARKER)

A Ppoint_list structure provided in Pelem_data as:

```
        Ppoint_list     pts;            /* list of 2d points */
where
typedef struct {
        Pint            num_points;     /* number of Ppoints */
        Ppoint          *points;        /* array of points */
} Ppoint_list;
```

POLYMARKER COLOUR (PELEM_MARKER_COLR)

A Pgcolr structure provided in Pelem_data as:

```
        Pgcolr          colr;           /* extended colour model */
where
typedef struct {
        Pint    type;   /* indirect, RGB, CIE, HSV, HLS */
        union {
                Pint    ind;    /* index in workstation colour bundle table */
                struct {
                        Pfloat  x;      /* red, hue, etc */
                        Pfloat  y;      /* green, saturation, lightness, etc */
                        Pfloat  z;      /* blue, value, saturation, etc */
                } general;
        } val;
} Pgcolr;
```

POLYMARKER INDEX (PELEM_MARKER_IND)

An integer provided in Pelem_data as:

```
        Pint            int_data;       /* integer valued data */
```

POLYMARKER 3 (PELEM_POLYMARKER3)

A Ppoint_list3 structure provided in Pelem_data as:

```
        Ppoint_list3    point_list3;    /* list of 3d points */
where
typedef struct {
        Pint            num_points;     /* number of Ppoint3s */
        Ppoint3         *points;        /* array of points */
} Ppoint_list3;
```

QUADRILATERAL MESH 3 WITH DATA (PELEM_QUAD_MESH3_DATA)

A qmd3 structure defined in Pelem_data as:

```
struct {
```

```
        Pint                fflag;          /* data per facet flag */
        Pint                vflag;          /* data per vertex flag */
        Pint                colr_model;     /* colour type */
        Pint_size           dim;            /* dimension of cells */
        Pfacet_data_arr3    fdata;          /* facet data */
        Pfacet_vdata_arr3   vdata;          /* vertex data */
    } qmd3;
```

REMOVE NAMES FROM SET (PELEM_REMOVE_NAMES_SET)

A Pint_list structure provided in Pelem_data as:

```
    Pint_list   names       /* name sets */
```

where

```
typedef struct {
    Pint        num_ints;   /* number of names in the set */
    Pint        *ints;      /* name set */
} Pint_list
```

RENDERING COLOUR MODEL (PELEM_RENDERING_COLR_MODEL)

An integer provided in Pelem_data as:

```
    Pint        int_data;   /* integer valued data */
```

RESTORE MODELLING CLIPPING VOLUME (PELEM_RESTORE_MODEL_CLIP_VOL)

No data involved

SET OF FILL AREA SET 3 WITH DATA (PELEM_SET_OF_FILL_AREA_SET3_DATA)

A sofas3 structure defined in Pelem_data as:

```
struct {
    Pint                fflag;
    Pint                eflag;
    Pint                vflag;
    Pint                colr_model;
    Pint                num_sets;
    Pfacet_data_arr3    fdata;
    Pedge_data_list_list *edata;
    Pint_list_list      *vlist;
    Pfacet_vdata_list3  vdata;
} sofas3;
```

SURFACE APPROXIMATION CRITERIA (PELEM_SURF_APPROX_CRIT)

A surf_approx structure defined in Pelem_data as:

```
struct {
    Pint        type;       /* approximation type */
    Pfloat      u_val;      /* u approximation value */
    Pfloat      v_val;      /* v approximation value */
} surf_approx;
```

TEXT (PELEM_TEXT)

A text structure defined in Pelem_data as:

```
struct {
    Ppoint      pos;        /* text pt */
    char        *char_string;   /* text string */
} text;
```

TEXT ALIGNMENT (PELEM_TEXT_ALIGN)

A Ptext_align structure provided in Pelem_data as:

```
        Ptext_align     text_align;         /* text alignment */
```

where

```
typedef struct {
        Phor_text_align  hor;               /* horizontal component */
        Pvert_text_align vert;              /* vertical component */
} Ptext_align;
```

TEXT COLOUR (PELEM_TEXT_COLR)

A Pgcolr structure provided in Pelem_data as:

```
        Pgcolr          colr;               /* extended colour model */
```

where

```
typedef struct {
        Pint    type;  /* indirect, RGB, CIE, HSV, HLS */
        union {
                Pint    ind;    /* index in workstation colour bundle table */
                struct {
                        Pfloat  x;      /* red, hue, etc */
                        Pfloat  y;      /* green, saturation, lightness, etc */
                        Pfloat  z;      /* blue, value, saturation, etc */
                } general;
        } val;
} Pgcolr;
```

TEXT COLOUR INDEX (PELEM_TEXT_COLR_IND)

An integer provided in Pelem_data as:

```
        Pint            int_data;           /* integer valued data */
```

TEXT FONT (PELEM_TEXT_FONT)

An integer provided in Pelem_data as:

```
        Pint            int_data;           /* integer valued data */
```

The predefined text font values are:

```
 1      PFONT_MONO
```

TEXT INDEX (PELEM_TEXT_IND)

An integer provided in Pelem_data as:

```
        Pint            int_data;           /* integer valued data */
```

TEXT PATH (PELEM_TEXT_PATH)

A Ptext_path data structure provided in Pelem_data as:

```
        Ptext_path      text_path;          /* text path */
```

where

```
typedef enum {
        PPATH_RIGHT,
        PPATH_LEFT,
        PPATH_UP,
        PPATH_DOWN
} Ptext_path;
```

TEXT PRECISION (PELEM_TEXT_PREC)

A Ptext_prec data structure provided in Pelem_data as:

```
        Ptext_prec      text_prec;          /* text precision */
```

where

```
typedef enum {
        PPREC_STRING,
        PPREC_CHAR,
        PPREC_STROKE
} Ptext_prec;
```

TEXT 3 (PELEM_TEXT3)

A text3 structure defined in Pelem_data as:

```
struct {
        Ppoint3         pos;            /* text pt */
        Pvec3           dir[2];         /* direction vectors */
        char            *char_string;   /* text string */
} text3;
```

TRIANGLE STRIP 3 WITH DATA (PELEM_TRI_STRIP3_DATA)

A tsd3 structure defined in Pelem_data as:

```
struct {
        Pint                fflag;          /* data per facet flag */
        Pint                vflag;          /* data per vertex flag */
        Pint                colr_model;     /* colour type */
        Pint                nv;             /* number of vertices */
        Pfacet_data_arr3    fdata;          /* facet data */
        Pfacet_vdata_arr3   vdata;          /* vertex data */
} tsd3;
```

VIEW INDEX (PELEM_VIEW_IND)

An integer provided in Pelem_data as:

```
        Pint            int_data;       /* integer valued data */
```

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

INQUIRE CURRENT ELEMENT TYPE AND SIZE (3P)
UNPACK DATA RECORD (3P)
INQUIRE ELEMENT CONTENT (3P)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

INQUIRE CURRENT ELEMENT TYPE AND SIZE – inquire the type and size of the current element

**SYNOPSIS**

**C Syntax**

**void**
**pinq_cur_elem_type_size ( error_ind, type, size )**

| | | |
|---|---|---|
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pelem_type** | ***type;** | *OUT element type* |
| **size_t** | ***size;** | *OUT element size* |

**Required PHIGS Operating States**

(PHOP, ∗, STOP, ∗)

**DESCRIPTION**

**Purpose**

INQUIRE CURRENT ELEMENT TYPE AND SIZE returns the type and size of the current element.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*type* Returns the type of the current element. For example, a polyline primitive element would be returned as PELEM_POLYLINE, a character height attribute as PEL_CHARACTER_HEIGHT, a modelling transformation as PEL_LOCAL_MODELLING_TRANSFORMATION3, and so forth. The Pelem_type enumerated type is defined in phigs.h as follows:

```
typedef enum {
        PELEM_ALL,
        PELEM_NIL,
        PELEM_POLYLINE3,
        PELEM_POLYLINE,
        PELEM_POLYMARKER3,
        PELEM_POLYMARKER,
        PELEM_TEXT3,
        PELEM_TEXT,
        PELEM_ANNO_TEXT_REL3,
        PELEM_ANNO_TEXT_REL,
        PELEM_FILL_AREA3,
        PELEM_FILL_AREA,
        PELEM_FILL_AREA_SET3,
        PELEM_FILL_AREA_SET,
        PELEM_CELL_ARRAY3,
        PELEM_CELL_ARRAY,
        PELEM_GDP3,
        PELEM_GDP,
        PELEM_LINE_IND,
        PELEM_MARKER_IND,
        PELEM_TEXT_IND,
        PELEM_INT_IND,
        PELEM_EDGE_IND,
        PELEM_LINETYPE,
        PELEM_LINEWIDTH,
        PELEM_LINE_COLR_IND,
        PELEM_MARKER_TYPE,
        PELEM_MARKER_SIZE,
        PELEM_MARKER_COLR_IND,
```

PELEM_TEXT_FONT,
PELEM_TEXT_PREC,
PELEM_CHAR_EXPAN,
PELEM_CHAR_SPACE,
PELEM_TEXT_COLR_IND,
PELEM_CHAR_HT,
PELEM_CHAR_UP_VEC,
PELEM_TEXT_PATH,
PELEM_TEXT_ALIGN,
PELEM_ANNO_CHAR_HT,
PELEM_ANNO_CHAR_UP_VEC,
PELEM_ANNO_PATH,
PELEM_ANNO_ALIGN,
PELEM_ANNO_STYLE,
PELEM_INT_STYLE,
PELEM_INT_STYLE_IND,
PELEM_INT_COLR_IND,
PELEM_EDGE_FLAG,
PELEM_EDGETYPE,
PELEM_EDGEWIDTH,
PELEM_EDGE_COLR_IND,
PELEM_PAT_SIZE,
PELEM_PAT_REF_POINT_VECS,
PELEM_PAT_REF_POINT,
PELEM_ADD_NAMES_SET,
PELEM_REMOVE_NAMES_SET,
PELEM_INDIV_ASF,
PELEM_HLHSR_ID,
PELEM_LOCAL_MODEL_TRAN3,
PELEM_LOCAL_MODEL_TRAN,
PELEM_GLOBAL_MODEL_TRAN3,
PELEM_GLOBAL_MODEL_TRAN,
PELEM_MODEL_CLIP_VOL3,
PELEM_MODEL_CLIP_VOL,
PELEM_MODEL_CLIP_IND,
PELEM_RESTORE_MODEL_CLIP_VOL,
PELEM_VIEW_IND,
PELEM_EXEC_STRUCT,
PELEM_LABEL,
PELEM_APPL_DATA,
PELEM_GSE,
PELEM_PICK_ID,
PELEM_POLYLINE_SET3_DATA,
PELEM_FILL_AREA_SET3_DATA,
PELEM_TRI_STRIP3_DATA,
PELEM_QUAD_MESH3_DATA,
PELEM_SET_OF_FILL_AREA_SET3_DATA,
PELEM_NUNI_BSP_CURVE,
PELEM_NUNI_BSP_SURF,
PELEM_CELL_ARRAY3_PLUS,
PELEM_TEXT_COLR,
PELEM_MARKER_COLR,
PELEM_EDGE_COLR,

```
        PELEM_LINE_COLR,
        PELEM_CURVE_APPROX_CRIT,
        PELEM_LINE_SHAD_METH,
        PELEM_INT_COLR,
        PELEM_BACK_INT_COLR,
        PELEM_BACK_INT_STYLE,
        PELEM_BACK_INT_STYLE_IND,
        PELEM_REFL_PROPS,
        PELEM_BACK_REFL_PROPS,
        PELEM_INT_SHAD_METH,
        PELEM_BACK_INT_SHAD_METH,
        PELEM_INT_REFL_EQN,
        PELEM_BACK_INT_REFL_EQN,
        PELEM_SURF_APPROX_CRIT,
        PELEM_PARA_SURF_CHARACS,
        PELEM_FACE_DISTING_MODE,
        PELEM_FACE_CULL_MODE,
        PELEM_LIGHT_SRC_STATE,
        PELEM_DCUE_IND,
        PELEM_COLR_MAP_IND,
        PELEM_RENDERING_COLR_MODEL,
        PELEM_NUM_EL_TYPES
    } Pelem_type;
```

*size* Returns the size, in bytes, that the application will have to allocate in order to call INQUIRE CURRENT ELEMENT CONTENT to retrieve the contents of the current element. If the current element type is such that it is not necessary to allocate any dynamic memory to retrieve its contents, a value of zero is returned.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

INQUIRE CURRENT ELEMENT CONTENT (3P)
INQUIRE ELEMENT POINTER (3P)
ELEMENT SEARCH (3P)
INQUIRE ELEMENT TYPE AND SIZE (3P)

**NAME**

INQUIRE CURVE AND SURFACE FACILITIES – inquire the available curve and surface facilities of a workstation type

**SYNOPSIS**

**C Syntax**

**void**
**pinq_curv_surf_facs ( type, cat_len, cat_st, sat_len, sat_st, tcat_len, tcat_st, psc_len,**
**psc_st, error_ind, facil, tot_cat_len, tot_sat_len, tot_tcat_len, tot_psc_len )**

| | | |
|---|---|---|
| **Pint** | **type;** | *workstation type* |
| **Pint** | **cat_len;** | *length of curve approx types list* |
| **Pint** | **cat_st;** | *starting position* |
| **Pint** | **sat_len;** | *length of surface approx types list* |
| **Pint** | **sat_st;** | *starting position* |
| **Pint** | **tcat_len;** | *length of trim curve approx types list* |
| **Pint** | **tcat_st;** | *starting position* |
| **Pint** | **psc_len;** | *length of parametric surface characteristics list* |
| **Pint** | **psc_st;** | *starting position* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pcurvsurf_facs** | ***facil;** | *OUT curve and surface facilities* |
| **Pint** | ***tot_cat_len;** | *OUT total length of curve approx types list* |
| **Pint** | ***tot_sat_len;** | *OUT total length of surface approx types list* |
| **Pint** | ***tot_tcat_len;** | *OUT total length of trim curve approx types list* |
| **Pint** | ***tot_psc_len;** | *OUT total length of parametric surface characteristics list* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

INQUIRE CURVE AND SURFACE FACILITIES returns the available curve and surface facilities associated with the specified workstation type.

**C Input Parameters**

All of the following data types are predefined in phigs.h.

*type* The workstation type in question.

*cat_len* The number of entries to return from the list of curve approximation types.

*cat_st* The starting position of the portion of the list to return.

*sat_len* The number of entries to return from the list of surface approximation types.

*sat_st* The starting position of the portion of the list to return.

*tcat_len* The number of entries to return from the list of trimming curve approximation types.

*tcat_st* The starting position of the portion of the list to return.

*psc_len* The number of entries to return from the list of parametric surface characteristics.

*psc_st* The starting position of the portion of the list to return.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*facil* A pointer to a Pcurvsurf_facs structure which returns the requested information. Pcurvsurf_facs is defined as:

```
typedef struct {
        Pint        max_bsp_order;    /* maximum B-spline order */
```

```
Pint        max_pp_order;    /* maximum parametric polynomial order */
Pint        max_tc_order;    /* maximum trim curve order */
Pint_list   ppc_types;       /* list of parametric polynomial curve types */
Pint_list   pps_types;       /* list of parametric polynomial surface types*/
Pint_list   cat_types;       /* list of curve approximation types */
Pint_list   sat_types;       /* list of surface approximation types */
Pint_list   tcat_types;      /* list of trim curve approximation types */
Pint_list   psc_types;       /* list of parametric surface characteristics types */
Pint        num_pred_inds;   /* number of predefined bundles */
} Pcurvsurf_facs;
```

Pint_list is defined as:

```
typedef struct {
    Pint    num_ints;   /* number of Pints in list */
    Pint    *ints;      /* list of integers */
} Pint_list;
```

Prior to calling this function, the integers field of the Pint_list structures must contain a pointer to an array of integers. This array must be at least as large as the corresponding *len* parameter.

*tot_cat_len*
: A pointer to an integer which returns the total length of the curve approximation type list as stored in the workstation description table.

*tot_sat_len*
: A pointer to an integer which returns the total length of the surface approximation type list as stored in the workstation description table.

*tot_tcat_len*
: A pointer to an integer which returns the total length of the trimming curve approximation type list as stored in the workstation description table.

*tot_psc_len*
: A pointer to an integer which returns the total length of the parametric surface characteristics. list as stored in the workstation description table.

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the error numbers listed below to indicate the reason for non-availability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

051 Ignoring function, this information is not yet available for this generic workstation type; open a workstation of this type and use the specific workstation type

052 Ignoring function, workstation type not recognized by the implementation

059 Ignoring function, the specified workstation does not have output capability (i.e. the workstation category is neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

**SEE ALSO**

PHIGS WORKSTATION DESCRIPTION TABLE (7P)

NON-UNIFORM B-SPLINE CURVE (3P+)
NON-UNIFORM B-SPLINE SURFACE (3P+)
SET CURVE APPROXIMATION CRITERIA (3P+)
SET SURFACE APPROXIMATION CRITERIA (3P+)
SET TRIMMING CURVE APPROXIMATION CRITERIA (3P+)

**NAME**

INQUIRE DEFAULT CHOICE DEVICE DATA – inquire the predefined choice data

**SYNOPSIS**

**C Syntax**

**void**
**pinq_def_choice_data ( type, device, store, error_ind, max_choices, pets, echo_area, choice_data )**

| | | |
|---|---|---|
| **Pint** | **type;** | *workstation type* |
| **Pint** | **device;** | *logical input device number* |
| **Pstore** | **store;** | *handle to Store object* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pint** | ***max_choices;** | *OUT max. number of choices* |
| **Pint_list** | ****pets;** | *OUT list of prompt and echo types* |
| **Plimit** | ***echo_area;** | *OUT default echo area* |
| **Pchoice_data** | ****choice_data;** | *OUT pointer to default data record* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE DEFAULT CHOICE DEVICE DATA to determine the following information for a CHOICE input device associated with a given workstation type:

- Number and list of available prompt/echo types
- Default echo area
- Default input data record
- Maximum number of available choices

Since the default prompt/echo type for all input devices is 1, the default input data record is for that prompt/echo type. There are no default input data records for prompt/echo types other than 1.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing portions of the device data. This buffer is passed as the *store* argument.

The store buffer is a data area managed by PHIGS. While the application is responsible for creating the initial buffer through a call to CREATE STORE, PHIGS manages this area such that there is sufficient memory for the specific inquiry. The choice device data record within the store buffer is accessed via the pointer returned in *choice_data*.

*type* The workstation type with which the device is associated.

*device* The device number of the choice device. See the *Available Devices* section of INITIALIZE CHOICE for a description of the available devices.

*store* The memory buffer PHIGS is to use for storing the information returned for the Pchoice_data structure. This memory is not used for the data returned in the Pint_list structure. This buffer must exist prior to calling this function (see CREATE STORE (3P)).

**C Output Parameters**

*error_ind*

The error indicator. See the *Execution* section below for a description of its use. See the *Errors* section below for the possible values it may return.

*max_choices*

Maximum number of choices.

*pets* A pointer to a list of the available prompt/echo types. Pint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint            num_ints;       /* number of Pints in list */
        Pint            *ints;          /* list of integers        */
} Pint_list;
```

The *num_ints* component specifies the number of elements in the list. The *ints* component is a pointer to a list *num_ints* long.

The application must allocate memory for *len* elements in the list of *ints*.

*echo_area*

A pointer a Plimit structure to contain the default echo area. Plimit is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          x_min;          /* x min */
        Pfloat          x_max;          /* x max */
        Pfloat          y_min;          /* y min */
        Pfloat          y_max;          /* y max */
} Plimit;
```

*choice_data*

A pointer to a Pchoice_data pointer. PHIGS assigns this pointer to the location in the Pstore structure that contains the data record contents. Pchoice_data is defined in phigs.h as follows:

```
typedef struct {
    union {
        struct {
                Pint            unused;
        } pet_r1;
        struct {
                Pint            num_prompts;    /* number of alternatives */
                Ppr_switch      *prompts;       /* array of prompts */
        } pet_r2;
        struct {
                Pint            num_strings;    /* number of choice strings */
                char            **strings;      /* array of choice strings */
        } pet_r3;
        struct {
                Pint            num_strings;    /* number of alternatives */
                char            **strings;      /* array of strings */
        } pet_r4;
        struct {
                Pint            struct_id;      /* struct identifier */
                Pint            num_pick_ids;   /* number of alternatives */
                Pint            *pick_ids;      /* array of pick identifiers */
        } pet_r5;
    } pets;
} Pchoice_data;
```

**Execution**

INQUIRE DEFAULT CHOICE DEVICE DATA returns the default data of the specified choice device. This data is stored in the workstation description table associated with the workstation type. See INITIALIZE CHOICE for a description of the prompt/echo types, echo area, and data record contents and how to set these values.

In the event of an error, the *error indicator* indicates the error number of the error detected, and no other output data is returned. If no error is detected, the *error indicator* is set to zero, and the inquired information will be available in the output parameters. Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

052 Ignoring function, workstation type not recognized by the implementation

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

**SEE ALSO**

INITIALIZE CHOICE (3P)
INQUIRE DEFAULT CHOICE DEVICE DATA 3 (3P)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

INQUIRE DEFAULT CHOICE DEVICE DATA 3 – inquire the predefined choice data

**SYNOPSIS**

**C Syntax**

**void**
**pinq_def_choice_data3 ( type, device, store, error_ind, max_choices, pets, echo_vol, choice_data )**

| | | |
|---|---|---|
| **Pint** | **type;** | *workstation type* |
| **Pint** | **device;** | *logical input device number* |
| **Pstore** | **store;** | *handle to Store object* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pint** | ***max_choices;** | *OUT max. number of choices* |
| **Pint_list** | ****pets;** | *OUT list of prompt and echo types* |
| **Plimit3** | ***echo_vol;** | *OUT default echo volume* |
| **Pchoice_data3** | ****choice_data;** | *OUT pointer to default data record* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE DEFAULT CHOICE DEVICE DATA 3 to determine the following information for a CHOICE input device associated with a given workstation type:

- Number and list of available prompt/echo types
- Default echo volume
- Default input data record
- Maximum number of available choices

Since the default prompt/echo type for all input devices is 1, the default input data record is for that prompt/echo type. There are no default input data records for prompt/echo types other than 1.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing portions of the device data. This buffer is passed as the *store* argument.

The store buffer is a data area managed by PHIGS. While the application is responsible for creating the initial buffer through a call to CREATE STORE, PHIGS manages this area such that there is sufficient memory for the specific inquiry. The choice device data record within the store buffer is accessed via the pointer returned in *choice_data*.

*type* The workstation type with which the device is associated.

*device* The device number of the choice device. See the *Available Devices* section of INITIALIZE CHOICE 3 for a description of the available devices.

*store* The memory buffer PHIGS is to use for storing the information returned for the Pchoice_data3 structure. This memory is not used for the data returned in the Pint_list structure. This buffer must exist prior to calling this function (see CREATE STORE (3P)).

**C Output Parameters**

*error_ind*

The error indicator. See the *Execution* section below for a description of its use. See the *Errors* section below for the possible values it may return.

*max_choices*

Maximum number of choices.

*pets* A pointer to a list of available prompt/echo types. Pint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint            num_ints;       /* number of Pints in list */
        Pint            *ints;          /* list of integers        */
} Pint_list;
```

The *num_ints* component specifies the number of elements in the list. The *ints* component is a pointer to a list *num_ints* long.

The application must allocate memory for *len* elements in the list of *ints*.

*echo_vol*

A pointer to a Plimit3 structure defining the *x*, *y*, and *z* components of the echo volume, in Device Coordinates. Plimit3 is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          x_min;          /* minimum x coordinate value */
        Pfloat          x_max;          /* maximum x coordinate value */
        Pfloat          y_min;          /* minimum y coordinate value */
        Pfloat          y_max;          /* maximum y coordinate value */
        Pfloat          z_min;          /* minimum z coordinate value */
        Pfloat          z_max;          /* maximum z coordinate value */
} Plimit3;
```

*choice_data*

A pointer to a Pchoice_data3 pointer. PHIGS assigns this pointer to the location in the Pstore structure that contains the device's data record contents. Pchoice_data3 is defined in phigs.h as follows:

```
typedef struct {
    union {
        struct {
                Pint            unused;
        } pet_r1;
        struct {
                Pint            num_prompts;    /* number of alternatives */
                Ppr_switch      *prompts;       /* array of prompts */
        } pet_r2;
        struct {
                Pint            num_strings;    /* number of choice strings */
                char            **strings;      /* array of choice strings */
        } pet_r3;
        struct {
                Pint            num_strings;    /* number of alternatives */
                char            **strings;      /* array of strings */
        } pet_r4;
        struct {
                Pint            struct_id;      /* struct identifier */
                Pint            num_pick_ids;   /* number of alternatives */
                Pint            *pick_ids;      /* array of pick identifiers */
        } pet_r5;
    } pets;
} Pchoice_data3;
```

**Execution**

INQUIRE DEFAULT CHOICE DEVICE DATA 3 returns the default data of the specified choice device. This data is stored in the workstation description table associated with the workstation type. See INITIALIZE CHOICE 3

for a description of the prompt/echo types, echo volume, and data record contents, and how to set these values.

In the event of an error, the *error indicator* indicates the error number of the error detected, and no other output data is returned. If no error is detected, the *error indicator* is set to zero, and the inquired information will be available in the output parameters. Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

052 Ignoring function, workstation type not recognized by the implementation

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

**SEE ALSO**

INITIALIZE CHOICE 3 (3P)
INQUIRE DEFAULT CHOICE DEVICE DATA (3P)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

INQUIRE DEFAULT DISPLAY UPDATE STATE – inquire the default display update state for a specified workstation type

**SYNOPSIS**

**C Syntax**

```
void
pinq_def_disp_upd_st ( type, error_ind, def_mode, mod_mode )
Pint            type;           workstation type
Pint            *error_ind;     OUT error indicator
Pdefer_mode     *def_mode;      OUT deferral mode
Pmod_mode       *mod_mode;      OUT modification mode
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE DEFAULT DISPLAY STATE to determine the default display update state for a specified workstation type. The display update state consists of a deferral mode and a modification mode.

See the description of the function SET DISPLAY UPDATE STATE for information on the meaning of these modes.

**C Input Parameters**

*type* Type of workstation.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*def_mode*

The deferral mode, *def_mode*, is defined in phigs.h as:

```
typedef enum {
        PDEFER_ASAP,    As Soon As Possible
        PDEFER_BNIG,    Before the Next Interaction Globally
        PDEFER_BNIL,    Before the Next Interaction Locally
        PDEFER_ASTI,    At Some Time
        PDEFER_WAIT     When the Application Requests It
} Pdefer_mode;
```

*mod_mode*

The modification mode, *mod_mode*, is defined in phigs.h as:

```
typedef enum {
        PMODE_NIVE,     No immediate visual effects mandated
        PMODE_UWOR,     Update without regeneration
        PMODE_UQUM      Use quick update methods
} Pmod_mode;
```

**ERRORS**

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

052 Ignoring function, workstation type not recognized by the implementation

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

059 Ignoring function, the specified workstation does not have output capability (i.e. the workstation

category in neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

**SEE ALSO**

SET DISPLAY UPDATE STATE (3P)
INQUIRE DISPLAY UPDATE STATE (3P)

**NAME**

INQUIRE DEFAULT LOCATOR DEVICE DATA – inquire the predefined locator data

**SYNOPSIS**

**C Syntax**

**void**
**pinq_def_loc_data ( type, device, store, error_ind, loc_pos, pets, echo_area, loc_data )**

| | | |
|---|---|---|
| **Pint** | **type;** | *workstation type* |
| **Pint** | **device;** | *logical input device number* |
| **Pstore** | **store;** | *handle to Store object* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Ppoint** | ***loc_pos;** | *OUT default initial position* |
| **Pint_list** | ****pets;** | *OUT list of prompt and echo types* |
| **Plimit** | ***echo_area;** | *OUT default echo area* |
| **Ploc_data** | ****loc_data;** | *OUT pointer to default data record* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE DEFAULT LOCATOR DEVICE DATA to determine the following information for a LOCATOR input device associated with a given workstation type:

- Number and list of available prompt/echo types
- Default echo area
- Default input data record
- Default initial locator position

Since the default prompt/echo type for all input devices is 1, the default input data record is for that prompt/echo type. There are no default input data records for prompt/echo types other than 1.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing portions of the device data. This buffer is passed as the *store* argument.

The store buffer is a data area managed by PHIGS. While the application is responsible for creating the initial buffer through a call to CREATE STORE, PHIGS manages this area such that there is sufficient memory for the specific inquiry. The locator device data record within the store buffer is accessed via the pointer returned in *loc_data*.

*type* A Pint value specifying the workstation type with which the device is associated.

*device* The device number of the locator device. See the *Available Devices* section of INITIALIZE LOCATOR for a description of the available devices.

*store* The memory buffer PHIGS is to use for storing the information returned for the Ploc_data structure. This memory is not used for the data returned in the Pint_list structure. This buffer must exist prior to calling this function (see CREATE STORE (3P)).

**C Output Parameters**

*error_ind*

The error indicator. See the *Execution* section below for a description of its use. See the *Errors* section below for the possible values it may return.

*loc_pos* The default initial locator position.

*pets* A pointer to a list of prompt and echo types. Pint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint        num_ints;       /* number of Pints in list */
```

```
    Pint        *ints;          /* list of integers    */
} Pint_list;
```

The *num_ints* component specifies the number of elements in the list. The *ints* component is a pointer to a list *num_ints* long.

The application must allocate memory for *len* elements in the list of *ints*.

*echo_area*

A pointer to an object of type Plimit that will contain the echo area of the device. Plimit is defined in phigs.h as follows:

```
typedef struct {
    Pfloat      x_min;      /* x min */
    Pfloat      x_max;      /* x max */
    Pfloat      y_min;      /* y min */
    Pfloat      y_max;      /* y max */
} Plimit;
```

*loc_data*

A pointer to a Ploc_data pointer. PHIGS will assign this pointer to the location in the Pstore structure that contains the device's data record contents. Ploc_data is defined in phigs.h as follows:

```
typedef struct {
    union {
        struct {
            Pint        unused;
        } pet_r1;
        struct {
            Pint        unused;
        } pet_r2;
        struct {
            Pint        unused;
        } pet_r3;
        struct {
            Pline_attrs         line_attrs;                         /* polyline attributes */
        } pet_r4;
        struct {
            Pline_fill_ctrl_flag                line_fill_ctrl_flag;    /* polyline/fill area ctrl flag */
            union {
                Pline_attrs                     line_attrs;         /* polyline attributes */
                Pint_attrs                      int_attrs;          /* interior attributes */
                struct {
                    Pint_attrs          int_attrs;          /* interior attributes */
                    Pedge_attrs         edge_attrs;         /* edge attributes */
                } fill_set;
            } attr;
        } pet_r5;
    } pets;
} Ploc_data;
```

**Execution**

INQUIRE DEFAULT LOCATOR DEVICE DATA returns the default data of the specified locator device. This data is stored in the workstation description table associated with the workstation type. See INITIALIZE LOCATOR for a description of the prompt/echo types, echo area, and data record contents, and how to set these values.

In the event of an error, the *error indicator* indicates the error number of the error detected and no other output data is returned. If no error is detected, the *error indicator* is set to zero, and the inquired information will be available in the output parameters. Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

052 Ignoring function, workstation type not recognized by the implementation

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

061 Ignoring function, specified workstation category is neither INPUT nor OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

**SEE ALSO**

INITIALIZE LOCATOR (3P)
INQUIRE DEFAULT LOCATOR DEVICE DATA 3 (3P)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

INQUIRE DEFAULT LOCATOR DEVICE DATA 3 – inquire the predefined locator data

**SYNOPSIS**

**C Syntax**

**void**
**pinq_def_loc_data3 ( type, device, store, error_ind, loc_pos, pets, echo_vol, loc_data )**

| | | |
|---|---|---|
| **Pint** | **type;** | *workstation type* |
| **Pint** | **device;** | *logical input device number* |
| **Pstore** | **store;** | *handle to Store object* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Ppoint3** | ***loc_pos;** | *OUT default initial position* |
| **Pint_list** | ****pets;** | *OUT list of prompt and echo types* |
| **Plimit3** | ***echo_vol;** | *OUT default echo volume* |
| **Ploc_data3** | ****loc_data;** | *OUT pointer to default data record* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE DEFAULT LOCATOR DEVICE DATA 3 to determine the following information for a LOCATOR input device associated with a given workstation type:

- Number and list of available prompt/echo types
- Default echo volume
- Default input data record
- Default initial locator position

Since the default prompt/echo type for all input devices is 1, the default input data record is for that prompt/echo type. There are no default input data records for prompt/echo types other than 1.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing portions of the device data. This buffer is passed as the *store* argument.

The store buffer is a data area managed by PHIGS. While the application is responsible for creating the initial buffer through a call to CREATE STORE, PHIGS manages this area such that there is sufficient memory for the specific inquiry. The locator device data record within the store buffer is accessed via the pointer returned in *loc_data*.

*type* The workstation type with which the device is associated.

*device* The device number of the locator device. See the *Available Devices* section of INITIALIZE LOCATOR 3 for a description of the available devices.

*store* The memory buffer PHIGS is to use for storing the information returned for the Ploc_data3 structure. This memory is not used for the data returned in the Pint_list structure. This buffer must exist prior to calling this function (see CREATE STORE (3P)).

**C Output Parameters**

*error_ind*

The error indicator. See the *Execution* section below for a description of its use. See the *Errors* section below for the possible values it may return.

*loc_pos* Default initial locator position.

*pets* List of available prompt/echo types. Pint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint            num_ints;       /* number of Pints in list */
```

```
    Pint        *ints;          /* list of integers    */
} Pint_list;
```

The *num_ints* component specifies the number of elements in the list. The *ints* component is a pointer to a list *num_ints* long.

The application must allocate memory for *len* elements in the list of *ints*.

*echo_vol*

A pointer to a Plimit3 structure defining the *x*, *y*, and *z* components of the echo volume, in Device Coordinates. Plimit3 is defined in phigs.h as follows:

```
typedef struct {
    Pfloat      x_min;      /* minimum x coordinate value */
    Pfloat      x_max;      /* maximum x coordinate value */
    Pfloat      y_min;      /* minimum y coordinate value */
    Pfloat      y_max;      /* maximum y coordinate value */
    Pfloat      z_min;      /* minimum z coordinate value */
    Pfloat      z_max;      /* maximum z coordinate value */
} Plimit3;
```

*loc_data*

A pointer to a Ploc_data3 pointer. PHIGS assigns this pointer to the location in the Pstore structure that contains the device's data record contents. Ploc_data3 is defined in phigs.h as follows:

```
typedef struct {
    union {
        struct {
                Pint        unused;
        } pet_r1;
        struct {
                Pint        unused;
        } pet_r2;
        struct {
                Pint        unused;
        } pet_r3;
        struct {
                Pline_attrs         line_attrs;                         /* polyline attributes */
        } pet_r4;
        struct {
                Pline_fill_ctrl_flag    line_fill_ctrl_flag;            /* polyline/fill area ctrl flag */
                union {
                        Pline_attrs             line_attrs;             /* polyline attributes */
                        Pint_attrs              int_attrs;              /* interior attributes */
                        struct {
                                Pint_attrs                              int_attrs;/* interior attributes */
                                Pedge_attrs                             edge_attrs;/* edge attributes */
                        } fill_set;
                } attr;
        } pet_r5;
    } pets;
} Ploc_data3;
```

**Execution**

INQUIRE DEFAULT LOCATOR DEVICE DATA 3 returns the default data of the specified locator device. This data is stored in the workstation description table associated with the workstation type. See INITIALIZE LOCATOR 3 for a description of the prompt/echo types, echo volume, and data record contents, and how to set these values.

In the event of an error, the *error indicator* indicates the error number of the error detected and no other output data is returned. If no error is detected, the *error indicator* is set to zero, and the inquired information is available in the output parameters. Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

052 Ignoring function, workstation type not recognized by the implementation

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

**SEE ALSO**

INITIALIZE LOCATOR 3 (3P)
INQUIRE DEFAULT LOCATOR DEVICE DATA (3P)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

INQUIRE DEFAULT PICK DEVICE DATA – inquire the predefined pick data

**SYNOPSIS**

**C Syntax**

**void**
**pinq_def_pick_data ( type, device, store, error_ind, pets, echo_area, pick_data)**

| | | |
|---|---|---|
| **Pint** | **type;** | *workstation type* |
| **Pint** | **device;** | *logical input device number* |
| **Pstore** | **store;** | *handle to Store object* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pint_list** | ****pets;** | *OUT list of prompt and echo types* |
| **Plimit** | ***echo_area;** | *OUT default echo volume* |
| **Ppick_data** | ****pick_data;** | *OUT pointer to default data record* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE DEFAULT PICK DEVICE DATA to determine the following information for a PICK input device associated with a given workstation type:

- Number and list of available prompt/echo types
- Default echo area
- Default input data record

Since the default prompt/echo type for all input devices is 1, the default input data record is for that prompt/echo type. There are no default input data records for prompt/echo types other than 1.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing portions of the device data. This buffer is passed as the *store* argument.

The store buffer is a data area managed by PHIGS. While the application is responsible for creating the initial buffer through a call to CREATE STORE, PHIGS manages this area such that there is sufficient memory for the specific inquiry. The pick device data record within the store buffer is accessed via the pointer returned in *pick_data*.

*type* The workstation type with which the device is associated.

*device* The device number of the pick device. See the *Available Devices* section of INITIALIZE PICK for a description of the available devices.

*store* The memory buffer PHIGS is to use for storing the information returned for the Ppick_data structure. This memory is not used for the data returned in the Pint_list structure. This buffer must exist prior to calling this function (see CREATE STORE (3P)).

**C Output Parameters**

*error_ind*

The error indicator. See the *Execution* section below for a description of its use. See the *Errors* section below for the possible values it may return.

*pets* Pointer to available prompt/echo types. Pint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint        num_ints;       /* number of Pints in list */
        Pint        *ints;          /* list of integers        */
} Pint_list;
```

The *num_ints* component specifies the number of elements in the list. The *ints* component is a

pointer to a list *num_ints* long.

The application must allocate memory for *len* elements in the list of *ints*.

*echo_area*

A pointer to an object of type Plimit that will represent the echo area of the device. Plimit is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          x_min;          /* x min */
        Pfloat          x_max;          /* x max */
        Pfloat          y_min;          /* y min */
        Pfloat          y_max;          /* y max */
} Plimit;
```

*pick_data*

A pointer to a pointer to a Ppick_data pointer. PHIGS assigns this pointer to the location in the Pstore structure that contains the device's data record contents. Ppick_data is defined in phigs.h as follows:

```
typedef struct {
    union {
        struct {
                        Pint            unused;
        } pet_r1;
    } pets;
} Ppick_data;
```

**Execution**

INQUIRE DEFAULT PICK DEVICE DATA returns the default data of the specified pick device. This data is stored in the workstation description table associated with the workstation type. See INITIALIZE PICK for a description of the prompt/echo types, echo area, and data record contents, and how to set these values.

In the event of an error, the *error indicator* indicates the error number of the error detected, and no other output data is returned. If no error is detected, the *error indicator* is set to zero, and the inquired information will be available in the output parameters. Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

052 Ignoring function, workstation type not recognized by the implementation

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

**SEE ALSO**

INITIALIZE PICK (3P)
INQUIRE DEFAULT PICK DEVICE DATA 3 (3P)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

INQUIRE DEFAULT PICK DEVICE DATA 3 – inquire the predefined pick data

**SYNOPSIS**

**C Syntax**

**void**
**pinq_def_pick_data3 ( type, device, store, error_ind, pets, echo_vol, pick_data)**

| | | |
|---|---|---|
| **Pint** | **type;** | *workstation type* |
| **Pint** | **device;** | *logical input device number* |
| **Pstore** | **store;** | *handle to Store object* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pint_list** | ****pets;** | *OUT list of prompt and echo types* |
| **Plimit3** | ***echo_vol;** | *OUT default echo volume* |
| **Ppick_data3** | ****pick_data;** | *OUT pointer to default data record* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE DEFAULT PICK DEVICE DATA 3 to determine the following information for a PICK input device associated with a given workstation type:

- Number and list of available prompt/echo types
- Default echo volume
- Default input data record

Since the default prompt/echo type for all input devices is 1, the default input data record is for that prompt/echo type. There are no default input data records for prompt/echo types other than 1.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing portions of the device data. This buffer is passed as the *store* argument.

The store buffer is a data area managed by PHIGS. While the application is responsible for creating the initial buffer through a call to CREATE STORE, PHIGS manages this area such that there is sufficient memory for the specific inquiry. The pick device data record within the store buffer is accessed via the pointer returned in *pick_data*.

*type* The workstation type with which the device is associated.

*device* The device number of the pick device. See the *Available Devices* section of INITIALIZE PICK 3 for a description of the available devices.

*store* The memory buffer PHIGS is to use for storing the information returned for the Ppick_data3 structure. This memory is not used for the data returned in the Pint_list structure. This buffer must exist prior to calling this function (see CREATE STORE (3P)).

*error_ind*
The error indicator. See the *Execution* section below for a description of its use. See the *Errors* section below for the possible values it may return.

*pets* A pointer to the available prompt/echo types. Pint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint        num_ints;       /* number of Pints in list */
        Pint        *ints;          /* list of integers        */
} Pint_list;
```

The *num_ints* component specifies the number of elements in the list. The *ints* component is a pointer to a list *num_ints* long.

The application must allocate memory for *len* elements in the list of *ints*.

*echo_vol*

A pointer to a Plimit3 structure defining the *x*, *y*, and *z* components of the echo volume, in Device Coordinates. Plimit3 is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          x_min;          /* minimum x coordinate value */
        Pfloat          x_max;          /* maximum x coordinate value */
        Pfloat          y_min;          /* minimum y coordinate value */
        Pfloat          y_max;          /* maximum y coordinate value */
        Pfloat          z_min;          /* minimum z coordinate value */
        Pfloat          z_max;          /* maximum z coordinate value */
} Plimit3;
```

*pick_data*

A pointer to a Ppick_data3 pointer. PHIGS assigns this pointer to the location in the Pstore structure that contains the device's data record contents. Ppick_data3 is defined in phigs.h as follows:

```
typedef struct {
    union {
        struct {
                        Pint            unused;
        } pet_r1;
    } pets;
} Ppick_data3;
```

**Execution**

INQUIRE DEFAULT PICK DEVICE DATA 3 returns the default data of the specified pick device. This data is stored in the workstation description table associated with the workstation type. See INITIALIZE PICK 3 for a description of the prompt/echo types, echo volume and data record contents and how to set these values.

In the event of an error, the *error indicator* indicates the error number of the error detected, and no other output data is returned. If no error is detected, the *error indicator* is set to zero, and the inquired information will be available in the output parameters. Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

052 Ignoring function, workstation type not recognized by the implementation

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

**SEE ALSO**

INITIALIZE PICK 3 (3P)
INQUIRE DEFAULT PICK DEVICE DATA (3P)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

INQUIRE DEFAULT STRING DEVICE DATA – inquire the predefined string data

**SYNOPSIS**

**C Syntax**

**void**
**pinq_def_string_data ( type, device, store, error_ind, max_buf_size, pets, echo_area, string_data )**

| | | |
|---|---|---|
| **Pint** | **type;** | *workstation type* |
| **Pint** | **device;** | *logical input device number* |
| **Pstore** | **store;** | *handle to Store object* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pint** | ***max_buf_size;** | *OUT max. input buffer size* |
| **Pint_list** | ****pets;** | *OUT list of prompt and echo types* |
| **Plimit** | ***echo_area;** | *OUT default echo volume* |
| **Pstring_data** | ****string_data;** | *OUT pointer to default data record* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE DEFAULT STRING DEVICE DATA to determine the following information for a STRING input device associated with a given workstation type:

- Number and list of available prompt/echo types
- Default echo area
- Default input data record
- Maximum string buffer size

Since the default prompt/echo type for all input devices is 1, the default input data record is for that prompt/echo type. There are no default input data records for prompt/echo types other than 1.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing portions of the device data. This buffer is passed as the *store* argument.

The store buffer is a data area managed by PHIGS. While the application is responsible for creating the initial buffer through a call to CREATE STORE, PHIGS manages this area such that there is sufficient memory for the specific inquiry. The string device data record within the store buffer is accessed via the pointer returned in *string_data*.

*type* The workstation type with which the device is associated.

*device* The device number of the string device. See the *Available Devices* section of INITIALIZE STRING for a description of the available devices.

*store* The memory buffer PHIGS is to use for storing the information returned for the Pstring_data structure. This memory is not used for the data returned in the Pint_list structure. This buffer must exist prior to calling this function (see CREATE STORE (3P)).

**C Output Parameters**

*error_ind*

The error indicator. See the *Execution* section below for a description of its use. See the *Errors* section below for the possible values it may return.

*max_buf_size*

Maximum input buffer size.

*pets* A pointer to the available prompt/echo types. Pint_list is defined in phigs.h as follows:

typedef struct {

```
        Pint        num_ints;       /* number of Pints in list */
        Pint        *ints;          /* list of integers        */
} Pint_list;
```

The *num_ints* component specifies the number of elements in the list. The *ints* component is a pointer to a list *num_ints* long.

The application must allocate memory for *len* elements in the list of *ints*.

*echo_area*

A pointer to an object of type Plimit that represents the echo area of the device. Plimit is defined in phigs.h as follows:

```
typedef struct {
        Pfloat      x_min;          /* x min */
        Pfloat      x_max;          /* x max */
        Pfloat      y_min;          /* y min */
        Pfloat      y_max;          /* y max */
} Plimit;
```

*string_data*

A pointer to a Pstring_data pointer. PHIGS assigns this pointer to the location in the Pstore structure that contains the device's data record contents. Pstring_data is defined in phigs.h as follows:

```
typedef struct {
        Pint    buffer_size;    /* input buffer size */
        Pint    init_pos;       /* initial editing position */
        union {
                struct {
                        Pint    unused;
                } pet_r1;
        } pets;
} Pstring_data;
```

**Execution**

INQUIRE DEFAULT STRING DEVICE DATA returns the default data of the specified string device. This data is stored in the workstation description table associated with the workstation type. See INITIALIZE STRING for a description of the prompt/echo types, echo area, and data record contents, and how to set these values.

In the event of an error, the *error indicator* indicates the error number of the error detected, and no other output data is returned. If no error is detected, the *error indicator* is set to zero, and the inquired information will be available in the output parameters. Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

052 Ignoring function, workstation type not recognized by the implementation

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

**SEE ALSO**

INITIALIZE STRING (3P)

INQUIRE DEFAULT STRING DEVICE DATA 3 (3P)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

INQUIRE DEFAULT STRING DEVICE DATA 3 – inquire the predefined string data

**SYNOPSIS**

**C Syntax**

**void**
**pinq_def_string_data3 ( type, device, store, error_ind, max_buf_size, pets, echo_vol, string_data )**

| | | |
|---|---|---|
| **Pint** | **type;** | *workstation type* |
| **Pint** | **device;** | *logical input device number* |
| **Pstore** | **store;** | *handle to Store object* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pint** | ***max_buf_size;** | *OUT max. input buffer size* |
| **Pint_list** | ****pets;** | *OUT list of prompt and echo types* |
| **Plimit3** | ***echo_vol;** | *OUT default echo volume* |
| **Pstring_data3** | ****string_data;** | *OUT pointer to default data record* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE DEFAULT STRING DEVICE DATA 3 to determine the following information for a STRING input device associated with a given workstation type:

- Number and list of available prompt/echo types
- Default echo volume
- Default input data record
- Maximum string buffer size

Since the default prompt/echo type for all input devices is 1, the default input data record is for that prompt/echo type. There are no default input data records for prompt/echo types other than 1.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing portions of the device data. This buffer is passed as the *store* argument.

The store buffer is a data area managed by PHIGS. While the application is responsible for creating the initial buffer through a call to CREATE STORE, PHIGS manages this area such that there is sufficient memory for the specific inquiry. The string device data record within the store buffer is accessed via the pointer returned in *string_data*.

*type* The workstation type with which the device is associated.

*device* The device number of the string device. See the *Available Devices* section of INITIALIZE STRING 3 for a description of the available devices.

*store* The memory buffer PHIGS is to use for storing the information returned for the Pstring_data3 structure. This memory is not used for the data returned in the Pint_list structure. This buffer must exist prior to calling this function (see CREATE STORE (3P)).

**C Output Parameters**

*error_ind*

The error indicator. See the *Execution* section below for a description of its use. See the *Errors* section below for the possible values it may return.

*max_buf_size*

Maximum input buffer size.

*pets* A pointer to the available prompt/echo types. Pint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint            num_ints;       /* number of Pints in list */
        Pint            *ints;          /* list of integers */
} Pint_list;
```

The *num_ints* component specifies the number of elements in the list. The *ints* component is a pointer to a list *num_ints* long.

The application must allocate memory for *len* elements in the list of *ints*.

*echo_vol*

A pointer to a Plimit3 structure defining the *x*, *y*, and *z* components of the echo volume, in Device Coordinates. Plimit3 is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          x_min;          /* minimum x coordinate value */
        Pfloat          x_max;          /* maximum x coordinate value */
        Pfloat          y_min;          /* minimum y coordinate value */
        Pfloat          y_max;          /* maximum y coordinate value */
        Pfloat          z_min;          /* minimum z coordinate value */
        Pfloat          z_max;          /* maximum z coordinate value */
} Plimit3;
```

*string_data*

A pointer to a Pstring_data3 pointer. PHIGS assigns this pointer to the location in the Pstore structure that contains the device's data record contents. Pstring_data3 is defined in phigs.h as follows:

```
typedef struct {
        Pint    buffer_size;    /* input buffer size */
        Pint    init_pos;       /* initial editing position */
        union {
                struct {
                        Pint    unused;
                } pet_r1;
        } pets;
} Pstring_data3;
```

**Execution**

INQUIRE DEFAULT STRING DEVICE DATA 3 returns the default data of the specified string device. This data is stored in the workstation description table associated with the workstation type. See INITIALIZE STRING 3 for a description of the prompt/echo types, echo volume, and data record contents, and how to set these values.

In the event of an error, the *error indicator* indicates the error number of the error detected and no other output data is returned. If no error is detected, the *error indicator* is set to zero, and the inquired information will be available in the output parameters. Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

052 Ignoring function, workstation type not recognized by the implementation

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

**SEE ALSO**

INITIALIZE STRING 3 (3P)
INQUIRE DEFAULT STRING DEVICE DATA (3P)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

INQUIRE DEFAULT STROKE DEVICE DATA – inquire the predefined stroke data

**SYNOPSIS**

**C Syntax**

**void**
**pinq_def_stroke_data ( type, device, store, error_ind, max_buf_size, pets, echo_area, stroke_data )**

| | | |
|---|---|---|
| **Pint** | **type;** | *workstation type* |
| **Pint** | **device;** | *logical input device number* |
| **Pstore** | **store;** | *handle to Store object* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pint** | ***max_buf_size;** | *OUT max. input buffer size* |
| **Pint_list** | ****pets;** | *OUT list of prompt and echo types* |
| **Plimit** | ***echo_area;** | *OUT default echo volume* |
| **Pstroke_data** | ****stroke_data;** | *OUT pointer to default data record* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE DEFAULT STROKE DEVICE DATA to determine the following information for a STROKE input device associated with a given workstation type:

- Number and list of available prompt/echo types
- Default echo area
- Default input data record
- Maximum stroke buffer size

Since the default prompt/echo type for all input devices is 1, the default input data record is for that prompt/echo type. There are no default input data records for prompt/echo types other than 1.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing portions of the device data. This buffer is passed as the *store* argument.

The store buffer is a data area managed by PHIGS. While the application is responsible for creating the initial buffer through a call to CREATE STORE, PHIGS manages this area such that there is sufficient memory for the specific inquiry. The stroke device data record within the store buffer is accessed via the pointer returned in *stroke_data*.

*type* The workstation type with which the device is associated.

*device* The device number of the stroke device. See the *Available Devices* section of INITIALIZE STROKE for a description of the available devices.

*store* The memory buffer PHIGS is to use for storing the information returned for the Pstroke_data structure. This memory is not used for the data returned in the Pint_list structure. This buffer must exist prior to calling this function (see CREATE STORE (3P)).

**C Output Parameters**

*error_ind*

The error indicator. See the *Execution* section below for a description of its use. See the *Errors* section below for the possible values it may return.

*max_buf_size*

Maximum buffer size.

*pets* A pointer to a list of the available prompt/echo type. Pint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint            num_ints;       /* number of Pints in list */
        Pint            *ints;          /* list of integers        */
} Pint_list;
```

The *num_ints* component specifies the number of elements in the list. The *ints* component is a pointer to a list *num_ints* long.

The application must allocate memory for *len* elements in the list of *ints*.

*echo_area*

A pointer to an object of type Plimit that contains the echo area of the device. Plimit is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          x_min;          /* x min */
        Pfloat          x_max;          /* x max */
        Pfloat          y_min;          /* y min */
        Pfloat          y_max;          /* y max */
} Plimit;
```

*stroke_data*

A pointer to a Pstroke_data pointer. PHIGS will assign this pointer to the location in the Pstore structure that contains the device's data record contents. Pstroke_data is defined in phigs.h as follows:

```
typedef struct {
        Pint    buffer_size;    /* input buffer size */
        Pint    init_pos;       /* initial editing position */
        Pfloat  x_interval;     /* x motion interval */
        Pfloat  y_interval;     /* y motion interval */
        Pfloat  time_interval;  /* time interval */
        union {
                struct {
                        Pint    unused;
                } pet_r1;
                struct {
                        Pint    unused;
                } pet_r2;
                struct {
                        Pmarker_attrs   marker_attrs;   /* marker attributes */
                } pet_r3;
                struct {
                        Pline_attrs     line_attrs;     /* line attributes */
                } pet_r4;
        } pets;
} Pstroke_data;
```

**Execution**

INQUIRE DEFAULT STROKE DEVICE DATA returns the default data of the specified stroke device. This data is stored in the workstation description table associated with the workstation type. See INITIALIZE STROKE for a description of the prompt/echo types, echo area, and data record contents, and how to set these values.

In the event of an error, the *error indicator* indicates the error number of the error detected, and no other output data is returned. If no error is detected, the *error indicator* is set to zero, and the inquired

information will be available in the output parameters. Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

052 Ignoring function, workstation type not recognized by the implementation

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

**SEE ALSO**

INITIALIZE STROKE (3P)
INQUIRE DEFAULT STROKE DEVICE DATA 3 (3P)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

INQUIRE DEFAULT STROKE DEVICE DATA 3 – inquire the predefined stroke data

**SYNOPSIS**

**C Syntax**

**void**
**pinq_def_stroke_data3 ( type, device, store, error_ind, max_buf_size, pets, echo_volume, stroke_data )**

| | | |
|---|---|---|
| **Pint** | **type;** | *workstation type* |
| **Pint** | **device;** | *logical input device number* |
| **Pstore** | **store;** | *handle to Store object* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pint** | ***max_buf_size;** | *OUT max. input buffer size* |
| **Pint_list** | ****pets;** | *OUT list of prompt and echo types* |
| **Plimit3** | ***echo_volume;** | *OUT default echo volume* |
| **Pstroke_data3** | ****stroke_data;** | *OUT default data record* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE DEFAULT STROKE DEVICE DATA 3 to determine the following information for a STROKE input device associated with a given workstation type:

- Number and list of available prompt/echo types
- Default echo volume
- Default input data record
- Maximum stroke buffer size

Since the default prompt/echo type for all input devices is 1, the default input data record is for that prompt/echo type. There are no default input data records for prompt/echo types other than 1.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing portions of the device data. This buffer is passed as the *store* argument.

The store buffer is a data area managed by PHIGS. While the application is responsible for creating the initial buffer through a call to CREATE STORE, the implementation manages this area such that there is sufficient memory for the specific inquiry. The stroke device data record within the store buffer is accessed via the pointer pointed to by *stroke_data*.

*type* The workstation type with which the device is associated.

*device* The device number of the stroke device. See the *Available Devices* section of INITIALIZE STROKE 3 for a description of the available devices.

*store* The memory buffer PHIGS is to use for storing the information returned for the Pstroke_data3 structure. This memory is not used for the data returned in the Pint_list structure. This buffer must exist prior to calling this function (see CREATE STORE (3P)).

**C Output Parameters**

*error_ind*

The error indicator. See the *Execution* section below for a description of its use. See the *Errors* section below for the possible values it may return.

*max_buf_size*

Maximum buffer size.

*pets* A pointer to the default prompt/echo types. Pint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint            num_ints;       /* number of Pints in list */
        Pint            *ints;          /* list of integers        */
} Pint_list;
```

The *num_ints* component specifies the number of elements in the list. The *ints* component is a pointer to a list *num_ints* long.

The application must allocate memory for *len* elements in the list of *ints*.

*echo_volume*

A pointer to a Plimit3 structure defining the *x*, *y*, and *z* components of the echo volume, in Device Coordinates. Plimit3 is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          x_min;          /* minimum x coordinate value */
        Pfloat          x_max;          /* maximum x coordinate value */
        Pfloat          y_min;          /* minimum y coordinate value */
        Pfloat          y_max;          /* maximum y coordinate value */
        Pfloat          z_min;          /* minimum z coordinate value */
        Pfloat          z_max;          /* maximum z coordinate value */
} Plimit3;
```

*stroke_data*

A pointer to a pointer to a Pstroke_data3 structure. Pstroke_data3 is defined in phigs.h as follows:

```
typedef struct {
        Pint    buffer_size;            /* input buffer size */
        Pint    init_pos;               /* initial editing position */
        Pfloat  x_interval;             /* x interval */
        Pfloat  y_interval;             /* y interval */
        Pfloat  z_interval;             /* z interval */
        Pfloat  time_interval;          /* time interval */
        union {
                struct {
                        Pint            unused;
                } pet_r1;
                struct {
                        Pint            unused;
                } pet_r2;
                struct {
                        Pmarker_attrs   marker_attrs;   /* marker attributes */
                } pet_r3;
                struct {
                        Pline_attrs     line_attrs;     /* line attributes */
                } pet_r4;
        } pets;
} Pstroke_data3;
```

**Execution**

INQUIRE DEFAULT STROKE DEVICE DATA 3 returns the default data of the specified stroke device. This data is stored in the workstation description table associated with the workstation type. See INITIALIZE STROKE 3 for a description of the prompt/echo types, echo volume and data record contents and how to set these values.

In the event of an error, the *error indicator* will indicate the error number of the error detected and no other output data will be returned. If no error is detected, the *error indicator* will be set to zero and the inquired information will be available in the output parameters. Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

052 Ignoring function, workstation type not recognized by the implementation

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

**SEE ALSO**

INITIALIZE STROKE 3 (3P)
INQUIRE DEFAULT STROKE DEVICE DATA (3P)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

INQUIRE DEFAULT VALUATOR DEVICE DATA – inquire the predefined valuator data

**SYNOPSIS**

**C Syntax**

**void**
**pinq_def_val_data ( type, device, store, error_ind, def_value, pets,**
**echo_area, val_data )**

| | | |
|---|---|---|
| **Pint** | **type;** | *workstation type* |
| **Pint** | **device;** | *logical input device number* |
| **Pstore** | **store;** | *handle to Store object* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pfloat** | ***def_value;** | *OUT default initial value* |
| **Pint_list** | ****pets;** | *OUT list of prompt and echo types* |
| **Plimit** | ***echo_area;** | *OUT default echo volume* |
| **Pval_data** | ****val_data;** | *OUT pointer to default data record* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE DEFAULT VALUATOR DEVICE DATA to determine the following information for a VALUATOR input device associated with a given workstation type:

- Number and list of available prompt/echo types
- Default echo area
- Default input data record
- Default initial valuator value

Since the default prompt/echo type for all input devices is 1, the default input data record is for that prompt/echo type. There are no default input data records for prompt/echo types other than 1.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing portions of the device data. This buffer is passed as the *store* argument.

The store buffer is a data area managed by PHIGS. While the application is responsible for creating the initial buffer through a call to CREATE STORE, PHIGS manages this area such that there is sufficient memory for the specific inquiry. The valuator device data record within the store buffer is accessed via the pointer returned in *val_data*.

*type* The workstation type with which the device is associated.

*device* The device number of the valuator device. See the *Available Devices* section of INITIALIZE VALUATOR for a description of the available devices.

*store* The memory buffer PHIGS is to use for storing the information returned for the Pval_data structure. This memory is not used for the data returned in the Pint_list structure. This buffer must exist prior to calling this function (see CREATE STORE (3P)).

**C Output Parameters**

*error_ind*

The error indicator. See the *Execution* section below for a description of its use. See the *Errors* section below for the possible values it may return.

*def_value*

Default initial valuator value.

*pets* A pointer to the available prompt/echo types. Pint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint            num_ints;       /* number of Pints in list */
        Pint            *ints;          /* list of integers        */
} Pint_list;
```

The *num_ints* component specifies the number of elements in the list. The *ints* component is a pointer to a list *num_ints* long.

The application must allocate memory for *len* elements in the list of *ints*.

*echo_area*

A pointer to an object of type Plimit that contains the echo area of the device. Plimit is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          x_min;          /* x min */
        Pfloat          x_max;          /* x max */
        Pfloat          y_min;          /* y min */
        Pfloat          y_max;          /* y max */
} Plimit;
```

*val_data*

A pointer to a Pval_data pointer. PHIGS will assign this pointer to the location in the Pstore structure that contains the device's data record contents. Pval_data is defined in phigs.h as follows:

```
typedef struct {
        Pfloat  low;    /* low range limit */
        Pfloat  high;   /* high range limit */
        union {
                struct {
                        Pint            unused;
                } pet_r1;
                struct {
                        char            *label;
                        char            *format;
                        char            *low_label;
                        char            *high_label;
                } pet_u1;
        } pets;
} Pval_data;
```

**Execution**

INQUIRE DEFAULT VALUATOR DEVICE DATA returns the default data of the specified valuator device. This data is stored in the workstation description table associated with the workstation type. See INITIALIZE VALUATOR for a description of the prompt/echo types, echo area and data record contents and how to set these values.

In the event of an error, the *error indicator* indicates the error number of the error detected, and no other output data will be returned. If no error is detected, the *error indicator* is set to zero, and the inquired information will be available in the output parameters. Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

052 Ignoring function, workstation type not recognized by the implementation

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

**SEE ALSO**

INITIALIZE VALUATOR (3P)
INQUIRE DEFAULT VALUATOR DEVICE DATA 3 (3P)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

INQUIRE DEFAULT VALUATOR DEVICE DATA 3 – inquire the predefined valuator data

**SYNOPSIS**

**C Syntax**

**void**
**pinq_def_val_data3 ( type, device, store, error_ind, def_value, pets, echo_vol, val_data )**

| | | |
|---|---|---|
| **Pint** | **type;** | *workstation type* |
| **Pint** | **device;** | *logical input device number* |
| **Pstore** | **store;** | *handle to Store object* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pfloat** | ***def_value;** | *OUT default initial value* |
| **Pint_list** | ****pets;** | *OUT list of prompt and echo types* |
| **Plimit3** | ***echo_vol;** | *OUT default echo volume* |
| **Pval_data3** | ****val_data;** | *OUT pointer to default data record* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE DEFAULT VALUATOR DEVICE DATA 3 to determine the following information for a VALUATOR input device associated with a given workstation type:

- Number and list of available prompt/echo types
- Default echo volume
- Default input data record
- Default initial valuator value

Since the default prompt/echo type for all input devices is 1, the default input data record is for that prompt/echo type. There are no default input data records for prompt/echo types other than 1.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing portions of the device data. This buffer is passed as the *store* argument.

The store buffer is a data area managed by PHIGS. While the application is responsible for creating the initial buffer through a call to CREATE STORE, PHIGS manages this area such that there is sufficient memory for the specific inquiry. The valuator device data record within the store buffer is accessed via the pointer returned in *val_data*.

*type* The workstation type with which the device is associated.

*device* The device number of the valuator device. See the *Available Devices* section of INITIALIZE VALUATOR 3 for a description of the available devices.

*store* The memory buffer PHIGS is to use for storing the information returned for the Pval_data3 structure. This memory is not used for the data returned in the Pint_list structure. This buffer must exist prior to calling this function (see CREATE STORE (3P)).

**C Output Parameters**

*error_ind*

The error indicator. See the *Execution* section below for a description of its use. See the *Errors* section below for the possible values it may return.

*def_value*

Default initial valuator value.

*pets* A pointer to available prompt/echo types. Pint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint            num_ints;       /* number of Pints in list */
        Pint            *ints;          /* list of integers        */
} Pint_list;
```

The *num_ints* component specifies the number of elements in the list. The *ints* component is a pointer to a list *num_ints* long.

The application must allocate memory for *len* elements in the list of *ints*.

*echo_vol*

A pointer to a Plimit3 structure defining the *x*, *y*, and *z* components of the echo volume, in Device Coordinates. Plimit3 is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          x_min;          /* minimum x coordinate value */
        Pfloat          x_max;          /* maximum x coordinate value */
        Pfloat          y_min;          /* minimum y coordinate value */
        Pfloat          y_max;          /* maximum y coordinate value */
        Pfloat          z_min;          /* minimum z coordinate value */
        Pfloat          z_max;          /* maximum z coordinate value */
} Plimit3;
```

*val_data*

A pointer to a Pval_data3 pointer. PHIGS will assign this pointer to the location in the Pstore structure that contains the device's data record contents. Pval_data3 is defined in phigs.h as follows:

```
typedef struct {
        Pfloat  low;    /* low range limit */
        Pfloat  high;   /* high range limit */
        union {
                struct {
                        Pint            unused;
                } pet_r1;
                struct {
                        char            *label;
                        char            *format;
                        char            *low_label;
                        char            *high_label;
                } pet_u1;
        } pets;
} Pval_data3;
```

**Execution**

INQUIRE DEFAULT VALUATOR DEVICE DATA 3 returns the default data of the specified valuator device. This data is stored in the workstation description table associated with the workstation type. See INITIALIZE VALUATOR 3 for a description of the prompt/echo types, echo volume, and data record contents, and how to set these values.

In the event of an error, the *error indicator* indicates the error number of the error detected, and no other output data is returned. If no error is detected, the *error indicator* is set to zero, and the inquired information will be available in the output parameters. Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

052 Ignoring function, workstation type not recognized by the implementation

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

**SEE ALSO**

INITIALIZE VALUATOR 3 (3P)
INQUIRE DEFAULT VALUATOR DEVICE DATA (3P)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

INQUIRE DEPTH CUE FACILITIES – inquire depth cue facilities for the specified workstation type

**SYNOPSIS**

**C Syntax**

```
void
pinq_dcue_facs ( type, error_ind, facilities )
Pint        type;          workstation type
Pint        *error_ind;    OUT error indicator
Pint        *facilities;   OUT depth cue facilities
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

INQUIRE DEPTH CUE FACILITIES returns the number of predefined depth cure indices associated with the specified workstation type.

**C Input Parameter**

*type* The workstation type.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*facilities*

A pointer to the location to store the number of predefined indices.

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the following error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

051 Ignoring function, this information is not yet available for this generic workstation type; open a workstation of this type and use the specific workstation type

052 Ignoring function, workstation type not recognized by the implementation

059 Ignoring function, the specified workstation does not have output capability (i.e. the workstation category is neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

**SEE ALSO**

SET DEPTH CUE REPRESENTATION (3P+)
SET DEPTH CUE INDEX (3P+)
PHIGS WORKSTATION DESCRIPTION TABLE (7P)

**NAME**

INQUIRE DEPTH CUE REPRESENTATION – inquire depth cue representation for a given workstation

**SYNOPSIS**

**C Syntax**

**void**
**pinq_dcue_rep ( ws, index, type, error_ind, rep )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **index;** | *polyline index* |
| **Pinq_type** | **type;** | *type of returned value* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pdcue_bundle** | ***rep;** | *OUT depth cue representation* |

**Required PHIGS Operating States**

(PHOP,WSOP, *, *)

**DESCRIPTION**

**Purpose**

INQUIRE DEPTH CUE REPRESENTATION returns the attribute values of a specified entry in the workstation table of depth cue representations. See SET DEPTH CUE REPRESENTATION for a full description of the contents of the bundle table entry and a description of the depth cue representation table.

**C Input Parameters**

*ws* The workstation identifier.

*index* The depth cue index.

*type* An enumerated value specifying whether the values to be returned are those originally specified by the application (PINQ_SET), or those resulting after PHIGS mapped them to ones available on the workstation (PINQ_REALIZED). A Pinq_type structure is defined as:

```
typedef enum {
        PINQ_SET,
        PINQ_REALIZED
} Pinq_type;
```

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*rep* A pointer to a Pdcue_bundle data structure containing attribute values defining a depth cue representation. A Pdcue_bundle structure is defined as:

```
typedef struct {
        Pdcue_mode      mode;           /* depth cue mode */
        Pfloat          ref_planes[2];  /* depth cue reference planes */
        Pfloat          scaling[2];     /* depth cue scaling */
        Pgcolr          colr;           /* depth cue colour */
} Pdcue_bundle;
```

Pdcue_mode is defined as:

```
typedef enum {
        PSUPPRESSED,
        PALLOWED
} Pdcue_mode;
```

*ref_planes* and *scaling* entries of 0 are back reference plane and corresponding scaling factors.

*ref_planes* and *scaling* entries of 1 are front reference plane and corresponding scaling factors. These scaling factors define the portion of the primitive colour that should be combined with the depth cue colour, as a function of *z* in NPC.

Pgcolr is defined as:

```
typedef struct {
        Pint                type;       /* indirect, RGB, CIE, HSV, HLS */
        union {
            Pint            ind;        /* index in workstation colour bundle table */
            struct {
                Pfloat      x;          /* red, hue, etc. */
                Pfloat      y;          /* green, saturation, lightness, etc. */
                Pfloat      z;          /* blue, value, saturation, etc. */
            } general;
        } val;
} Pgcolr;
```

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the specified depth cue index is not present in the depth cue table on the workstation and the specified type of returned value is *Realized*, the representation for the depth cue index zero is returned.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the following error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

003 Ignoring function, function requires state (PHOP,WSOP,*,*)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT, OUTIN, nor MO)

101 Ignoring function, the specified representation has not been defined

603 Ignoring function, the depth cue index is less than zero

**SEE ALSO**

SET DEPTH CUE REPRESENTATION (3P+)
SET DEPTH CUE INDEX (3P+)
INQUIRE LIST OF DEPTH CUE INDICES (3P+)
INQUIRE PREDEFINED DEPTH CUE REPRESENTATION (3P+)

**NAME**

INQUIRE DIRECT COLOUR MODEL FACILITIES – inquire list of workstation direct colour facilities

**SYNOPSIS**

**C Syntax**

**void**
**pinq_direct_colr_model_facs (type, length, start, error_ind, models, total_length )**

| | | |
|---|---|---|
| **Pint** | **type;** | *workstation type* |
| **Pint** | **length;** | *length of application list* |
| **Pint** | **start;** | *starting position* |
| **Pint** | ∗**error_ind;** | *OUT error indicator* |
| **Pint_list** | ∗**models;** | *OUT list of colour models* |
| **Pint** | ∗**total_length;** | *OUT length of list in PHIGS* |

**Required PHIGS Operating States**

(PHOP, ∗, ∗, ∗)

**DESCRIPTION**

**Purpose**

INQUIRE DIRECT COLOUR MODEL FACILITIES inquires for a list of workstation direct colour facilities.

**C Input Parameters**

All of the following data types are predefined in phigs.h.

*type* The workstation type.

*length* The number of *ints* in the *models* output parameter for which the application has allocated memory. *length* is the number of list elements that the system can return in *models→ints*. If a value of 0 is used here, no data will be returned in the *models→ints* list, but the total number of elements will be returned in *total_length*.

*start* Starting position of inquiry. The elements in the list, beginning with the item number specified by *start*, are copied sequentially into *models→ints* until *models→ints* is full or all the elements have been copied.

**C Output Parameters**

*error_ind*

A pointer to the location for storing the error number of any error detected by this function.

*models* A pointer to a Pint_list that returns the portion of the list of available colour models from the workstation description table, starting with *start*. Pint_list is defined in phigs.h as:

```
typedef struct {
        Pint    num_ints;   /* number of integers */
        Pint    *ints;      /* list of integers */
} Pint_list;
```

Constants defined for colour model are:

| | | |
|---|---|---|
| 0 | PINDIRECT | |
| 1 | PMODEL_RGB | *Red, Green, Blue* |
| 2 | PMODEL_CIELUV | *CIE* |
| 3 | PMODEL_HSV | *Hue, Saturation, Value* |
| 4 | PMODEL_HLS | *Hue, Lightness, Saturation* |

The pointer to the list of integers must be initialized to an array of *length* Pint elements.

*total_length*

A pointer to an integer in which the system returns the total number of elements in the list. This is the value required for *length* if all elements in the list are to be returned.

**Execution**

If the inquired information is available; then the error indicator is returned as zero, and values are returned in the output parameters.

If the inquired information is not available; then the values returned in the output parameters are undefined, and the error indicator is set to one of the following error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

051 Ignoring function, this information unavailable for this workstation type

052 Ignoring function, workstation type not recognized by the implementation

**SEE ALSO**

INQUIRE COLOUR MODEL FACILITIES (3P)
PHIGS WORKSTATION DESCRIPTION TABLE (7P)

**NAME**

INQUIRE DISPLAY SPACE SIZE– inquire the display space size of a workstation type

**SYNOPSIS**

**C Syntax**

```
void
pinq_disp_space_size ( type, error_ind, size )
Pint              type;          workstation type
Pint              *error_ind;    OUT error indicator
Pdisp_space_size  *size;         OUT display size
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

INQUIRE DISPLAY SPACE SIZE returns the display space size associated with the workstation type specified. The workstation type passed as an argument to this function can be either a generic type or a specific type, a specific type being one associated with an open workstation and obtained by a call to INQUIRE WORKSTATION CONNECTION AND TYPE. The information is retrieved from the workstation description table associated with the workstation type.

INQUIRE DISPLAY SPACE SIZE returns only the *x* and *y* components of the display space limits. Use INQUIRE DISPLAY SPACE SIZE 3 to retrieve all three dimensions.

**C Input Parameters**

*type* The workstation type, either generic or specific.

**C Output Parameters**

*error_ind*

The error indicator. See the *Execution* section below for a description of its use. See the *Errors* section below for the possible values it may return.

*size* A pointer to a Pdisp_space_size structure in which to store the requested information. Pdisp_space_size is defined in phigs.h as:

```
typedef struct {
        Pdc_units       dc_units;       /* device coordinate units */
        Pfloat_size     size_dc;        /* device volume in coordinate units */
        Pint_size       size_raster;    /* addressable units */
} Pdisp_space_size;
```

*dc_units* indicates the unit type, if any, of the device coordinate units. Pdc_units is defined in phigs.h as:

```
typedef enum {
        PDC_METRES,
        PDC_OTHER
} Pdc_units;
```

*size_dc* contains the upper limits of the device coordinate units. The lower limits are zero in all dimensions. See the *Execution* section below for a description of the meaning of these values. Pfloat_size is defined in phigs.h as:

```
typedef struct {
        Pfloat          size_x;
        Pfloat          size_y;
} Pfloat_size;
```

*size_raster* contains the number of addressable units corresponding to the device coordinate units. See the *Execution* section below for a description of the meaning of these values for each

supported workstation type. Pint_size is defined in phigs.h as:

```
typedef struct {
        Pint            size_x;
        Pint            size_y;
} Pint_size;
```

**Execution**

INQUIRE DISPLAY SPACE SIZE retrieves the device coordinate space limits and the number of addressable units of a workstation type. The device coordinate space limits correspond to the maximum device coordinate (DC) values of a workstation. DC is the conceptual coordinate system of the display device and is the coordinate system used by the application to specify the workstation viewport and input device echo areas. The lower limits of DC are always zero in all three dimensions.

The DC limits and addressable units are specified in the workstation description table associated with a workstation type. They can be different in all three dimensions. The DC limits are set to the size of the workstation's X window when the workstation is opened. When the window is resized the DC limits change to the new size of the window. Subsequent calls to INQUIRE DISPLAY SPACE SIZE will reflect this change by returning the limits in effect at the time of the function call. See the PHIGS_DC_MODEL attribute description in WORKSTATION TYPE SET for a description of how to change this behavior so that DC limits do not change with the window size.

If the type specified is a generic x_tool workstation type (in other words, one not associated with an open workstation), then the display space size corresponds to the size of the window that *would be* used if a workstation were opened with that type.

If the type specified is a generic x_drawable workstation type, then error 51 will be returned in the error indicator, indicating that the information is unavailable for such a workstation type and a specific workstation type is required.

If an error is detected by this function the *error indicator* will indicate the error number of the error detected and no other output data will be returned. If no error is detected, the *error indicator* will be set to zero and the inquired information will be available in the output parameters. Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

052 Ignoring function, workstation type not recognized by the implementation

057 Ignoring function, specified workstation is of category MI

062 Ignoring function, this information is not available for this MO workstation type

**SEE ALSO**

INQUIRE WORKSTATION CONNECTION AND TYPE (3P)
WORKSTATION TYPE SET (3P)
INQUIRE DISPLAY SPACE SIZE 3 (3P)

**NAME**

INQUIRE DISPLAY SPACE SIZE 3– inquire the display space size of a workstation type

**SYNOPSIS**

**C Syntax**

```
void
pinq_disp_space_size3 ( type, error_ind, size )
Pint              type;          workstation type
Pint              *error_ind;    OUT error indicator
Pdisp_space_size3 *size;         OUT display size
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

INQUIRE DISPLAY SPACE SIZE 3 returns the display space size associated with the workstation type specified. The workstation type passed as an argument to this function can be either a generic type or a specific type, a specific type being one associated with an open workstation and obtained by a call to INQUIRE WORKSTATION CONNECTION AND TYPE. The information is retrieved from the workstation description table associated with the workstation type.

**C Input Parameters**

*type* The workstation type, either generic or specific.

**C Output Parameters**

*error_ind*

The error indicator. See the *Execution* section below for a description of its use. See the *Errors* section below for the possible values it may return.

*size* A pointer to a Pdisp_space_size3 structure in which to store the requested information. Pdisp_space_size3 is defined in phigs.h as:

```
typedef struct {
        Pdc_units       dc_units;       /* device coordinate units */
        Pfloat_size3    size_dc;        /* device volume in coordinate units */
        Pint_size3      size_raster;    /* addressable units */
} Pdisp_space_size3;
```

*dc_units* indicates the unit type, if any, of the device coordinate units. Pdc_units is defined in phigs.h as:

```
typedef enum {
        PDC_METRES,
        PDC_OTHER
} Pdc_units;
```

*size_dc* contains the upper limits of the device coordinate units. The lower limits are zero in all dimensions. See the *Execution* section below for a description of the meaning of these values. Pfloat_size3 is defined in phigs.h as:

```
typedef struct {
        Pfloat          size_x;
        Pfloat          size_y;
        Pfloat          size_z;
} Pfloat_size3;
```

*size_raster* contains the number of addressable units corresponding to the device coordinate units. See the *Execution* section below for a description of the meaning of these values for each supported workstation type. Pint_size3 is defined in phigs.h as:

```
typedef struct {
        Pint            size_x;
        Pint            size_y;
        Pint            size_z;
} Pint_size3;
```

**Execution**

INQUIRE DISPLAY SPACE SIZE 3 retrieves the device coordinate space limits and the number of addressable units of a workstation type. The device coordinate space limits correspond to the maximum device coordinate (DC) values of a workstation. DC is the conceptual coordinate system of the display device and is the coordinate system used by the application to specify the workstation viewport and input device echo areas. The lower limits of DC are always zero in all three dimensions.

The DC limits and addressable units are specified in the workstation description table associated with a workstation type. They can be different in all three dimensions. The DC limits are set to the size of the workstation's X window when the workstation is opened. When the window is resized the DC limits change to the new size of the window. Subsequent calls to INQUIRE DISPLAY SPACE SIZE 3 will reflect this change by returning the limits in effect at the time of the function call. See the PHIGS_DC_MODEL attribute description in WORKSTATION TYPE SET for a description of how to change this behavior so that DC limits do not change with the window size.

If the type specified is a generic x_tool workstation type (in other words, one not associated with an open workstation), then the display space size corresponds to the size of the window that *would be* used if a workstation were opened with that type.

> If the type specified is a generic x_drawable workstation type, then error 51 will be returned in the error indicator, indicating that the information is unavailable for such a workstation type and a specific workstation type is required.

If an error is detected by this function the *error indicator* will indicate the error number of the error detected and no other output data will be returned. If no error is detected, the *error indicator* will be set to zero and the inquired information will be available in the output parameters. Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

052 Ignoring function, function requires state (PHOP, *, *, *)

057 Ignoring function, specified workstation is of category MI

062 Ignoring function, this information is not available for this MO workstation type

**SEE ALSO**

INQUIRE WORKSTATION CONNECTION AND TYPE (3P)
WORKSTATION TYPE SET (3P)
INQUIRE DISPLAY SPACE SIZE (3P)

**NAME**

INQUIRE DISPLAY UPDATE STATE – inquire the current display update state

**SYNOPSIS**

**C Syntax**

**void**
**pinq_disp_upd_st ( ws, error_ind, def_mode, mod_mode, display_empty, state )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pdefer_mode** | ***def_mode;** | *OUT deferral mode* |
| **Pmod_mode** | ***mod_mode;** | *OUT modification mode* |
| **Pdisp_surf_empty** | ***disp_empty;** | *OUT display surface empty* |
| **Pvisual_st** | ***state;** | *OUT state of visual representation* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE DISPLAY UPDATE STATE to determine the current display update state from a specified workstation's state list.

The deferral mode and modification mode were set by the application. The display surface empty and state of visual representation state list values are maintained by PHIGS.

**C Input Parameters**

*ws* The *workstation identifier* of the workstation whose state list is queried.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*def_mode*

A pointer to a location in which the system returns the current deferral mode. Possible values for Pdefer_mode are defined in phigs.h as follows:

| | |
|---|---|
| PDEFER_ASAP | Make the display visually correct — *As Soon As Possible* |
| PDEFER_BNIG | Make the display visually correct — *Before the Next Interaction Globally* |
| PDEFER_BNIL | Make the display visually correct — *Before the Next Interaction Locally* |
| PDEFER_ASTI | Make the display visually correct — *At Some Time* |
| PDEFER_WAIT | Make the display visually correct — *When the Application Requests It* |

*mod_mode*

A pointer to a location in which the system returns the current modification mode. Possible values for Pmod_mode are defined in phigs.h as follows:

| | |
|---|---|
| PMODE_NIVE | *No Immediate Visual Effects* |
| PMODE_UWOR | *Update Without Regeneration* |
| PMODE_UQUM | *Use Quick Update Methods* |

*display_empty*

A pointer to a location in which the system returns the current display surface empty status. Possible values for Pdisp_surf_empty are:

| | |
|---|---|
| PSURF_EMPTY | *Empty* |
| PSURF_NOT_EMPTY | *Not Empty* |

*state* A pointer to a location in which the system returns the current state of visual representation. Possible values for Pvisual_st are:

| | |
|---|---|
| PVISUAL_ST_CORRECT | *Correct* |
| PVISUAL_ST_DEFER | *Deferred* |

PVISUAL_ST_SIMULATED *Simulated*

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT, OUTIN, nor MO)

**SEE ALSO**

SET DISPLAY UPDATE STATE (3P)
INQUIRE DEFAULT DISPLAY UPDATE STATE (3P)

**NAME**

INQUIRE DYNAMICS OF STRUCTURES – inquire the dynamics of structures for a workstation type

**SYNOPSIS**

**C Syntax**

```
void
pinq_dyns_structs ( type, error_ind, dynamics )
Pint            type;           workstation type
Pint            *error_ind;     OUT error indicator
Pdyns_structs   *dynamics;      OUT structure dynamics
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE DYNAMICS OF STRUCTURES to determine the dynamics of changes to structure contents and posting for a specified workstation type.

The possible *dynamic modification accepted* values are:

| | |
|---|---|
| PDYN_IRG | Modification leads to an *Implicit Regeneration*. |
| PDYN_IMM | Modification is performed *Immediately*, without an implicit regeneration. |
| PDYN_CBS | Modification *Can Be Simulated* before an implicit regeneration is permitted. |

**C Input Parameters**

*type* The workstation type whose capabilities are inquired.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*dynamics*

A pointer to a structure in which the system returns the *dynamic modification accepted* values for various structure manipulations. Pdyns_structs is defined in phigs.h as follows:

```
typedef struct {
        Pdyn_mod    content;    /* structure content */
        Pdyn_mod    post;       /* post structure */
        Pdyn_mod    unpost;     /* unpost structure */
        Pdyn_mod    del;        /* delete structure */
        Pdyn_mod    ref;        /* structure reference */
} Pdyns_structs;
```

The Pdyn_mod enumeration holds the dynamic modification accepted value, one of the values PDYN_IRG, PDYN_IMM, or PDYN_CBS.

**ERRORS**

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

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

052 Ignoring function, workstation type not recognized by the implementation

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

**SEE ALSO**

PHIGS WORKSTATION DESCRIPTION TABLE (7P)

**NAME**

INQUIRE DYNAMICS OF WORKSTATION ATTRIBUTES – inquire the dynamics of workstation attributes for a workstation type

**SYNOPSIS**

**C Syntax**

```
void
pinq_dyns_ws_attrs ( type, error_ind, attr )
Pint            ws_type;        workstation type
Pint            *err_ind;       OUT error indicator
Pdyns_ws_attrs  *attr;          OUT attributes dynamics
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE DYNAMICS OF WORKSTATION ATTRIBUTES to determine dynamics of workstation attributes for a specified workstation type.

The possible dynamic modification accepted values are:

*PDYN_IRG* Modification leads to an *Implicit Regeneration*.

*PDYN_IMM* Modification is performed *Immediately*, without an implicit regeneration.

*PDYN_CBS* Modification *Can Be Simulated* before an implicit regeneration is permitted.

**C Input Parameters**

*type* The workstation type whose capabilities are inquired.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*attr* A pointer to a structure in which the system returns the dynamic modification accepted values for various workstation attributes. Pdyns_ws_attrs is defined in phigs.h as follows:

```
typedef struct {
        Pdyn_mod    line_bundle;    /* polyline representation */
        Pdyn_mod    marker_bundle;  /* marker representation */
        Pdyn_mod    text_bundle;    /* text representation */
        Pdyn_mod    int_bundle;     /* interior representation */
        Pdyn_mod    edge_bundle;    /* edge representation */
        Pdyn_mod    pat_rep;        /* pattern representation */
        Pdyn_mod    colr_rep;       /* colour representation */
        Pdyn_mod    view_rep;       /* view representation */
        Pdyn_mod    ws_tran;        /* workstation transformation */
        Pdyn_mod    highl_filter;   /* highlight filter */
        Pdyn_mod    invis_filter;   /* invisibility filter */
        Pdyn_mod    hlhsr_mode;     /* HLHSR mode */
} Pdyns_ws_attrs;
```

The Pdyn_mod enumeration holds the dynamic modification accepted value, one of the values PDYN_IRG, PDYN_IMM, or PDYN_CBS.

**ERRORS**

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

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

052 Ignoring function, workstation type not recognized by the implementation

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

**SEE ALSO**

PHIGS WORKSTATION DESCRIPTION TABLE (7P)

**NAME**

INQUIRE DYNAMICS OF WORKSTATION PLUS – inquire the dynamics of PHIGS PLUS workstation attributes for a workstation type

**SYNOPSIS**

**C Syntax**

```
void
pinq_dyns_ws_attrs_plus ( type, error_ind, attr )
Pint                type;           workstation type
Pint                *error_ind;     OUT error indicator
Pdyns_ws_attrs_plus                 *attr;OUT attribute dynamics
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE DYNAMICS OF WORKSTATION PLUS to determine dynamics of PHIGS PLUS workstation attributes for a specified workstation type.

**C Input Parameters**

*type* The workstation type whose capabilities are inquired.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*attr* A pointer to a structure in which the system returns the dynamic modification accepted values for various workstation attributes. Pdyns_ws_attrs_plus is defined in phigs.h as follows:

```
typedef struct {
        Pdyn_mod        light_src_rep;      /* light source representation */
        Pdyn_mod        depth_cue_rep;      /* depth cue representation */
        Pdyn_mod        colr_map_rep;       /* colour mapping representation */
} Pdyns_ws_attrs_plus;
```

The Pdyn_mod is an enumeration defined in phigs.h as follows:

```
typedef enum {
        PDYN_IRG,       /* modification leads to an Implicit Regeneration */
        PDYN_IMM,       /* modification is performed Immediately, without an
                           implicit regeneration */
        PDYN_CBS        /* modification Can Be Simulated before an implicit
                           regeneration is permitted */
} Pdyn_mod;
```

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the following error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

052 Ignoring function, workstation type not recognized by the implementation

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is not OUTPUT or OUTIN)

062 Ignoring function, this information is not available for this MO workstation type

**SEE ALSO**

INQUIRE DYNAMICS OF WORKSTATION ATTRIBUTES (3P)
PHIGS WORKSTATION DESCRIPTION TABLE (7P)

**NAME**

INQUIRE EDGE FACILITIES – inquire list of workstation edge facilities

**SYNOPSIS**

**C Syntax**

```
void
pinq_edge_facs ( type, length, start, error_ind, facilities, total_length )
Pint            type;           workstation type
Pint            length;         length of application list
Pint            start;          starting position
Pint            *error_ind;     OUT error indicator
Pedge_facs      *facilities;    OUT edge facilities
Pint            *total_length;  OUT length of list in PHIGS
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE EDGE FACILITIES to obtain a list of the edge facilities available for the specified type of workstation.

**C Input Parameters**

*type* Workstation type.

*length* The number of elements for which the application has allocated memory in the output parameter list of integers. If a value of zero is used, no data will be returned in this list, but the total number of elements in the workstation list of available edge types will be returned in *total_length*.

*start* Starting position in the list of available edge types in the workstation description table at which to begin the inquiry. The elements of the workstation list of available edge types, beginning with the item number specified by *start*, are copied sequentially into the list of integers until it is full or all the items on the workstation list have been copied.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*facilities*

A pointer to a Pedge_facs structure in which the system returns a list of the edge facilities available for the specified workstation type. Pedgefac is defined in phigs.h as follows:

```
typedef struct {
        Pint_list       types;          /* list of edge types */
        Pint            num_widths;     /* number of available edge widths */
        Pfloat          nom_width;      /* nominal edge width */
        Pfloat          min_width;      /* minimum edge width */
        Pfloat          max_width;      /* maximum edge width */
        Pint            num_pred_inds;  /* number of predefined bundles */
} Pedge_facs;
```

And Pint_list is defined as follows:

```
typedef struct {
        Pint    num_ints;       /* number of Pints in list */
        Pint    *ints;          /* list of integers */
} Pint_list;
```

The application must allocate memory for *length* elements in the list of integers.

*total_length*

A pointer to an integer in which the system returns the total number of items in the list of available edge types. This is the value required for *length* if all the items on the workstation list are to be returned.

**ERRORS**

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

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

052 Ignoring function, workstation type not recognized by the implementation

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

**SEE ALSO**

PHIGS WORKSTATION DESCRIPTION TABLE (7P)
INQUIRE PREDEFINED EDGE REPRESENTATION (3P)
INQUIRE PREDEFINED EDGE REPRESENTATION PLUS (3P+)

**NAME**

INQUIRE EDGE REPRESENTATION – inquire an edge representation on a workstation

**SYNOPSIS**

**C Syntax**

**void**
**pinq_edge_rep ( ws, index, type, error_ind, rep )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **index;** | *edge index* |
| **Pinq_type** | **type;** | *type of returned value* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pedge_bundle** | ***rep;** | *OUT edge representation* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE EDGE REPRESENTATION to determine the current attribute values for a specified entry in a specified workstation's table of defined edge representations.

See the description of the subroutine SET EDGE REPRESENTATION for information on the meaning of these attribute values.

**C Input Parameters**

*ws* Workstation identifier.

*index* Entry to be returned from the workstation's table of edge representations; if this entry is not present in the table and the *type of returned value* parameter is REALIZED, the representation for edge index 1 is returned.

*type* An enumerated value specifying whether the inquired values are to be returned as the values originally specified by the application (SET), or as the values actually being used by the workstation if any of the application-specified values had to be mapped to ones available on the workstation (REALIZED). Valid values are defined in phigs.h as:

| | |
|---|---|
| PINQ_SET | *Return Application-specified Value* |
| PINQ_REALIZED | *Return Value Available on the Workstation* |

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*rep* A pointer to a Pedge_bundle structure in which the system returns the edge representation at *index* in the workstation's table of edge representations. Pedge_bundle is defined in phigs.h as follows:

```
typedef struct {
        Pedge_flag      flag;           /* edge flag */
        Pint            type;           /* edgetype */
        Pfloat          width;          /* edgewidth scale factor */
        Pint            colr_ind;       /* edge colour index */
} Pedge_bundle;
```

Valid values for the Pedge_flag enumerated type are defined in phigs.h as:

PEDGE_OFF
PEDGE_ON

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (in other words, the workstation category is neither OUTPUT, OUTIN, nor MO)

100 Ignoring function, the bundle index value is less than one

101 Ignoring function, the specified representation has not been defined

**SEE ALSO**

INQUIRE EDGE FACILITIES (3P)
SET EDGE REPRESENTATION (3P)
INQUIRE PREDEFINED EDGE REPRESENTATION (3P)
INQUIRE EDGE REPRESENTATION PLUS (3P+)

**NAME**

INQUIRE EDGE REPRESENTATION PLUS – inquire an extended edge representation on a workstation

**SYNOPSIS**

**C Syntax**

**void**
**pinq_edge_rep_plus ( ws, index, type, error_ind, rep )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **index;** | *edge index* |
| **Pinq_type** | **type;** | *type of returned value* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pedge_bundle_plus** | ***rep;** | *OUT edge representation* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

INQUIRE EDGE REPRESENTATION PLUS returns the attribute values of a specified entry in the workstation table of extended edge representations. See SET EDGE REPRESENTATION PLUS for a full description of the contents of the bundle table entry and a description of the extended edge representation table.

**C Input Parameters**

*ws* The workstation identifier.

*index* The edge index.

*type* An enumerated value specifying whether the values to be returned are those originally specified by the application (PINQ_SET), or those resulting after PHIGS mapped them to ones available on the workstation (PINQ_REALIZED). A Pinq_type structure is defined as:

```
typedef enum {
        PINQ_SET,
        PINQ_REALIZED
} Pinq_type;
```

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*rep* A pointer to a Pedge_bundle_plus structure which returns the specified extended edge representation. Pedge_bundle_plus is defined as:

```
typedef struct {
        Pedge_flag      flag;   /* edge flag */
        Pint            type;   /* edgetype */
        Pfloat          width;  /* edgewidth scale factor */
        Pgcolr          colr;   /* edge colour */
} Pedge_bundle_plus;
```

Pedge_flag is defined in phigs.h as:

```
typedef enum {
        PEDGE_OFF,
        PEDGE_ON
} Pedge_flag;
```

Pgcolr is defined as:

```
typedef struct {
        Pint    type;   /* indirect, RGB, CIE, HSV, HLS */
        union {
                Pint    ind;    /* index in workstation colour bundle table */
                struct {
                        Pfloat  x;      /* red, hue, etc. */
                        Pfloat  y;      /* green, saturation, lightness, etc. */
                        Pfloat  z;      /* blue, value, saturation, etc. */
                } general;
        } val;
} Pgcolr;
```

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the specified edge index is not present in the edge bundle table on the workstation and the specified type of returned value is *Realized*, the representation for edge index one is returned.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the following error numbers to indicate the reason for non-availability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT, OUTIN, nor MO)

100 Ignoring function, the bundle index value is less than one

101 Ignoring function, the specified representation has not been defined

**SEE ALSO**

INQUIRE EDGE REPRESENTATION (3P)
SET EDGE REPRESENTATION PLUS (3P+)
INQUIRE PREDEFINED EDGE REPRESENTATION PLUS (3P+)

**NAME**

INQUIRE EDIT MODE – inquire current edit mode for structure

**SYNOPSIS**

**C Syntax**

```
void
pinq_edit_mode ( error_ind, edit_mode )
Pint            *error_ind;          OUT error indicator
Pedit_mode      *edit_mode;          OUT edit mode
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE EDIT MODE to determine the current edit mode for structures. The edit mode may be INSERT or REPLACE.

See the description of the subroutine SET EDIT MODE for information on the meaning of these modes.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*edit_mode*

A pointer to a Pedit_mode enumerated type in which the system returns the current structure edit mode. Values for Pedit_mode are defined in phigs.h as follows:

PEDIT_INSERT
PEDIT_REPLACE

**ERRORS**

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

**SEE ALSO**

SET EDIT MODE (3P)

**NAME**

INQUIRE ELEMENT CONTENT – inquire the contents of the specified element

**SYNOPSIS**

**C Syntax**

**void**
**pinq_elem_content ( struct_id, element, store, error_ind, data )**

| | | |
|---|---|---|
| **Pint** | **struct_id;** | *structure identifier* |
| **Pint** | **element;** | *element number* |
| **Pstore** | **store;** | *handle to Store object* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pelem_data** | ****data;** | *OUT data record* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

INQUIRE ELEMENT CONTENT determines the contents of the specified element in the specified structure.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing data associated with the device state. This buffer is passed as the *store* argument.

The store buffer is a data area managed by PHIGS. While the application is responsible for creating the inital buffer through a call to STORE, PHIGS manages this area such that there is sufficient memory for the specific inquiry. The data record within the store buffer is accessed by the pointer pointed to by *data*.

*struct_id*
Identifier of the structure containing the element for which to return the contents.

*element* Position in the specified structure of the element for which to return the contents.

*store* The memory buffer PHIGS is to use for storing the information returned. This buffer must exist prior to calling this function (see CREATE STORE (3P)).

**C Output Parameters**

*error_ind*
A pointer to the location to store the error number of any error detected by this function.

*data* A pointer to a Pelem_data union, in which the contents of the element will be returned, as appropriate for the element type. Pelem_data is defined in phigs.h as follows:

```
typedef union {
        Pint                            int_data;           /* integer valued data */
        Pfloat                          float_data;         /* float valued data */
        Ppoint_list3                    point_list3;        /* list of 3d points */
        Ppoint_list                     point_list;         /* list of 2d points */
    Ppoint_list_list3   point_list_list3;       /* list of 3d point lists */
        Ppoint_list_list    point_list_list;        /* list of 2d point lists */
        struct {
                        Ppoint3         pos;                /* text pt */
                        Pvec3           dir[2];             /* direction vectors */
                        char            *char_string;       /* text string */
        } text3;
        struct {
                        Ppoint          pos;                /* text pt */
                        char            *char_string;       /* text string */
        } text;
        struct {
```

```
                Ppoint3         ref_point;      /* reference pt */
                Pvec3           offset;         /* anno. pt/offset */
                char            *char_string;   /* text string */
} anno_text_rel3;
struct {
                Ppoint          ref_point;      /* reference pt */
                Pvec            offset;         /* anno. pt/offset */
                char            *char_string;   /* text string */
} anno_text_rel;
struct {
                Pparal          paral;          /* parallelogram */
                Ppat_rep        colr_array;     /* colour array */
} cell_array3;
struct {
                Prect           rect;           /* rectangle */
 Ppat_rep       colr_array;     /* colour array */
} cell_array;
struct {
                Pint            id;             /* GDP3 id */
                Ppoint_list3    point_list;     /* pts */
                Pgdp_data3      data;           /* data record */
} gdp3;
struct {
                Pint            id;             /* GDP id */
                Ppoint_list     point_list;     /* pts */
                Pgdp_data       data;           /* data record */
} gdp;
Ptext_prec                      text_prec;      /* text precision */
Pvec                            char_up_vec;    /* char up vector */
Ptext_path                      text_path;      /* text path */
Ptext_align                     text_align;     /* text alignment */
Pint_style                      int_style;      /* interior style */
Pedge_flag                      edge_flag;      /* edge flag */
Ppoint                          pat_ref_point;  /* pat ref pt */
Pfloat_size             pat_size;       /* pattern size */
struct {
                Ppoint3         ref_point;      /* pattern ref. pt */
                Pvec3           ref_vec[2];     /* vectors */
} pat_ref_point_vecs;
Pint_list                       names;          /* name sets */
struct {
                Paspect         id;             /* attribute id */
                Pasf            source;         /* asf */
} asf;
struct {
                Pcompose_type   compose_type;   /* composition type */
                Pmatrix3        matrix;         /* matrix */
} local_tran3;
struct {
                Pcompose_type   compose_type;   /* composition type */
                Pmatrix         matrix;         /* matrix */
} local_tran;
Pmatrix3                        global_tran3;   /* global transform3 */
```

```
Pmatrix                         global_tran;        /* global transform */
struct {
        Pint                    op;                 /* operator */
        Phalf_space_list3       half_spaces;        /* half-space list */
} model_clip3;
struct {
        Pint                    op;                 /* operator */
        Phalf_space_list        half_spaces;        /* half-space list */
} model_clip;
Pclip_ind                       clip_ind;           /* clipping indicator */
Pdata                           appl_data;          /* application data */
struct {
        Pint                    id;                 /* GSE id */
        Pgse_data               data;               /* GSE data record */
} gse;
struct {
        Pint                    order;
        Pfloat_list             knots;
        Prational               rationality;
        Ppoint_list34           cpts;
        Pfloat                  min;
        Pfloat                  max;
} nurb_curve;                   /* NURBS curve */
struct {
        Pint                    type;
        Pfloat                  value;
} curv_approx;                  /* curve approximation criteria */
struct {
        Pint                    u_order;
        Pint                    v_order;
        Prational               rationality;
        Pfloat_list             uknots;
        Pfloat_list             vknots;
        Ppoint_grid34           grid;
        Pint                    num_trim_loops;
        Ptrimcurve_list         *trim_loops;
} nurb_surf;                    /* NURBS surface */
struct {
        Pint                    type;
        Pfloat                  u_val;
        Pfloat                  v_val;
} surf_approx;                  /* surface approximation criteria */
struct {
        Pint                    vflag;
        Pint                    colr_model;
        Pint                    npl;
        Pline_vdata_list3       *vdata;
} plsd3;
struct {
        Pint                    fflag;
        Pint                    eflag;
        Pint                    vflag;
        Pint                    colr_model;
```

```
                    Pfacet_data3        fdata;
                    Pint                nfa;
                    Pedge_data_list     *edata;
                    Pfacet_vdata_list3  *vdata;
            } fasd3;                                    /* fill area set with data */
            struct {
                    Pint                fflag;
                    Pint                vflag;
                    Pint                colr_model;
                    Pint                nv;
                    Pfacet_data_arr3    fdata;
                    Pfacet_vdata_arr3   vdata;
            } tsd3;                                     /* triangle strip with data */
            struct {
                    Pint                fflag;
                    Pint                vflag;
                    Pint                colr_model;
                    Pint_size           dim;
                    Pfacet_data_arr3    fdata;
                    Pfacet_vdata_arr3   vdata;
            } qmd3;                                     /* quad mesh */
            struct {
                    Pint                fflag;
                    Pint                eflag;
                    Pint                vflag;
                    Pint                colr_model;
                    Pint                num_sets;
                    Pfacet_data_arr3    fdata;
                    Pedge_data_list_list *edata;
                    Pint_list_list      *vlist;
                    Pfacet_vdata_list3  vdata;
            } sofas3;                                   /* set of fill area set */
            struct {
                    Pparal              paral;          /* parallelogram */
                    Ppat_rep_plus   colr_array;     /* colour array */
            } cell_array_plus;
            Pgcolr                      colr;
            Prefl_props         props;              /*surface relectance properties */
            struct {
                    Pint_list           activation;
                    Pint_list           deactivation;
            } lss;
            Pcull_mode          cull_mode;          /* culling mode */
            Pdisting_mode       disting_mode;       /* distinguishing mode */
            struct {
                    Pint                type;
                    Ppara_surf_characs  data;
            } para_surf_charac;                     /* parametric surface characteristics */
    } Pelem_data;
```

**Execution**

INQUIRE CURRENT ELEMENT CONTENT is the second function called in a two function, or step, process. First INQUIRE CURRENT ELEMENT TYPE AND SIZE must be called to determine the type of element to return for the INQUIRE CURRENT ELEMENT CONTENT request. Depending on the element type, the corresponding structure

in the Pelem_data union is returned, as described below.

ADD NAMES TO SET (PEL_ADD_NAMES_TO_SET)

A Pint_list structure provided in Pelem_data as:

```
Pint_list names          /* name sets */
```

where

typedef struct {

```
Pint     num_ints;       /* number of Pints in list */
Pint     *ints;          /* list of integers */
```

} Pint_list;

ALL (PELEM_ALL)

No data involved

ANNOTATION STYLE (PELEM_ANNO_STYLE)

An integer provided in Pelem_data as:

```
Pint          int_data;          /* integer valued data */
```

ANNOTATION TEXT ALIGNMENT (PELEM_ANNO_ALIGN)

A Ptext_align structure provided in Pelem_data as:

```
Ptext_align       text_align;        /* text alignment */
```

where

typedef struct {

```
Phor_text_align   hor;               /* horizontal component */
Pvert_text_align  vert;              /* vertical component */
```

} Ptext_align;

ANNOTATION TEXT CHARACTER HEIGHT (PELEM_ANNO_CHAR_HT)

A Pfloat floating point data provided in Pelem_data as:

```
Pfloat        float_data;        /* float valued data */
```

ANNOTATION TEXT CHARACTER UP VECTOR (PELEM_ANNO_CHAR_UP_VEC)

A Pvec structure provided in Pelem_data as:

```
Pvec          char_up_vec;       /* char up vector */
```

where

typedef struct {

```
Pfloat        delta_x;           /* x magnitude */
Pfloat        delta_y;           /* y magnitude */
```

} Pvec;

ANNOTATION TEXT PATH (PELEM_ANNO_PATH)

A Ptext_path data structure provided in Pelem_data as:

```
Ptext_path        text_path;         /* text path */
```

where

typedef enum {

```
PPATH_RIGHT,
PPATH_LEFT,
PPATH_UP,
PPATH_DOWN
```

} Ptext_path;

ANNOTATION TEXT RELATIVE (PELEM_ANNO_TEXT_REL)

An anno_text_rel structure defined in Pelem_data as:

```
struct {
        Ppoint          ref_point;          /* reference pt */
        Pvec            offset;             /* annotation offset */
        char            *char_string;       /* annotation text string */
} anno_text_rel;
```

ANNOTATION TEXT RELATIVE 3 (PELEM_ANNO_TEXT_REL3)

An anno_text_rel3 structure defined in Pelem_data as:

```
struct {
        Ppoint3         ref_point;          /* reference pt */
        Pvec3           offset;             /* annotation offset */
        char            *char_string;       /* annotation text string */
} anno_text_rel3;
```

APPLICATION DATA (PELEM_APPL_DATA)

A Pdata structure provided in Pelem_data as:

```
        Pdata           appl_data;          /* application data */
where
typedef struct {
        size_t          size;               /* size of data */
        char            *data;              /* pointer to data */
} Pdata;
```

AREA PROPERTIES (PELEM_REFL_PROPS)

A Prefl_props structure provided in Pelem_data as:

```
        Prefl_props     properties;         /* area properties */
where
typedef struct {
        Pfloat          ambient_coef;       /* ambient reflectance coefficient */
        Pfloat          diffuse_coef;       /* diffuse reflectance coefficient */
        Pfloat          specular_coef;      /* specular reflectance coefficient */
        Pgcolr          specular_colr;      /* specular colour */
        Pfloat          specular_exp;       /* specular exponent */
        Pfloat          transpar_coef;      /* transparency coefficient */
} Prefl_props;
```

BACK AREA PROPERTIES (PELEM_BACK_REFL_PROPS)

A Prefl_props structure provided in Pelem_data as:

```
        Prefl_props     properties;         /* area properties */
where
typedef struct {
        Pfloat          ambient_coef;       /* ambient reflectance coefficient */
        Pfloat          diffuse_coef;       /* diffuse reflectance coefficient */
        Pfloat          specular_coef;      /* specular reflectance coefficient */
        Pgcolr          specular_colr;      /* specular colour */
        Pfloat          specular_exp;       /* specular exponent */
        Pfloat          transpar_coef;      /* transparency coefficient */
} Prefl_props;
```

BACK INTERIOR COLOUR (PELEM_BACK_INT_COLR)

A Pgcolr structure provided in Pelem_data as:

```
        Pgcolr          colr;               /* extended colour model */
```

where
typedef struct {
    Pint type; /* indirect, RGB, CIE, HSV, HLS */
    union {
        Pint ind; /* index in workstation colour bundle table */
        struct {
            Pfloat x; /* red, hue, etc */
            Pfloat y; /* green, saturation, lightness, etc */
            Pfloat z; /* blue, value, saturation, etc */
        } general;
    } val;
} Pgcolr;

BACK INTERIOR REFLECTANCE EQUATION (PELEM_BACK_INT_REFL_EQN)
An integer provided in Pelem_data as:

```
Pint    int_data;    /* integer valued data */
```

The predefined reflectance equation values are:

| | | |
|---|---|---|
| 1 | PREFL_NONE | *No Reflectance Calculation Performed* |
| 2 | PREFL_AMBIENT | *Use Ambient Term* |
| 3 | PREFL_AMB_DIFF | *Use Ambient and Diffuse Terms* |
| 4 | PREFL_AMB_DIFF_SPEC | *Use Ambient, Diffuse, and Specular Terms* |

BACK INTERIOR SHADING METHOD (PELEM_BACK_INT_SHAD_METH)
An integer provided in Pelem_data as:

```
Pint    int_data;    /* integer valued data */
```

The predefined shading method values are:

| | | |
|---|---|---|
| 1 | PSD_NONE | *No Shading* |
| 2 | PSD_COLOUR | *Colour Interpolation Shading* |
| 3 | PSD_DOT_PRODUCT | *Dot Product Interpolation Shading* |
| 4 | PSD_NORMAL | *Normal Interpolation Shading* |

BACK INTERIOR STYLE (PELEM_BACK_INT_STYLE)
An Pint_style structure provided in Pelem_data as:

```
Pint_style    int_style;    /* back interior style */
```

where
typedef enum {
    PSTYLE_HOLLOW,
    PSTYLE_SOLID,
    PSTYLE_PAT,
    PSTYLE_HATCH,
    PSTYLE_EMPTY
} Pint_style;

BACK INTERIOR STYLE INDEX (PELEM_BACK_INT_STYLE_IND)
An integer provided in Pelem_data as:

```
Pint    int_data;    /* integer valued data */
```

CELL ARRAY (PELEM_CELL_ARRAY)
A cell_array structure defined in Pelem_data as:

```
struct {
    Prect       rect;     /* rectangle */
    Pint_size   dim;      /* dimension */
    Pint        *colr;    /* colour array */
```

```
} cell_array;
```

CELL ARRAY 3 (PELEM_CELL_ARRAY3)

A cell_array3 structure defined in Pelem_data as:

```
struct {
        Pparal          paral;          /* parallelogram */
        Pint_size       dim;            /* dimension */
        Pint            *colr;          /* colour array */
} cell_array3;
```

CHARACTER EXPANSION FACTOR (PELEM_CHAR_EXPAN)

A floating point provided in Pelem_data as:

```
        Pfloat          float_data;     /* float valued data */
```

CHARACTER HEIGHT (PELEM_CHAR_HT)

A floating point provided in Pelem_data as:

```
        Pfloat          float_data;     /* float valued data */
```

CHARACTER SPACING (PELEM_CHAR_SPACE)

A float_data floating point data provided in Pelem_data as:

```
        Pfloat          float_data;     /* float valued data */
```

CHARACTER UP VECTOR (PELEM_CHAR_UP_VEC)

A Pvec structure provided in Pelem_data as:

```
        Pvec            char_up_vec;    /* char up vector */
```

where

```
typedef struct {
        Pfloat          delta_x;        /* x magnitude */
        Pfloat          delta_y;        /* y magnitude */
} Pvec;
```

COLOUR MAPPING INDEX (PEX_COLR_MAPPING_IND)

An integer provided in Pelem_data as:

```
        Pint            int_data;       /* integer valued data */
```

CURVE APPROXIMATION CRITERIA (PELEM_CURVE_APPROX_CRIT)

A curv_approx structure defined in Pelem_data as:

```
struct {
        Pint            type;           /* approximation type */
        Pfloat          value;          /* approximation value */
} curv_approx;
```

DEPTH CUE INDEX (PELEM_DCUE_IND)

An integer provided in Pelem_data as:

```
        Pint            int_data;       /* integer valued data */
```

EDGE COLOUR (PELEM_EDGE_COLR)

A Pgcolr structure provided in Pelem_data as:

```
        Pgcolr          colr;           /* extended colour model */
```

where

```
typedef struct {
        Pint    type;   /* indirect, RGB, CIE, HSV, HLS */
        union {
                Pint    ind;    /* index in workstation colour bundle table */
```

```
                struct {
                        Pfloat    x;        /* red, hue, etc */
                        Pfloat    y;        /* green, saturation, lightness, etc */
                        Pfloat    z;        /* blue, value, saturation, etc */
                } general;
        } val;
} Pgcolr;
```

EDGE COLOUR INDEX (PELEM_EDGE_COLR_IND)

An integer provided in Pelem_data as:

```
Pint            int_data;           /* integer valued data */
```

EDGE FLAG (PELEM_EDGE_FLAG)

A Pedge_flag data structure provided in Pelem_data as:

```
Pedge_flag      edge_flag;          /* edge flag */
```

where

```
typedef enum {
        PEDGE_OFF,
        PEDGE_ON
} Pedge_flag;
```

EDGE INDEX (PELEM_EDGE_IND)

An integer provided in Pelem_data as:

```
Pint            int_data;           /* integer valued data */
```

EDGETYPE (PELEM_EDGETYPE)

An integer provided in Pelem_data as:

```
Pint            int_data;           /* integer valued data */
```

The predefined line type values are:

| | |
|---|---|
| 1 | PLINE_SOLID |
| 2 | PLINE_DASH |
| 3 | PLINE_DOT |
| 4 | PLINE_DOT_DASH |

EDGEWIDTH SCALE FACTOR (PELEM_EDGEWIDTH)

A floating point provided in Pelem_data as:

```
Pfloat          float_data;         /* float valued data */
```

EXECUTE STRUCTURE (PELEM_EXEC_STRUCT)

An integer provided in Pelem_data as:

```
Pint            int_data;           /* integer valued data */
```

CELL ARRAY 3 PLUS (PELEM_CELL_ARRAY3_PLUS)

An cell_array_plus structure provided in Pelem_data as:

```
struct {
        Pparal          paral;          /* parallelogram */
        Pint_size       dim;            /* dimension */
        Pint            colr_model;     /* colour model */
        Pcoval          *colr;          /* colour array */
} cell_array_plus;
```

FACE CULLING MODE (PELEM_FACE_CULL_MODE)

An Pcull_mode data structure provided in Pelem_data as:

```
Pcull_mode      cull_mode;          /* culling mode */
```

where
typedef enum {
PCULL_NONE,
PCULL_BACKFACE,
PCULL_FRONTFACE
} Pcull_mode;

FACE DISTINGUISHING MODE (PELEM_FACE_DISTING_MODE)
An Pdisting_mode data structure provided in Pelem_data as:

```
Pdisting_mode    disting_mode;        /* distinguishing mode */
```

where
typedef enum {
PDISTING_NO,
PDISTING_YES
} Pdisting_mode;

FILL AREA (PELEM_FILL_AREA)
A Ppoint_list structure provided in Pelem_data as:

```
Ppoint_list      point_list;          /* list of 2d points */
```

where
typedef struct {

```
Pint             num_points;          /* number of Ppoints in the list */
Ppoint           *points;             /* list of points */
```

} Ppoint_list;

FILL AREA SET (PELEM_FILL_AREA_SET)
A fill_area_set structure defined in Pelem_data as:

struct {

```
Pint             num_point_lists;     /* number of sets */
Ppoint_list      *point_lists;        /* list of sets */
```

} fill_area_set;

FILL AREA SET 3 (PELEM_FILL_AREA_SET3)
A fill_area_set3 structure defined in Pelem_data as:

struct {

```
Pint             num_point_lists;     /* number of sets */
Ppoint_list3     *point_lists;        /* list of sets */
```

} fill_area_set3;

FILL AREA SET 3 WITH DATA (PELEM_FILL_AREA_SET3_DATA)
A fasd3 structure defined in Pelem_data as:

struct {

```
Pint                 fflag;           /* data specified per facet */
Pint                 eflag;           /* edge visibility status */
Pint                 vflag;           /* data per vertex flag */
Pint                 colr_model;      /* colour type */
Pfacet_data3         fdata;           /* facet data */
Pint                 nfa;             /* number of fill areas in the set */
Pedge_data_list      *edata;          /* edge data */
Pfacet_vdata_arr3    *vdata;          /* vertex data */
```

} fasd3;

FILL AREA 3 (PELEM_FILL_AREA3)

A Ppoint_list3 structure provided in Pelem_data as:

```
        Ppoint_list3    point_list3;        /* list of 3d points */
```

where

typedef struct {

```
        Pint            num_points;         /* number of Ppoint3s in the list*/
        Ppoint3         *points;            /* list of points */
```

} Ppoint_list3;

GDP (PELEM_GDP)

A gdp structure defined in Pelem_data as:

struct {

```
        Pint            id;                 /* GDP id */
        Ppoint_list     point_list;         /* pts */
        Pgdp_data       data;               /* data record */
```

} gdp;

GDP 3 (PELEM_GDP3)

A gdp3 structure defined in Pelem_data as:

struct {

```
        Pint            id;                 /* GDP3 id */
        Ppoint_list3    point_list;         /* pts */
        Pgdp_data3      data;               /* data record */
```

} gdp3;

GLOBAL MODELLING TRANSFORMATION (PELEM_GLOBAL_MODEL_TRAN)

A Pmatrix structure provided in Pelem_data as:

```
        Pmatrix         global_tran;        /* global transform */
```

where

typedef Pfloat Pmatrix[3][3];

GLOBAL MODELLING TRANSFORMATION 3 (PELEM_GLOBAL_MODEL_TRAN3)

A Pmatrix3 structure provided in Pelem_data as:

```
        Pmatrix3        global_tran3;       /* global transform3 */
```

where

typedef Pfloat Pmatrix3[4][4];

GSE (PELEM_GSE)

A gse structure defined in Pelem_data as:

struct {

```
        Pint            id;                 /* GSE id */
        Pgse_data       data;               /* GSE data record */
```

} gse;

HLHSR IDENTIFIER (PELEM_HLHSR_ID)

An integer provided in Pelem_data as:

```
        Pint            int_data;           /* integer valued data */
```

INDIVIDUAL ASF (PELEM_INDIV_ASF)

An asf structure defined in Pelem_data as:

struct {

```
        Paspect         id;                 /* attribute id */
        Pasf            source;             /* asf */
```

```
} asf;
```

where

```
typedef enum {
    PASPECT_LINETYPE,
    PASPECT_LINEWIDTH,
    PASPECT_LINE_COLR_IND,
    PASPECT_MARKER_TYPE,
    PASPECT_MARKER_SIZE,
    PASPECT_MARKER_COLR_IND,
    PASPECT_TEXT_FONT,
    PASPECT_TEXT_PREC,
    PASPECT_CHAR_EXPAN,
    PASPECT_CHAR_SPACE,
    PASPECT_TEXT_COLR_IND,
    PASPECT_INT_STYLE,
    PASPECT_INT_STYLE_IND,
    PASPECT_INT_COLR_IND,
    PASPECT_EDGE_FLAG,
    PASPECT_EDGETYPE,
    PASPECT_EDGEWIDTH,
    PASPECT_EDGE_COLR_IND,
    PASPECT_CURVE_APPROX_CRIT,
    PASPECT_SURF_APPROX_CRIT,
    PASPECT_LINE_SHAD_METH,
    PASPECT_REFL_PROPS,
    PASPECT_INT_REFL_EQN,
    PASPECT_INT_SHAD_METH,
    PASPECT_BACK_INT_STYLE,
    PASPECT_BACK_INT_STYLE_IND,
    PASPECT_BACK_INT_COLR,
    PASPECT_BACK_REFL_PROPS,
    PASPECT_BACK_INT_REFL_EQN,
    PASPECT_BACK_INT_SHAD_METH
} Paspect;
```

and

```
typedef enum {
    PASF_BUNDLED,
    PASF_INDIV
} Pasf;
```

INTERIOR COLOUR (PELEM_INT_COLR)

A Pgcolr structure provided in Pelem_data as:

```
        Pgcolr          colur;              /* extended colour model */
```

where

```
typedef struct {
        Pint    type;  /* indirect, RGB, CIE, HSV, HLS */
        union {
                Pint    ind;    /* index in workstation colour bundle table */
                struct {
                        Pfloat  x;      /* red, hue, etc */
                        Pfloat  y;      /* green, saturation, lightness, etc */
                        Pfloat  z;      /* blue, value, saturation, etc */
```

```
            } general;
        } val;
    } Pgcolr;
```

INTERIOR COLOUR INDEX (PELEM_INT_COLR_IND)

An integer provided in Pelem_data as:

```
    Pint        int_data;       /* integer valued data */
```

INTERIOR INDEX (PELEM_INT_IND)

An integer provided in Pelem_data as:

```
    Pint        int_data;       /* integer valued data */
```

INTERIOR REFLECTANCE EQUATION (PELEM_INT_REFL_EQN)

An integer provided in Pelem_data as:

```
    Pint        int_data;       /* integer valued data */
```

The predefined reflectance equation values are:

| | | |
|---|---|---|
| 1 | PREFL_NONE | *No Reflectance Calculation Performed* |
| 2 | PREFL_AMBIENT | *Use Ambient Term* |
| 3 | PREFL_AMB_DIFF | *Use Ambient and Diffuse Terms* |
| 4 | PREFL_AMB_DIFF_SPEC | *Use Ambient, Diffuse, and Specular Terms* |

INTERIOR SHADING METHOD (PELEM_INT_SHAD_METH)

An integer provided in Pelem_data as:

```
    Pint        int_data;       /* integer valued data */
```

The predefined shading method values are:

| | | |
|---|---|---|
| 1 | PSD_NONE | *No Shading* |
| 2 | PSD_COLOUR | *Colour Interpolation Shading* |
| 3 | PSD_DOT_PRODUCT | *Dot Product Interpolation Shading* |
| 4 | PSD_NORMAL | *Normal Interpolation Shading* |

INTERIOR STYLE (PELEM_INT_STYLE)

An Pint_style data structure provided in Pelem_data as:

```
    Pint_style  int_style;      /* interior style */
```

where

```
typedef enum {
    PSTYLE_HOLLOW,
    PSTYLE_SOLID,
    PSTYLE_PAT,
    PSTYLE_HATCH,
    PSTYLE_EMPTY,
} Pint_style;
```

INTERIOR STYLE INDEX (PELEM_INT_STYLE_IND)

An integer provided in Pelem_data as:

```
    Pint        int_data;       /* integer valued data */
```

LABEL (PELEM_LABEL)

An integer provided in Pelem_data as:

```
    Pint        int_data;       /* integer valued data */
```

LIGHT SOURCE STATE (PELEM_LIGHT_SRC_STATE)

An lss structure defined in Pelem_data as:

```
struct {
    Pint_list   activation;     /* activation list */
```

```
        Pint_list       deactivation;       /* deactivation list */
    } lss;
```

LINETYPE (PELEM_LINETYPE)

An integer provided in Pelem_data as:

```
        Pint            int_data;           /* integer valued data */
```

The predefined line type values are:

1 PLINE_SOLID
2 PLINE_DASH
3 PLINE_DOT
4 PLINE_DOT_DASH

LINEWIDTH SCALE FACTOR (PELEM_LINEWIDTH)

A floating point provided in Pelem_data as:

```
        Pfloat          float_data;         /* float valued data */
```

LOCAL MODELLING TRANSFORMATION (PELEM_LOCAL_MODEL_TRAN)

A local_tran structure defined in Pelem_data as:

```
struct {
        Pcompose_type   comp;               /* composition type */
        Pmatrix         tran;               /* matrix */
} local_tran;
```

where

```
typedef enum {
        PTYPE_PRECONCAT,
        PTYPE_POSTCONCAT,
        PTYPE_REPLACE
} Pcompose_type;
```

LOCAL MODELLING TRANSFORMATION 3 (PELEM_LOCAL_MODEL_TRAN3)

A local_tran3 structure defined in Pelem_data as:

```
struct {
        Pcompose_type   comp;               /* composition type */
        Pmatrix3        tran;               /* matrix */
} local_tran3;
```

where

```
typedef enum {
   PTYPE_PRECONCAT,
   PTYPE_POSTCONCAT,
   PTYPE_REPLACE
} Pcompose_type;
```

MARKER COLOUR INDEX (PELEM_MARKER_COLR_IND)

An integer provided in Pelem_data as:

```
        Pint            int_data;           /* integer valued data */
```

MARKER SIZE SCALE FACTOR (PELEM_MARKER_SIZE)

A floating point provided in Pelem_data as:

```
        Pfloat          float_data;         /* float valued data */
```

MARKER TYPE (PELEM_MARKER_TYPE)

An integer provided in Pelem_data as:

```
        Pint            int_data;           /* integer valued data */
```

The predefined marker type values are:

1 PMARKER_DOT
2 PMARKER_PLUS
3 PMARKER_ASTERISK
4 PMARKER_CIRCLE
5 PMARKER_CROSS

MODELLING CLIPPING INDICATOR (PELEM_MODEL_CLIP_IND)

A Pclip_ind structure provided in Pelem_data as:

```
        Pclip_ind       clip_ind;           /* clipping indicator */
```

where

```
typedef enum {
        PIND_NO_CLIP,
        PIND_CLIP
} Pclip_ind;
```

MODELLING CLIPPING VOLUME (PELEM_MODEL_CLIP_VOL)

A model_clip structure defined in Pelem_data as:

```
struct {
        Pint                op;             /* operator */
        Phalf_space_list    half_spaces;    /* half-space list */
} model_clip;
```

MODELLING CLIPPING VOLUME 3 (PELEM_MODEL_CLIP_VOL3)

A model_clip3 structure defined in Pelem_data as:

```
struct {
        Pint                op;             /* operator */
        Phalf_space_list3   half_spaces;    /* half-space list */
} model_clip3;
```

NIL (PELEM_NIL)

No data involved

NON-UNIFORM B-SPLINE CURVE (PELEM_NUNI_BSP_CURVE)

A nurb_curve structure defined in Pelem_data as:

```
struct {
        Pint            order;              /* curve order */
        Pfloat_list     knots;              /* knots */
        Prational       rationality;        /* rationality specifier */
        Ppoint_list34   cpts;               /* control points */
        Pfloat          min;                /* curve min */
        Pfloat          max;                /* curve max */
} nurb_curve;
```

NON-UNIFORM B-SPLINE SURFACE (PELEM_NUNI_BSP_SURF)

A nurb_surf structure defined in Pelem_data as:

```
struct {
        Pint            u_order;            /* order of u surface */
        Pint            v_order;            /* order of v surface */
        Prational       rationality;        /* rationality specifier */
        Pfloat_list     uknots;             /* knots in u dimension */
        Pfloat_list     vknots;             /* knots in v dimension */
```

```
        Ppoint_grid34    grid;          /* grid of control points */
        Pint             num_trim_loops;/* number of trim curve loops */
        Ptrimcurve_list  *trim_loops;   /* trim curve loops */
} nurb_surf;
```

NUM_EL_TYPES (PELEM_NUM_EL_TYPES)

No data involved

PARAMETRIC SURFACE CHARACTERISTICS (PEL_PARA_SURF_CHARACTERISITICS)

A para_surf_characs structure provided in Pelem_data as:

```
struct {
        Pint                type;
        Ppara_surf_characs  data;
} para_surf_characs;        /* parametric surface characteristics */
```

where

```
typedef union {
        struct {
                Pint            unused;
        } psc_1;
        struct {
                Pint            unused;
        } psc_2;
        struct {
                Pcurve_placement    placement;
                Pint                u_count;
                Pint                v_count;
        } psc_3;
        struct {
                Ppoint3         origin;
                Pvec3           direction;
                Pfloat_list     params;
        } psc_4;
        struct {
                Ppoint3         origin;
                Pvec3           direction;
                Pfloat_list     params;
        } psc_5;
} Ppara_surf_characs;
```

PATTERN REFERENCE POINT (PELEM_PAT_REF_POINT)

A Ppoint structure provided in Pelem_data as:

```
        Ppoint      point;          /* pattern reference pt */
```

where

```
typedef struct {
        Pfloat      x;              /* x coordinate */
        Pfloat      y;              /* y coordinate */
} Ppoint;
```

PATTERN REFERENCE POINT AND VECTORS (PELEM_PAT_REF_POINT_VECS)

A pat_ref_point_vecs structure defined in Pelem_data as:

```
struct {
        Ppoint3     ref_point;      /* pattern ref. pt */
        Pvec3       ref_vec[2];     /* vectors */
```

```
} pat_ref_point_vecs;
```

PATTERN SIZE (PELEM_PAT_SIZE)

A Ppoint structure provided in Pelem_data as:

```
        Ppoint          point;          /* pattern size */
```

where

```
typedef struct {
        Pfloat          x;              /* x coordinate */
        Pfloat          y;              /* y coordinate */
} Ppoint;
```

PICK ID (PELEM_PICK_ID)

An integer provided in Pelem_data as:

```
        Pint            int_data;       /* integer valued data */
```

POLYLINE (PELEM_POLYLINE)

A Ppoint_list structure provided in Pelem_data as:

```
        Ppoint_list     point_list;     /* list of 2d points */
```

where

```
typedef struct {
        Pint            num_points;     /* number of Ppoints */
        Ppoint          *points;        /* array of points */
} Ppoint_list;
```

POLYLINE COLOUR (PELEM_LINE_COLR)

A Pgcolr structure provided in Pelem_data as:

```
        Pgcolr          colr;           /* extended colour model */
```

where

```
typedef struct {
        Pint    type;   /* indirect, RGB, CIE, HSV, HLS */
        union {
                Pint    ind;    /* index in workstation colour bundle table */
                struct {
                        Pfloat  x;      /* red, hue, etc */
                        Pfloat  y;      /* green, saturation, lightness, etc */
                        Pfloat  z;      /* blue, value, saturation, etc */
                } general;
        } val;
} Pgcolr;
```

POLYLINE COLOUR INDEX (PELEM_LINE_COLR_IND)

An integer provided in Pelem_data as:

```
        Pint            int_data;       /* integer valued data */
```

POLYLINE INDEX (PELEM_LINE_IND)

An integer provided in Pelem_data as:

```
        Pint            int_data;       /* integer valued data */
```

POLYLINE SET 3 WITH DATA (PELEM_POLYLINE_SET3_DATA)

A plsd3 structure defined in Pelem_data as:

```
struct {
        Pint                    vflag;          /* data per vertex flag */
        Pint                    colr_model;     /* colour type */
```

```
        Pint                    npl;            /* number of polylines in set */
        Pline_vdata_list3       *vdata;         /* per line vertex data list */
} plsd3;
```

POLYLINE SHADING METHOD (PELEM_LINE_SHAD_METH)

An integer provided in Pelem_data as:

```
        Pint            int_data;       /* integer valued data */
```

The predefined shading method values are:

| | | |
|---|---|---|
| 1 | PSD_NONE | *No Shading* |
| 2 | PSD_COLOUR | *Colour Interpolation Shading* |

POLYLINE 3 (PELEM_POLYLINE3)

A Ppoint_list3 structure provided in Pelem_data as:

```
        Ppoint_list3    pts3;           /* list of 3d points */
```

where

```
typedef struct {
        Pint            num_points;     /* number of Ppoint3s */
        Ppoint3         *points;        /* array of points */
} Ppoint_list3;
```

POLYMARKER (PELEM_POLYMARKER)

A Ppoint_list structure provided in Pelem_data as:

```
        Ppoint_list     pts;            /* list of 2d points */
```

where

```
typedef struct {
        Pint            num_points;     /* number of Ppoints */
        Ppoint          *points;        /* array of points */
} Ppoint_list;
```

POLYMARKER COLOUR (PELEM_MARKER_COLR)

A Pgcolr structure provided in Pelem_data as:

```
        Pgcolr          colr;           /* extended colour model */
```

where

```
typedef struct {
        Pint    type;   /* indirect, RGB, CIE, HSV, HLS */
        union {
                Pint    ind;    /* index in workstation colour bundle table */
                struct {
                        Pfloat  x;      /* red, hue, etc */
                        Pfloat  y;      /* green, saturation, lightness, etc */
                        Pfloat  z;      /* blue, value, saturation, etc */
                } general;
        } val;
} Pgcolr;
```

POLYMARKER INDEX (PELEM_MARKER_IND)

An integer provided in Pelem_data as:

```
        Pint            int_data;       /* integer valued data */
```

POLYMARKER 3 (PELEM_POLYMARKER3)

A Ppoint_list3 structure provided in Pelem_data as:

```
        Ppoint_list3    point_list3;    /* list of 3d points */
```

where

```
typedef struct {
        Pint            num_points;     /* number of Ppoint3s */
        Ppoint3         *points;        /* array of points */
} Ppoint_list3;
```

QUADRILATERAL MESH 3 WITH DATA (PELEM_QUAD_MESH3_DATA)

A qmd3 structure defined in Pelem_data as:

```
struct {
        Pint                    fflag;          /* data per facet flag */
        Pint                    vflag;          /* data per vertex flag */
        Pint                    colr_model;     /* colour type */
        Pint_size               dim;            /* dimension of cells */
        Pfacet_data_arr3        fdata;          /* facet data */
        Pfacet_vdata_arr3       vdata;          /* vertex data */
} qmd3;
```

REMOVE NAMES FROM SET (PELEM_REMOVE_NAMES_SET)

A Pint_list structure provided in Pelem_data as:

```
        Pint_list       names           /* name sets */
```

where

```
typedef struct {
        Pint            num_ints;       /* number of names in the set */
        Pint            *ints;          /* name set */
} Pint_list
```

RENDERING COLOUR MODEL (PELEM_RENDERING_COLR_MODEL)

An integer provided in Pelem_data as:

```
        Pint            int_data;       /* integer valued data */
```

RESTORE MODELLING CLIPPING VOLUME (PELEM_RESTORE_MODEL_CLIP_VOL)

No data involved

SET OF FILL AREA SET 3 WITH DATA (PELEM_SET_OF_FILL_AREA_SET3_DATA)

A sofas3 structure defined in Pelem_data as:

```
struct {
        Pint                    fflag;
        Pint                    eflag;
        Pint                    vflag;
        Pint                    colr_model;
        Pint                    num_sets;
        Pfacet_data_arr3        fdata;
        Pedge_data_list_list    *edata;
        Pint_list_list          *vlist;
        Pfacet_vdata_list3      vdata;
} sofas3;
```

SURFACE APPROXIMATION CRITERIA (PELEM_SURF_APPROX_CRIT)

A surf_approx structure defined in Pelem_data as:

```
struct {
        Pint            type;           /* approximation type */
        Pfloat          u_val;          /* u approximation value */
        Pfloat          v_val;          /* v approximation value */
} surf_approx;
```

TEXT (PELEM_TEXT)

A text structure defined in Pelem_data as:

```
struct {
        Ppoint          pos;            /* text pt */
        char            *char_string;   /* text string */
} text;
```

TEXT ALIGNMENT (PELEM_TEXT_ALIGN)

A Ptext_align structure provided in Pelem_data as:

```
        Ptext_align     text_align;     /* text alignment */
where
typedef struct {
        Phor_text_align  hor;           /* horizontal component */
        Pvert_text_align vert;          /* vertical component */
} Ptext_align;
```

TEXT COLOUR (PELEM_TEXT_COLR)

A Pgcolr structure provided in Pelem_data as:

```
        Pgcolr          colr;           /* extended colour model */
where
typedef struct {
        Pint    type;   /* indirect, RGB, CIE, HSV, HLS */
        union {
                Pint    ind;    /* index in workstation colour bundle table */
                struct {
                        Pfloat  x;      /* red, hue, etc */
                        Pfloat  y;      /* green, saturation, lightness, etc */
                        Pfloat  z;      /* blue, value, saturation, etc */
                } general;
        } val;
} Pgcolr;
```

TEXT COLOUR INDEX (PELEM_TEXT_COLR_IND)

An integer provided in Pelem_data as:

```
        Pint            int_data;       /* integer valued data */
```

TEXT FONT (PELEM_TEXT_FONT)

An integer provided in Pelem_data as:

```
        Pint            int_data;       /* integer valued data */
```

The predefined text font values are:

| | |
|---|---|
| 1 | PFONT_MONO |

TEXT INDEX (PELEM_TEXT_IND)

An integer provided in Pelem_data as:

```
        Pint            int_data;       /* integer valued data */
```

TEXT PATH (PELEM_TEXT_PATH)

A Ptext_path data structure provided in Pelem_data as:

```
        Ptext_path      text_path;      /* text path */
where
typedef enum {
        PPATH_RIGHT,
```

```
        PPATH_LEFT,
        PPATH_UP,
        PPATH_DOWN
    } Ptext_path;
```

TEXT PRECISION (PELEM_TEXT_PREC)

A Ptext_prec data structure provided in Pelem_data as:

```
    Ptext_prec      text_prec;          /* text precision */
```

where

```
typedef enum {
        PPREC_STRING,
        PPREC_CHAR,
        PPREC_STROKE
    } Ptext_prec;
```

TEXT 3 (PELEM_TEXT3)

A text3 structure defined in Pelem_data as:

```
    struct {
        Ppoint3         pos;            /* text pt */
        Pvec3           dir[2];         /* direction vectors */
        char            *char_string;   /* text string */
    } text3;
```

TRIANGLE STRIP 3 WITH DATA (PELEM_TRI_STRIP3_DATA)

A tsd3 structure defined in Pelem_data as:

```
    struct {
        Pint                fflag;      /* data per facet flag */
        Pint                vflag;      /* data per vertex flag */
        Pint                colr_model; /* colour type */
        Pint                nv;         /* number of vertices */
        Pfacet_data_arr3    fdata;      /* facet data */
        Pfacet_vdata_arr3   vdata;      /* vertex data */
    } tsd3;
```

VIEW INDEX (PELEM_VIEW_IND)

An integer provided in Pelem_data as:

```
    Pint        int_data;           /* integer valued data */
```

**ERRORS**

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

201 Ignoring function, the specified structure does not exist

202 Ignoring function, the specified element does not exist

**SEE ALSO**

UNPACK DATA RECORD (3P)
INQUIRE ELEMENT TYPE AND SIZE (3P)
INQUIRE CURRENT ELEMENT CONTENT (3P)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

INQUIRE ELEMENT POINTER – inquire the current element pointer value

**SYNOPSIS**

**C Syntax**

```
void
pinq_elem_ptr ( error_ind, ep_value )
Pint            *error_ind;         OUT error indicator
Pint            *ep_value;          OUT element pointer value
```

**Required PHIGS Operating States**

(PHOP, ∗, STOP, ∗)

**DESCRIPTION**

**Purpose**

Use INQUIRE ELEMENT POINTER to determine the current element pointer value.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*ep_value*

Returns current element pointer value.

**ERRORS**

005 Ignoring function, function requires state (PHOP, ∗, STOP, ∗)

**SEE ALSO**

INQUIRE CURRENT ELEMENT TYPE AND SIZE (3P)
SET ELEMENT POINTER (3P)
OFFSET ELEMENT POINTER (3P)
SET ELEMENT POINTER AT LABEL (3P)

**NAME**

INQUIRE ELEMENT TYPE AND SIZE – inquire the type and size of a specified element

**SYNOPSIS**

**C Syntax**

**void**
**pinq_elem_type_size ( struct_id, element, error_ind, type, size )**

| | | |
|---|---|---|
| **Pint** | **struct_id;** | *structure identifier* |
| **Pint** | **element;** | *element number* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pelem_type** | ***type;** | *OUT element type* |
| **size_t** | ***size;** | *OUT element size* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

INQUIRE ELEMENT TYPE AND SIZE returns the type and size of the specified element in the specified structure.

**C Input Parameters**

*struct_id*

Identifier of the structure containing the element for which to return a type and size.

*element* Sequence number in the specified structure of the element for which to return a type and size.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*type* Returns element type. For example, a polyline primitive element would be returned as PELEM_POLYLINE, a character height attribute as PEL_CHARACTER_HEIGHT, a modelling transformation as PEL_LOCAL_MODELLING_TRANSFORMATION3, and so forth. The Pelem_type enumerated type is defined in phigs.h as follows:

```
typedef enum {
        PELEM_ALL,
        PELEM_NIL,
        PELEM_POLYLINE3,
        PELEM_POLYLINE,
        PELEM_POLYMARKER3,
        PELEM_POLYMARKER,
        PELEM_TEXT3,
        PELEM_TEXT,
        PELEM_ANNO_TEXT_REL3,
        PELEM_ANNO_TEXT_REL,
        PELEM_FILL_AREA3,
        PELEM_FILL_AREA,
        PELEM_FILL_AREA_SET3,
        PELEM_FILL_AREA_SET,
        PELEM_CELL_ARRAY3,
        PELEM_CELL_ARRAY,
        PELEM_GDP3,
        PELEM_GDP,
        PELEM_LINE_IND,
        PELEM_MARKER_IND,
        PELEM_TEXT_IND,
```

PELEM_INT_IND,
PELEM_EDGE_IND,
PELEM_LINETYPE,
PELEM_LINEWIDTH,
PELEM_LINE_COLR_IND,
PELEM_MARKER_TYPE,
PELEM_MARKER_SIZE,
PELEM_MARKER_COLR_IND,
PELEM_TEXT_FONT,
PELEM_TEXT_PREC,
PELEM_CHAR_EXPAN,
PELEM_CHAR_SPACE,
PELEM_TEXT_COLR_IND,
PELEM_CHAR_HT,
PELEM_CHAR_UP_VEC,
PELEM_TEXT_PATH,
PELEM_TEXT_ALIGN,
PELEM_ANNO_CHAR_HT,
PELEM_ANNO_CHAR_UP_VEC,
PELEM_ANNO_PATH,
PELEM_ANNO_ALIGN,
PELEM_ANNO_STYLE,
PELEM_INT_STYLE,
PELEM_INT_STYLE_IND,
PELEM_INT_COLR_IND,
PELEM_EDGE_FLAG,
PELEM_EDGETYPE,
PELEM_EDGEWIDTH,
PELEM_EDGE_COLR_IND,
PELEM_PAT_SIZE,
PELEM_PAT_REF_POINT_VECS,
PELEM_PAT_REF_POINT,
PELEM_ADD_NAMES_SET,
PELEM_REMOVE_NAMES_SET,
PELEM_INDIV_ASF,
PELEM_HLHSR_ID,
PELEM_LOCAL_MODEL_TRAN3,
PELEM_LOCAL_MODEL_TRAN,
PELEM_GLOBAL_MODEL_TRAN3,
PELEM_GLOBAL_MODEL_TRAN,
PELEM_MODEL_CLIP_VOL3,
PELEM_MODEL_CLIP_VOL,
PELEM_MODEL_CLIP_IND,
PELEM_RESTORE_MODEL_CLIP_VOL,
PELEM_VIEW_IND,
PELEM_EXEC_STRUCT,
PELEM_LABEL,
PELEM_APPL_DATA,
PELEM_GSE,
PELEM_PICK_ID,
PELEM_POLYLINE_SET3_DATA,
PELEM_FILL_AREA_SET3_DATA,
PELEM_TRI_STRIP3_DATA,

```
        PELEM_QUAD_MESH3_DATA,
        PELEM_SET_OF_FILL_AREA_SET3_DATA,
        PELEM_NUNI_BSP_CURVE,
        PELEM_NUNI_BSP_SURF,
        PELEM_CELL_ARRAY3_PLUS,
        PELEM_TEXT_COLR,
        PELEM_MARKER_COLR,
        PELEM_EDGE_COLR,
        PELEM_LINE_COLR,
        PELEM_CURVE_APPROX_CRIT,
        PELEM_LINE_SHAD_METH,
        PELEM_INT_COLR,
        PELEM_BACK_INT_COLR,
        PELEM_BACK_INT_STYLE,
        PELEM_BACK_INT_STYLE_IND,
        PELEM_REFL_PROPS,
        PELEM_BACK_REFL_PROPS,
        PELEM_INT_SHAD_METH,
        PELEM_BACK_INT_SHAD_METH,
        PELEM_INT_REFL_EQN,
        PELEM_BACK_INT_REFL_EQN,
        PELEM_SURF_APPROX_CRIT,
        PELEM_PARA_SURF_CHARACS,
        PELEM_FACE_DISTING_MODE,
        PELEM_FACE_CULL_MODE,
        PELEM_LIGHT_SRC_STATE,
        PELEM_DCUE_IND,
        PELEM_COLR_MAP_IND,
        PELEM_RENDERING_COLR_MODEL,
        PELEM_NUM_EL_TYPES
    } Pelem_type;
```

*size* Returns the size, in bytes, that the application will have to allocate in order to call INQUIRE ELEMENT CONTENT to retrieve the contents of the specified element from the specified structure. If the element type is such that it is not necessary to allocate any dynamic memory to retrieve its contents, a value of zero is returned.

**ERRORS**

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

201 Ignoring function, the specified structure does not exist

202 Ignoring function, the specified element does not exist

**SEE ALSO**

INQUIRE ELEMENT POINTER (3P)
INQUIRE ELEMENT CONTENT (3P)
ELEMENT SEARCH (3P)

**NAME**

INQUIRE ERROR HANDLING MODE – inquire the current error handling mode

**SYNOPSIS**

**C Syntax**

```
void
pinq_err_hand_mode ( error_ind, mode )
Pint            *error_ind;         OUT error indicator
Perr_mode       *mode;              OUT error mode
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE ERROR HANDLING MODE to determine the current error handling mode. Error handling may be ON or OFF.

See the description of the subroutine SET ERROR HANDLING MODE for more information.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*mode* A pointer to a Perr_mode enumerated type, in which the system returns the current error handling mode. Values for Perr_mode are defined in phigs.h as follows:

PERR_OFF
PERR_ON

**ERRORS**

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

**SEE ALSO**

ERROR HANDLING (3P)
ERROR LOGGING (3P)

**NAME**

INQUIRE GENERALIZED DRAWING PRIMITIVE – inquire generalized drawing primitive attributes

**SYNOPSIS**

**C Syntax**

**void**
**pinq_gdp ( ws_type, gdp, error_ind, num_attrs, attrs )**

| | | |
|---|---|---|
| **Pint** | **ws_type;** | *workstation type* |
| **Pint** | **gdp;** | *GDP function number* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pint** | ***num_attrs;** | *OUT number of attributes* |
| **Pattrs** | **attrs[5];** | *OUT list of attributes* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

INQUIRE GENERALIZED DRAWING PRIMITIVE determines the sets of attributes used by the specified generalized drawing primitive. The representation of the generalized drawing primitive at a given workstation will be controlled by zero or more sets of polyline, polymarker, text, interior, and edge attributes, as indicated by this inquiry function. Whether bundle indices or the corresponding individual attributes are used will depend on the values of the appropriate aspect source flags at traversal time.

**C Input Parameters**

*ws_type* Type of workstation for which to inquire the generalized drawing primitive attributes.

*gdp* The identifier of the generalized drawing primitive for which to inquire attributes.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*num_attrs*

A pointer to an integer that indicates how many attributes are listed in *attrs*.

*attrs* A Pattrs structure indicating which set of attributes are used by the specified generalized drawing primitive on workstations of type *type*. Pattrs is an enumerated type defined in phigs.h as follows:

```
typedef enum {
        PATTR_LINE,         /* polyline attributes */
        PATTR_MARKER,       /* marker attributes */
        PATTR_TEXT,         /* text attributes */
        PATTR_INT,          /* interior attributes */
        PATTR_EDGE          /* edge attributes */
} Pattrs;
```

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the below error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

051 Ignoring function, this information is not yet available for this generic workstation type; open a

workstation of this type and use the specific workstation type

052 Ignoring function, workstation type not recognized by the implementation

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

064 Ignoring function, the specified workstation type is not able to generate the specified generalized drawing primitive

**SEE ALSO**

INQUIRE LIST OF AVAILABLE GENERALIZED DRAWING PRIMITIVES (3P)
PHIGS WORKSTATION DESCRIPTION TABLE (7P)
INQUIRE GENERALIZED DRAWING PRIMITIVE 3 (3P)

**NAME**

INQUIRE GENERALIZED DRAWING PRIMITIVE 3 – inquire generalized drawing primitive3 attributes

**SYNOPSIS**

**C Syntax**

**void**
**pinq_gdp3 ( ws_type, gdp, error_ind, num_attrs, attrs )**

| | | |
|---|---|---|
| **Pint** | **ws_type;** | *workstation type* |
| **Pint** | **gdp;** | *3D GDP function number* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pint** | ***num_attrs;** | *OUT number of attributes* |
| **Pattrs** | **attrs[5];** | *OUT list of attributes* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE GENERALIZED DRAWING PRIMITIVE 3 to determine the sets of attributes used by the specified 3D generalized drawing primitive. The representation of the generalized drawing primitive at a given workstation will be controlled by zero or more sets of polyline, polymarker, text, interior, and edge attributes, as indicated by this inquiry function. Whether bundle indices or the corresponding individual attributes are used will depend on the values of the appropriate aspect source flags at traversal time.

**C Input Parameters**

*ws_type* Type of workstation for which to inquire the generalized drawing primitive 3 attributes.

*gdp* The identifier of the generalized drawing primitive 3 for which to inquire attributes.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*num_attrs*

A pointer to an integer which returns the number of attributes listed in *attrs*.

*attrs* A Pattrs structure in which *num_attrs* values are returned indicating which set of attributes are used by the specified generalized drawing primitive on workstations of type *ws_type*. Pattrs is an enumerated type defined in phigs.h as follows:

```
typedef enum{
        PATTR_LINE,         /* polyline attributes */
        PATTR_MARKER,       /* polymarker attributes */
        PATTR_TEXT,         /* text attributes */
        PATTR_INT,          /* interior attributes */
        PATTR_EDGE          /* edge attributes */
}Pattrs;
```

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the below error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

052 Ignoring function, workstation type not recognized by the implementation

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

059 Ignoring function, the specified workstation does not have output capability (i.e. the workstation category is neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

064 Ignoring function, the specified workstation type is not able to generate the specified generalized drawing primitive

**SEE ALSO**

INQUIRE LIST OF AVAILABLE GENERALIZED DRAWING PRIMITIVES 3 (3P)
PHIGS WORKSTATION DESCRIPTION TABLE (7P)
INQUIRE GENERALIZED DRAWING PRIMITIVE (3P)

**NAME**

INQUIRE GENERALIZED STRUCTURE ELEMENT FACILITIES – inquire list of GSE facilities

**SYNOPSIS**

**C Syntax**

**void**
**pinq_gse_facs ( length, start, error_ind, gse, total_length )**

| | | |
|---|---|---|
| **Pint** | **length;** | *length of application list* |
| **Pint** | **start;** | *starting position* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pgse_id_dep_list** | ***gse;** | *OUT list of GSE ids and their dependencies* |
| **Pint** | ***total_length;** | *OUT length of list in PHIGS* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE GENERALIZED STRUCTURE ELEMENT FACILITIES to obtain a list of the GSE identifiers supported, and whether each is workstation-independent (i.e. supported on all workstations) or workstation-dependent (i.e. supported on some, but not all workstations).

**C Input Parameters**

*length* The number of elements for which memory is allocated in the output parameter *gse*. 0 may be specified, in order to get the *total length* of the list.

*start* Starting position in the list at which to begin the inquiry.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*gse* A pointer to a Pgse_id_dep_list in which the system returns the portion of the *list of GSE identifiers* from the PHIGS description table, starting with *start*. Pgse_id_dep_list is defined in phigs.h as follows:

```
typedef struct {
        Pint            num_id_facs;    /* number of identifiers/dependency element */
        Pgse_id_dep     *id_facs;       /* list of GSE facilities */
} Pgse_id_dep_list;
```

The pointer *gse→id_facs* must be initialized to an array of *length* Pgse_id_dep elements, defined as:

```
typedef struct {
        Pint            id;     /* GSE identifier */
        Pws_dep_ind     ind;    /* WS independent/dependent indicator */
} Pgse_id_dep;
```

Pws_dep_ind is an enumerated type, with values:

| | |
|---|---|
| PWS_INDEP | *Workstation independent* |
| PWS_DEP | *Workstation dependent* |

*total_length*

A pointer to an integer in which to return the total length of the list. This is the value required for *length* if all the items in the list are to be returned.

**ERRORS**

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

**SEE ALSO**

GENERALIZED STRUCTURE ELEMENT (3P)
INQUIRE LIST OF AVAILABLE GENERALIZED STRUCTURE ELEMENTS (3P)
PHIGS WORKSTATION DESCRIPTION TABLE (7P)

**NAME**

INQUIRE HIGHLIGHTING FILTER – inquire inclusion and exclusion name sets for a workstation's highlighting filter

**SYNOPSIS**

**C Syntax**

**void**
**pinq_highl_filter ( ws, store, error_ind, highl_filter )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pstore** | **store;** | *handle to Store object* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pfilter** | ****highl_filter;** | *OUT highlighting filter* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE HIGHLIGHTING FILTER to obtain the *highlighting filter* from a specified PHIGS workstation state list. This is the filter that is compared to the traversal-time *current name set* of each primitive to determine if the primitive is highlighted.

The filter contains an *inclusion set* and an *exclusion set* of names. During traversal, a primitive is eligible for highlighting if at least one name in the *current name set* is in the *inclusion set* and no name in the *current name set* is in the *exclusion set*. Each name in the *name set*, *inclusion set*, and *exclusion set* is a small positive integer.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing data associated with the device state. This buffer is passed as the *store* argument.

The store buffer is a data area managed by PHIGS. While the application is responsible for creating the inital buffer through a call to STORE, PHIGS manages this area such that there is sufficient memory for the specific inquiry. The data record within the store buffer is accessed by the pointer pointed to by *highl_filter*.

*ws* The *workstation identifier* of the workstation whose highlighting filter is to be returned.

*store* The memory buffer PHIGS is to use for storing the information returned. This buffer must exist prior to calling this function (see CREATE STORE (3P)).

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*highl_filter*

A pointer to a pointer to a list of highlighting filters. Pfilter is defined in phigs.h as follows:

```
typedef struct {
    Pint_list       incl_set;       /* inclusion set */
    Pint_list       excl_set;       /* exclusion set */
} Pfilter;
```

Pint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint        num_ints;       /* number of Pints in list */
        Pint        *ints;          /* list of integers        */
} Pint_list;
```

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (in other words, the workstation category is neither OUTPUT, OUTIN, nor MO)

**SEE ALSO**

SET HIGHLIGHTING FILTER (3P)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

INQUIRE HLHSR IDENTIFIER FACILITIES – inquire the available hidden line and hidden surface removal identifiers of a specified workstation type

**SYNOPSIS**

**C Syntax**

**void**
**pinq_hlhsr_id_facs ( type, length, start, error_ind, ids, length_list )**

| | | |
|---|---|---|
| **Pint** | **type;** | *workstation type* |
| **Pint** | **length;** | *length of id list* |
| **Pint** | **start;** | *starting position of id list* |
| **Pint** | ∗**error_ind;** | *OUT error indicator* |
| **Pint_list** | ∗**ids;** | *OUT list of HLHSR identifiers* |
| **Pint** | ∗**length_list;** | *OUT length of id list in PHIGS* |

**Required PHIGS Operating States**

(PHOP, ∗, ∗, ∗)

**DESCRIPTION**

**Purpose**

INQUIRE HLHSR IDENTIFIER FACILITIES obtains the available HLHSR (hidden line and hidden surface removal) identifiers on the specified workstation type. See SET HLHSR IDENTIFIER for a description of how to set the id.

*type* Type of workstation.

*length* The number of *ints* in the *ids* output parameter for which the application has allocated memory. *length* is the number of list elements that the system can return in *ids→ints*. If a value of 0 is used here, no data will be returned in the *ids→ints* list, but the total number of identifiers in the PHIGS state list will be returned in *length_list*.

*start* Starting position of inquiry into the PHIGS state list of current identifiers. The elements of the list of identifiers, beginning with the item number specified by *start*, are copied sequentially into *ids→ints* until *ids→ints* is full or all the identifiers have been copied.

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*ids* A pointer to a Pint_list data structure in which the system returns a list of the HLHSR identifiers available on this workstation type. The returned list starts with the *start* item in the list of identifiers and returns the next *length* items. Pint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint   num_ints;   /* number of identifiers in list */
        Pint   *ints;      /* list of identifiers */
} Pint_list;
```

Prior to calling this function, the *ints* field must contain a pointer to an application supplied buffer. This buffer must be able to hold *length ids*. *num_ints* is the number of *ids* returned in the buffer.

Defined *ids* are:

| | | |
|---|---|---|
| 0 | PHIGS_HLHSR_ID_OFF | *Do not do Hidden Line or Hidden Surface removal* |
| 1 | PHIGS_HLHSR_ID_ON | *Do Hidden Line and Hidden Surface removal according to the current HLHSR Mode* |

*length_list*

A pointer to an integer in which the system returns the total number of elements in the PHIGS state list of currently used identifiers. This is the value required for *length* if all identifiers are to be returned.

**ERRORS**

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

052 Ignoring function, workstation type not recognized by the implementation

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

057 Ignoring function, specified workstation is of category MI

062 Ignoring function, this information is not available for this MO workstation type

**SEE ALSO**

INQUIRE WORKSTATION CONNECTION AND TYPE (3P)
INQUIRE HLHSR MODE (3P)
INQUIRE HLHSR MODE FACILITIES (3P)
SET HLHSR MODE (3P)
SET HLHSR IDENTIFIER (3P)

**NAME**

INQUIRE HLHSR MODE – inquire a workstation's hidden line/hidden surface removal mode

**SYNOPSIS**

**C Syntax**

**void**
**pinq_hllhsr_mode ( ws, error_ind, state, cur_mode, req_mode )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pupd_st** | ***state;** | *OUT HLHSR update state* |
| **Pint** | ***cur_mode;** | *OUT current HLHSR mode* |
| **Pint** | ***req_mode;** | *OUT requested HLHSR mode* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE HLHSR MODE to determine the current HLHSR (hidden line and hidden surface removal) mode from a specified PHIGS workstation state list.

**C Input Parameters**

*ws* The *workstation identifier* of the workstation whose workstation state list is queried.

**C Output Parameters**

*err_ind* A pointer to the location to store the error number of any error detected by this function.

*state* A pointer to a location in which the system returns the *HLHSR update state*. Pupd_st is one of the values PUPD_NOT_PEND (*Not Pending*) or PUPD_PEND (*Pending*).

*cur_mode*
A pointer to a location in which the system returns the current *HLHSR mode*.

*req_mode*
A pointer to a location in which the system returns the *requested HLHSR mode*. The requested mode can differ from the current mode if the update state is PENDING.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

057 Ignoring function, specified workstation is of category MI

**SEE ALSO**

SET HLHSR MODE (3P)
SET HLHSR IDENTIFIER (3P)
INQUIRE HLHSR FACILITIES (3P)

**NAME**

INQUIRE HLHSR MODE FACILITIES – inquire the available hidden line and hidden surface removal modes of a specified workstation type

**SYNOPSIS**

**C Syntax**

**void**
**pinq_hlhsr_mode_facs ( type, length, start, error_ind, modes, length_list )**

| | | |
|---|---|---|
| **Pint** | **type;** | *workstation type* |
| **Pint** | **length;** | *length of mode list* |
| **Pint** | **start;** | *starting position of mode list* |
| **Pint** | ∗**error_ind;** | *OUT error indicator* |
| **Pint_list** | ∗**modes;** | *OUT list of HLHSR modes* |
| **Pint** | ∗**length_list;** | *OUT length of mode list in PHIGS* |

**Required PHIGS Operating States**

(PHOP, ∗, ∗, ∗)

**DESCRIPTION**

**Purpose**

INQUIRE HLHSR MODE FACILITIES obtains the available HLHSR (hidden line and hidden surface removal) modes on the specified workstation type. See SET HLHSR MODE for a description of how to select the mode.

**C Input Parameters**

*type* Type of workstation.

*length* The number of *ints* in the *modes* output parameter for which the application has allocated memory. *length* is the number of list elements that the system can return in *modes→ints*. If a value of 0 is used here, no data will be returned in the *modes→ints* list, but the total number of modes in the PHIGS state list will be returned in *length_list*.

*start* Starting position of inquiry into the PHIGS state list of current mode. The elements of the list of mode, beginning with the item number specified by *start*, are copied sequentially into *mode→ints* until *modes→ints* is full or all the modes have been copied.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*modes* A pointer to a Pint_list data structure in which the system returns a list of the HLHSR modes available on this workstation type. The returned list starts with the *start* item in the list of modes and returns the next *length* items. Pint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint   num_ints;   /* number of modes in list */
        Pint   *ints;      /* list of modes */
} Pint_list;
```

Possible modes are:

| | | |
|---|---|---|
| 0 | PHIGS_HLHSR_MODE_NONE | *Disable HLHSR* |
| 1 | PHIGS_HLHSR_MODE_ZBUFF | *Enable HLHSR using Z-buffer* |
| 2 | PHIGS_HLHSR_MODE_PAINTERS | *Enable HLHSR using painter* |
| 3 | PHIGS_HLHSR_MODE_SCANLINE | *Enable HLHSR using scanline* |
| 4 | PHIGS_HLHSR_MODE_LINE_ONLY | *Enable hidden line removal only* |

*length_list*

A pointer to an integer in which the system returns the total number of elements in the PHIGS state list of currently used modes. This is the value required for *length* if all modes are to be returned.

**ERRORS**

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

052 Ignoring function, workstation type not recognized by the implementation

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

057 Ignoring function, specified workstation is of category MI

062 Ignoring function, this information is not available for this MO workstation type

**SEE ALSO**

INQUIRE WORKSTATION CONNECTION AND TYPE (3P)
INQUIRE HLHSR MODE (3P)
INQUIRE HLHSR IDENTIFIER (3P)
SET HLHSR MODE (3P)
SET HLHSR IDENTIFIER (3P)

**NAME**

INQUIRE INPUT QUEUE OVERFLOW – inquire the *input queue overflow* state of the PHIGS error state list

**SYNOPSIS**

**C Syntax**

**void**
**pinq_in_overf ( error_ind, ws, class, dev )**

| | | |
|---|---|---|
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pint** | ***ws;** | *OUT workstation identifier* |
| **Pin_class** | ***class;** | *OUT input class* |
| **Pint** | ***dev;** | *OUT input device number* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE INPUT QUEUE OVERFLOW to retrieve the *input queue overflow* information from the PHIGS error state list.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*ws* A pointer to a Pint in which to store the workstation identifier of the workstation associated with the input device that caused the overflow.

*class* A pointer to a Pin_class structure in which to store the class of the input device that caused the overflow.

*Pin_class* is an enumerated type that may assume the following values:

PIN_NONE
PIN_LOC
PIN_STROKE
PIN_VAL
PIN_CHOICE
PIN_PICK
PIN_STRING

*dev* A pointer to a Pint in which to store the number of the input device that overflowed the input queue.

**Execution**

When the input queue overflows, the identification of one of the logical input devices that caused the overflow is placed in the error state list. This information remains in the error state list until:

- The workstation associated with the input device is closed
- PHIGS is closed
- This function is called

Calling this function clears the input queue overflow information in the error state list. If the input queue has not overflowed since this function or OPEN PHIGS was last called the appropriate error will be returned in *error indicator*.

When the input queue overflows, no more events can be added to it until AWAIT EVENT is called with the queue empty, i.e., all events must be removed from the queue, and AWAIT EVENT called **while the queue is empty** before any new events can be added.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

257 Ignoring function, input queue has not overflowed

258 Warning, input queue has overflowed, but associated workstation has been closed

**SEE ALSO**

AWAIT EVENT (3P)
FLUSH DEVICE EVENTS (3P)

**NAME**

INQUIRE INTERIOR FACILITIES – inquire list of workstation interior facilities from the workstation description table

**SYNOPSIS**

**C Syntax**

**void**
**pinq_int_facs ( type, h_len, h_st, error_ind, facil, tot_h_len )**

| | | |
|---|---|---|
| **Pint** | **type;** | *workstation type* |
| **Pint** | **h_len;** | *length of hatch style list* |
| **Pint** | **h_st;** | *starting position* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pint_facs** | ***facil;** | *OUT interior facilities* |
| **Pint** | ***tot_h_len;** | *OUT len of hatch list in PHIGS* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

INQUIRE INTERIOR FACILITIES obtains a list of the available interior styles and hatch styles for the specified type of workstation.

**C Input Parameters**

*type* Get the interior facilities for this *workstation type*.

*h_len* The number of elements for which memory is allocated in the output parameter *facil→hatches.ints*. 0 may be specified in order to get the *tot_h_len*.

*h_st* Starting position in the list of hatch styles at which to begin the inquiry.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*facil* A pointer to an allocated data structure in which the system returns a portion of the list of available hatch styles, starting with *h_st*. The structure is defined as follows:

```
typedef struct {
        Pint            num_int_styles;     /* number of interior styles */
        Pint_style      *int_styles[5];     /* list of available interior styles */
        Pint_list       hatch_styles;       /* list of available hatch styles */
        Pint            num_pred_inds;      /* number of predefined interior bundles */
} Pint_facs;
```

Pint_style is an enumerated type for the following *interior styles*; one of:

| | |
|---|---|
| PSTYLE_HOLLOW | *Hollow* |
| PSTYLE_SOLID | *Solid* |
| PSTYLE_PAT | *Patterned* |
| PSTYLE_HATCH | *Hatched* |
| PSTYLE_EMPTY | *Empty* |

*Pint_list* is defined in *phigs.h* as follows:

```
typedef struct {
        Pint     num_ints;      /* number of Pints in list */
        Pint     *ints;         /* list of integers */
} Pint_list;
```

Prior to calling this function, the *ints* field of the Pint_list structure must contain a pointer to an

application supplied buffer. This buffer must be at least as large as the *h_len* parameter.

*tot_h_len*

A pointer to an integer in which the system returns the total number of items in the workstation list of hatch styles. This is the value required by *h_len* if all the items in the list are to be returned.

**ERRORS**

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

051 Ignoring function, this information is not yet available for this generic workstation type; open a workstation of this type and use the specific workstation type

052 Ignoring function, workstation type not recognized by the implementation

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

**SEE ALSO**

.PHIGS WORKSTATION DESCRIPTION TABLE (7P)
.INQUIRE PREDEFINED INTERIOR REPRESENTATION (3P+)

**NAME**

INQUIRE INTERIOR FACILITIES PLUS – inquire extended interior facilities of a workstation type

**SYNOPSIS**

**C Syntax**

**void**
**pinq_int_facs_plus ( type, is_length, is_start, hs_length, hs_start, re_length, re_start, sh_length, sh_start, error_ind, facil, tot_is_length, tot_hs_length, tot_re_length, tot_sh_length )**

| | | |
|---|---|---|
| **Pint** | **type;** | *workstation type* |
| **Pint** | **is_length;** | *length of application's interior style list* |
| **Pint** | **is_start;** | *starting position* |
| **Pint** | **hs_length;** | *length of application's hatch style list* |
| **Pint** | **hs_start;** | *starting position* |
| **Pint** | **re_length;** | *length of application's reflectance equation list* |
| **Pint** | **re_start;** | *starting position* |
| **Pint** | **sh_length;** | *length of application's shading method list* |
| **Pint** | **sh_start;** | *starting position* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pint_facs_plus** | ***facil;** | *OUT extended interior facilities* |
| **Pint** | ***tot_is_length;** | *OUT total length of interior styles list* |
| **Pint** | ***tot_hs_length;** | *OUT total length of hatch styles list* |
| **Pint** | ***tot_re_length;** | *OUT total length of reflectance equation list* |
| **Pint** | ***tot_sh_length;** | *OUT total length of shading method list* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

INQUIRE INTERIOR FACILITIES PLUS returns the extended interior support available from a specified workstation type.

**C Input Parameters**

All of the following data types are predefined in phigs.h.

*type* The workstation type in question.

*is_length*
The number of entries to return from the list of interior styles.

*is_start* Starting position of the portion of the list to return.

*hs_length*
The number of entries to return from the list of hatch styles.

*hs_start* Starting position of the portion of the list to return.

*re_length*
The number of entries to return from the list of reflectance equations.

*re_start* Starting position of the portion of the list to return.

*sh_length*
The number of entries to return from the list of interior shading method.

*sh_start* Starting position of the portion of the list to return.

**C Output Parameters**

*error_ind*
A pointer to the location to store the error number of any error detected by this function.

*facil* A pointer to a Pint_facs_plus structure which returns the requested information. Pint_facs_plus is

defined in phigs.h as follows:

```
typedef struct {
        Pint        num_int_styles;   /* number of interior styles */
        Pint_style  *int_styles;      /* list of available interior styles */
        Pint_list   hatch_styles;     /* list of available hatch styles */
        Pint        num_pred_inds;    /* number of predefined interior indices */
        Pint_list   refl_eqns;        /* list of available reflectence equations */
        Pint_list   shad_meths;       /* list of available shading methods */
} Pint_facs_plus;
```

Pint_style is defined in phigs.h as follows:

```
typedef enum {
        PSTYLE_HOLLOW,
        PSTYLE_SOLID,
        PSTYLE_PAT,
        PSTYLE_HATCH,
        PSTYLE_EMPTY,
} Pint_style;
```

Pint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint    num_ints;    /* number of Pints in list */
        Pint    *ints;       /* list of integers */
} Pint_list;
```

Prior to calling this function, the *ints* field of the Pint_list structures and interiors must contain a pointer to an application supplied buffer. This buffer must be at least as large as the corresponding *length* parameter.

*tot_is_length*
: A pointer to the location to store the total length of the interior style list in the workstation description table.

*tot_hs_length*
: A pointer to the location to store the total length of the hatch style list in the workstation description table.

*tot_re_length*
: A pointer to the location to store the total length of the reflectance equation list in the workstation description table.

*tot_sh_length*
: A pointer to the location to store the total length of the interior shading method list in the workstation description table.

**Execution**

If the inquired information is available, the error indicator is returned as zero and the requested information is returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the below error numbers to indicate the reason for non-availability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

052 Ignoring function, workstation type not recognized by the implementation

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

059 Ignoring function, the specified workstation does not have output capability (i.e. the workstation category is neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

**SEE ALSO**

SET INTERIOR REPRESENTATION PLUS (3P+)
INQUIRE INTERIOR REPRESENTATION PLUS (3P+)
INQUIRE PREDEFINED INTERIOR REPRESENTATION PLUS (3P+)

**NAME**

INQUIRE INTERIOR REPRESENTATION – inquire an interior representation on a workstation

**SYNOPSIS**

**C Syntax**

```
void
pinq_int_rep ( ws, index, type, error_ind, rep )
Pint            ws;             workstation identifier
Pint            index;          interior index
Pinq_type       type;           type of returned value
Pint            *error_ind;     OUT error indicator
Pint_bundle     *rep;           OUT interior representation
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE INTERIOR REPRESENTATION to determine the current attribute values for a specified entry in a specified workstation's table of defined interior representations.

See the description of the subroutine SET INTERIOR REPRESENTATION for information on the meaning of these attribute values.

**C Input Parameters**

*ws* Workstation identifier.

*index* Entry to be returned from the workstation's table of interior representations; if this entry is not present in the table and the *type* parameter is REALIZED, the representation for interior index 1 is returned.

*type* An enumerated value specifying whether the inquired values are to be returned as the values originally specified by the application (SET), or as the values actually being used by the workstation if any of the application-specified values had to be mapped to ones available on the workstation (REALIZED). Valid values are defined in phigs.h as:

| | |
|---|---|
| PINQ_SET | *Return Application-specified Value* |
| PINQ_REALIZED | *Return Value Available on the Workstation* |

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*rep* A pointer to a Pint_bundle structure in which the system returns the interior representation at *index* in the workstation's table of interior representations. Pint_bundle is defined in phigs.h as follows:

```
typedef struct {
        Pint_style  style;      /* interior style */
        Pint        style_ind;  /* interior style index */
        Pint        colr_ind;   /* interior colour index */
} Pint_bundle;
```

Pint_style is defined in phigs.h as:

```
typedef enum {
        PSTYLE_HOLLOW,
        PSTYLE_SOLID,
        PSTYLE_PAT,
        PSTYLE_HATCH,
        PSTYLE_EMPTY,
} Pint_style;
```

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the below error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (in other words, the workstation category is neither OUTPUT, OUTIN, nor MO)

100 Ignoring function, the bundle index value is less than one

101 Ignoring function, the specified representation has not been defined

**SEE ALSO**

SET INTERIOR REPRESENTATION (3P)
INQUIRE PREDEFINED INTERIOR REPRESENTATION (3P+)
INQUIRE INTERIOR REPRESENTATION PLUS (3P+)

**NAME**

INQUIRE INTERIOR REPRESENTATION PLUS – inquire the values of an extended interior representation on a workstation

**SYNOPSIS**

**C Syntax**

```
void
pinq_int_rep_plus ( ws, index, type, error_ind, rep )
Pint            ws;             workstation identifier
Pint            index;          interior index
Pinq_type       type;           type of returned value
Pint            *error_ind;     OUT error indicator
Pint_bundle_plus *rep;          OUT interior representation
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

INQUIRE INTERIOR REPRESENTATION PLUS returns the values of a specified entry in a workstation extended interior bundle table.

**C Input Parameters**

*ws* The workstation identifier.

*index* Index of the entry to be returned. If this entry is not present in the table, and the type of returned value parameter is PINQ_REALIZED, the representation for interior index one is returned.

*type* An enumerated value specifying whether the values to be returned are those originally specified by the application (PINQ_SET), or those resulting after PHIGS mapped them to ones available on the workstation (PINQ_REALIZED). A Pinq_type structure is defined as:

```
typedef enum {
        PINQ_SET,
        PINQ_REALIZED
} Pinq_type;
```

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*rep* A pointer to a Pint_bundle_plus structure that returns the specified extended interior representation. Pint_bundle_plus is defined as:

```
typedef struct {
        Pint_style      style;          /* interior style */
        Pint            style_ind;      /* interior style index */
        Pgcolr          colr;           /* interior colour */
        Pint            refl_eqn;       /* reflectance equation */
        Pint            shad_meth;      /* shading method */
        Prefl_props     refl_props;     /* area properties */
        Pint_style      back_style;     /* interior style */
        Pint            back_style_ind; /* interior style index */
        Pgcolr          back_colr;      /* interior colour */
        Pint            back_refl_eqn;  /* back reflectance equation */
        Pint            back_shad_meth; /* back shading method */
        Prefl_props     back_refl_props;/* back area properties */
        Pint            approx_type;    /* approximation method */
        Pfloat          approx_val[2];  /* approximation values, u and v */
```

```
} Pint_bundle_plus;
```

Pint_style is defined as:

```
typedef enum {
        PSTYLE_HOLLOW,
        PSTYLE_SOLID,
        PSTYLE_PAT,
        PSTYLE_HATCH,
        PSTYLE_EMPTY,
} Pint_style;
```

Pgcolr is defined as:

```
typedef struct {
        Pint    type;    /* indirect, RGB, CIE, HSV, HLS */
        union {
                Pint    ind;    /* index in workstation colour bundle table */
                struct {
                        Pfloat  x;      /* red, hue, etc. */
                        Pfloat  y;      /* green, saturation, lightness, etc. */
                        Pfloat  z;      /* blue, value, saturation, etc. */
                } general;
        } val;
} Pgcolr;
```

Prefl_props is defined as:

```
typedef struct {
        Pfloat  ambient_coef;       /* ambient reflectance coefficient */
        Pfloat  diffuse_coef;       /* diffuse reflectance coefficient */
        Pfloat  specular_coef;      /* specular reflectance coefficient */
        Pgcolr  specular_colr;      /* specular colour */
        Pfloat  specular_exp;       /* specular exponent */
        Pfloat  transpar_coef;      /* transparency coefficient */
} Prefl_props;
```

The values for ambient, diffuse, specular, and transparency coefficients must all be in the range [0,1]. The specular exponent must be greater than zero.

Constants defined for colour type are:

| | | |
|---|---|---|
| 0 | PINDIRECT | *Indirect* |
| 1 | PMODEL_RGB | *Red, Green, Blue* |
| 2 | PMODEL_CIELUV | *CIE* |
| 3 | PMODEL_HSV | *Hue, Saturation, Value* |
| 4 | PMODEL_HLS | *Hue, Lightness, Saturation* |

The index member of the *val* union is used for type PINDIRECT. The general member is used for the other types.

Predefined reflectance equation values are:

| | | |
|---|---|---|
| 1 | PREFL_NONE | *No Reflectance Calculation Performed* |

| | | |
|---|---|---|
| 2 | PREFL_AMBIENT | *Use Ambient Term* |
| 3 | PREFL_AMB_DIFF | *Use Ambient and Diffuse Terms* |
| 4 | PREFL_AMB_DIFF_SPEC | *Use Ambient, Diffuse, and Specular Terms* |

Predefined constants for interior shading method are:

| | | |
|---|---|---|
| 1 | PSD_NONE | *No Shading* |
| 2 | PSD_COLOUR | *Colour Interpolation Shading* |
| 3 | PSD_DOT_PRODUCT | *Dot Product Interpolation Shading* |
| 4 | PSD_NORMAL | *Normal Interpolation Shading* |

Constants defined for surface approximation type are:

| | |
|---|---|
| 1 | PSURF_WS_DEP |
| 2 | PSURF_CONSTANT_PARAMETRIC_BETWEEN_KNOTS |
| 3 | PSURF_CHORDAL_SIZE_WC |
| 4 | PSURF_CHORDAL_SIZE_NPC |
| 5 | PSURF_CHORDAL_SIZE_DC |
| 6 | PSURF_PLANAR_DEVIATION_WC |
| 7 | PSURF_PLANAR_DEVIATION_NPC |
| 8 | PSURF_PLANAR_DEVIATION_DC |

See SET SURFACE APPROXIMATION CRITERIA for a description of their meaning.

**Execution**

If the inquired information is available, the error indicator is returned as zero and the requested information is returned in the output parameters.

If the specified interior index is not present in the interior bundle table on the workstation and the specified type of returned value is *Realized*, the representation for interior index one is returned.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the error numbers below to indicate the reason for non-availability.

The extended interior bundle table contains bundled entries of the PHIGS and PHIGS Extension interior attributes. During traversal, the attribute values of these bundles are used when the corresponding aspect source flag in the workstation state list is set to BUNDLED.

The PHIGS interior bundle table is a portion of the extended table. The function INQUIRE INTERIOR REPRESENTATION returns the following attributes from the extended table: interior style, interior style index, and colour index.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

| | |
|---|---|
| 003 | Ignoring function, function requires state (PHOP, WSOP, *, *) |
| 054 | Ignoring function, the specified workstation is not open |
| 059 | Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT, OUTIN, nor MO) |
| 100 | Ignoring function, the bundle index value is less than one |
| 101 | Ignoring function, the specified representation has not been defined |

**SEE ALSO**

SET INTERIOR REPRESENTATION PLUS (3P+)
INQUIRE INTERIOR REPRESENTATION (3P)
INQUIRE PREDEFINED INTERIOR REPRESENTATION PLUS (3P+)

**NAME**

INQUIRE INVISIBILITY FILTER – inquire inclusion and exclusion name sets for a workstation's invisibility filter

**SYNOPSIS**

**C Syntax**

**void**
**pinq_invis_filter ( ws, store, error_ind, invis_filter)**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pstore** | **store;** | *handle to Store object* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pfilter** | ****invis_filter** | *OUT invisibility filter* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE INVISIBILITY FILTER to obtain the *invisibility filter* from a specified PHIGS workstation state list. This is the filter that is compared to the traversal-time *current name set* of each primitive to determine if the primitive is invisible.

The filter contains an *inclusion set* and an *exclusion set* of names. During traversal, a primitive is eligible for invisibility if at least one name in the *current name set* is in the *inclusion set* and no name in the *current name set* is in the *exclusion set*. Each name in the *name set*, *inclusion set*, and *exclusion set* is a small positive integer.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing data associated with the device state. This buffer is passed as the *store* argument.

The store buffer is a data area managed by PHIGS. While the application is responsible for creating the inital buffer through a call to STORE, PHIGS manages this area such that there is sufficient memory for the specific inquiry. The data record within the store buffer is accessed by the pointer pointed to by *invis_filter*.

*ws* The *workstation identifier* of the workstation whose invisibility filter is to be returned.

*store* The memory buffer PHIGS is to use for storing the information returned. This buffer must exist prior to calling this function (see CREATE STORE (3P)).

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*invis_filter*

A pointer to a pointer that points to the invisibility filter. Pint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint_list       incl_set;       /* inclusion set */
        Pint_list       excl_set;       /* exclusion set */
} Pfilter;
```

Pint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint            num_ints;       /* number of Pints in list */
        Pint            *ints;          /* list of integers        */
} Pint_list;
```

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT, OUTIN, nor MO)

**SEE ALSO**

SET INVISIBILITY FILTER (3P)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

INQUIRE LIGHT SOURCE FACILITIES – inquire list of workstation light source facilities for a specified workstation type

**SYNOPSIS**

**C Syntax**

```
void
pinq_light_src_facs ( type, length, start, error_ind, facilities, total_length)
Pint            type;           workstation type
Pint            length;         length of application list
Pint            start;          starting position
Pint            *error_ind;     OUT error indicator
Plight_src_facs *facilities;    OUT light source facilities
Pint            *total_length;  OUT length of list in PHIGS
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

INQUIRE LIGHT SOURCE FACILITIES obtains a list of workstation light source facilities.

**C Input Parameters**

*type* The workstation type for which the light source facilities are being inquired.

*length* The number of *ints* in the *facilites→types* output parameter for which the application has allocated memory. *length* is the number of list elements that the system can return in *facilites→types.ints*. If a value of 0 is used here, no data will be returned in the *facilites→types.ints* list, but the total number of elements will be returned in *total_length*.

*start* Starting position of inquiry. The elements in the list, beginning with the item number specified by *start*, are copied sequentially into *facilites→types.ints* until *facilites→types.ints* is full or all the elements have been copied.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number.

*facilities*

A pointer to the location in which to return the light source facilities. Plight_src_facs is defined as:

```
typedef struct {
        Pint_list   types;          /* types of light source supported */
        Pint        max;            /* max. no. of simultaneously active lights */
        Pint        num_pred_inds   /* number of predefined bundles */
} Plight_src_facs;
```

*types* is a pointer to the types of light sources that are supported:

1 PLIGHT_AMBIENT
2 PLIGHT_DIRECTIONAL
3 PLIGHT_POSITIONAL
4 PLIGHT_SPOT

Pint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint        num_ints;       /* number of Pints in list */
        Pint        *ints;          /* list of integers    */
} Pint_list;
```

The *num_ints* component specifies the number of elements in the list. The *ints* component is a pointer to a list *num_ints* long.

Prior to calling this function, the *ints* field of the Pint_list structure must contain a pointer to an application supplied buffer. This buffer must be at least as large as the corresponding length parameter.

The light types supported are:

| | | |
|---|---|---|
| 1 | Ambient | A light source that affects all surface primitives uniformly. Ambient light sources have only a color attribute. |
| 2 | WCS_Vector | A light source that is specified in world coordinates with a color and a direction vector. |
| 3 | WCS_Point | A light source that is specified in world coordinates with a color, a position, and two attenuation coefficients. |
| 4 | WCS_Spot | A light source that is specified in world coordinates with a color, a position, a direction vector, a concentration exponent, two attenuation coefficients, and a spread angle. |

*total_length*

A pointer to an integer in which the system returns the total number of elements in the list. This is the value required for *length* if all elements in the list are to be returned.

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the following error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

051 Ignoring function, this information unavailable for this workstation type

052 Ignoring function, workstation type not recognized by the implementation

059 Ignoring function, the specified workstation does not have output capability (i.e. the workstation category is neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

**SEE ALSO**

SET LIGHT SOURCE REPRESENTATION (3P+)
INQUIRE LIST OF LIGHT SOURCE INDICES (3P+)
INQUIRE LIGHT SOURCE REPRESENTATION (3P+)
INQUIRE PREDEFINED LIGHT SOURCE REPRESENTATION (3P+)

**NAME**

INQUIRE LIGHT SOURCE REPRESENTATION – inquire light source representation for a given workstation

**SYNOPSIS**

**C Syntax**

**void**
**pinq_light_src_rep ( ws, index, type, error_ind, rep )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **index;** | *light source index* |
| **Pinq_type** | **type;** | *type of returned value* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Plight_src_bundle** | ***rep;** | *OUT light source representation* |

**Required PHIGS Operating States**

(PHOP, WSOP, ∗, ∗)

**DESCRIPTION**

**Purpose**

INQUIRE LIGHT SOURCE REPRESENTATION returns the attribute values of a specified entry in a workstation extended light source bundle table. See SET LIGHT SOURCE REPRESENTATION for a full description of the contents of the bundle table entry and a description of the extended light source bundle table.

**C Input Parameters**

*ws* The workstation identifier.

*index* The light source index.

*type* An enumerated value specifying whether the values to be returned are those originally specified by the application (PINQ_SET), or those resulting after PHIGS mapped them to ones available on the workstation (PINQ_REALIZED). A Pinq_type structure is defined as:

```
typedef enum {
        PINQ_SET,
        PINQ_REALIZED
} Pinq_type;
```

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*rep* A pointer to the location in which to return the light source representation. The Plight_src_bundle data structure contains values needed to define the specified type of light source representation. Plight_src_bundle is defined as:

```
typedef struct {
        Pint            type;       /* light source type */
        Plight_src_rec  rec;        /* light source data record */
} Plight_src_bundle;
```

Plight_src_rec is defined as:

```
typedef union {
        Pamb_light_src_rec      ambient;
        Pdir_light_src_rec      directional;
        Ppos_light_src_rec      positional;
        Pspot_light_src_rec     spot;
} Plight_src_rec;
```

Pamb_light_src_rec is defined as:

```
typedef struct {
        Pgcolr  colr;   /* light source colour */
} Pamb_light_src_rec;
```

Pgcolr is defined as:

```
typedef struct {
        Pint    type;   /* indirect, RGB, CIE, HSV, HLS */
        union {
                Pint    ind;    /* index in workstation colour bundle table */
                struct {
                        Pfloat  x;      /* red, hue, etc. */
                        Pfloat  y;      /* green, saturation, lightness, etc. */
                        Pfloat  z;      /* blue, value, saturation, etc. */
                } general;
        } val;
} Pgcolr;
```

Pdir_light_src_rec is defined as:

```
typedef struct {
        Pgcolr  colr;   /* light source colour */
        Pvec3   dir;    /* light source direction */
} Pdir_light_src_rec;
```

Pvec3 is defined as:

```
typedef struct {
        Pfloat  delta_x;    /* x magnitude */
        Pfloat  delta_y;    /* y magnitude */
        Pfloat  delta_z;    /* z magnitude */
} Pvec3;
```

Ppos_light_src_rec is defined as:

```
typedef struct {
        Pgcolr      colr;       /* light source colour */
        Ppoint3     pos;        /* light source position */
        Pfloat      coef[2];    /* attenuation coefficients */
} Ppos_light_src_rec;
```

Ppoint3 is defined as:

```
typedef struct {
        Pfloat    x;        /* x coordinate */
        Pfloat    y;        /* y coordinate */
        Pfloat    z;        /* z coordinate */
} Ppoint3;
```

Pspot_light_src_rec is defined as:

```
typedef struct {
        Pgcolr    colr;     /* light source colour */
        Ppoint3   pos;      /* light source position */
        Pvec3     dir;      /* light source direction */
        Pfloat    exp;      /* concentration exponent */
        Pfloat    coef[2];  /* attenuation coefficients */
        Pfloat    angle;    /* spread angle */
} Pspot_light_src_rec;
```

**Execution**

If the specified light source index is not present in the light source table on the workstation and the specified type of returned value is *Realized*, the representation for light source index one is returned.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the following error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

003 Ignoring function, function requires state (PHOP,WSOP,*,*)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (i.e. the workstation category is neither OUTPUT, OUTIN, nor MO)

101 Ignoring function, the specified representation has not been defined

614 Ignoring function, the light source index is less than one

**SEE ALSO**

SET LIGHT SOURCE STATE (3P+)
SET LIGHT SOURCE REPRESENTATION (3P+)
INQUIRE LIST OF LIGHT SOURCE INDICES (3P+)
INQUIRE LIGHT SOURCE FACILITIES (3P+)
INQUIRE PREDEFINED LIGHT SOURCE REPRESENTATION (3P+)

**NAME**

INQUIRE LIST OF AVAILABLE GENERALIZED DRAWING PRIMITIVES – inquire list of workstation available generalized drawing primitives

**SYNOPSIS**

**C Syntax**

**void**
**pinq_list_avail_gdp ( type, length, start, error_ind, gdps, total_length )**

| | | |
|---|---|---|
| **Pint** | **type;** | *workstation type* |
| **Pint** | **length;** | *length of application list* |
| **Pint** | **start;** | *starting position* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pint_list** | ***gdps;** | *OUT list of GDP identifiers* |
| **Pint** | ***total_length;** | *OUT length of list in PHIGS* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE LIST OF AVAILABLE GENERALIZED DRAWING PRIMITIVES to obtain a list of the identifiers of the generalized drawing primitives (GDP)s available on the specified workstation type.

**C Input Parameters**

*type* Workstation type.

*length* The number of *ints* items in the *gdps* output parameter for which the application has allocated memory. The *length* is the number of list elements (GDP identifiers) that the system can return in the list of integers. If a value of zero is used here, no data will be returned in the list of integers, but the total number of GDPs available on the specified workstation type will be returned in *total_length*.

*start* Starting position in the workstation type list of generalized drawing primitive ids at which the inquiry is to begin. The elements of the list of GDP identifiers, beginning with the item number specified by *start*, are copied sequentially into the list of integers until it is full or all the GDP identifiers have been copied.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*gdps* A pointer to a Pint_list structure in which the system returns the list of identifiers of the GDPs available on the specified type of workstation. Pint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint    num_ints;    /* number of Pints in list */
        Pint    *ints;       /* list of integers */
} Pint_list;
```

The *num_ints* component specifies the number of GDP identifiers in the list. The *ints* component is a pointer to a list, *num_ints* long, of the GDP identifiers.

The application must allocate memory for *length* elements in the list of integers.

*total_length*

A pointer to an integer in which the total number of elements in the specified workstation type list of available GDPs is returned. This is the value required for *length* if all GDP identifiers are to be returned.

**ERRORS**

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

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

052 Ignoring function, workstation type not recognized by the implementation

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

**SEE ALSO**

PHIGS WORKSTATION DESCRIPTION TABLE (7P)
INQUIRE GENERALIZED DRAWING PRIMITIVE (3P)

**NAME**

INQUIRE LIST OF AVAILABLE GENERALIZED DRAWING PRIMITIVES 3 – inquire list of workstation available 3D generalized drawing primitives

**SYNOPSIS**

**C Syntax**

**void**
**pinq_list_avail_gdp3 ( type, length, start, error_ind, gdps, total_length )**

| | | |
|---|---|---|
| **Pint** | **type;** | *workstation type* |
| **Pint** | **length;** | *length of application list* |
| **Pint** | **start;** | *starting position* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pint_list** | ***gdps;** | *OUT list of 3D GDP identifiers* |
| **Pint** | ***total_length;** | *OUT length of list in PHIGS* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE LIST OF AVAILABLE GENERALIZED DRAWING PRIMITIVES 3 to obtain a list of the identifiers of the 3D generalized drawing primitives (GDP)s available on the specified workstation type.

**C Input Parameters**

*type* Workstation type.

*length* The number of *ints* items in the *gdps* output parameter for which the application has allocated memory. The *length* is the number of list elements (GDP 3 identifiers) that the system can return in the list of integers. If a value of zero is used here, no data will be returned in the list of integers, but the total number of GDP 3s available on the specified workstation type will be returned in *total_length*.

*start* Starting position in the workstation type list of 3D generalized drawing primitive ids at which the inquiry is to begin. The elements of the list of GDP 3 identifiers, beginning with the item number specified by *start*, are copied sequentially into the list of integers until it is full or all the GDP 3 identifiers have been copied.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*gdps* A pointer to a Pint_list structure in which the system returns the list of identifiers of the GDP 3s available on the specified type of workstation. Pint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint    num_ints;    /* number of Pints in list */
        Pint    *ints;       /* list of integers */
} Pint_list;
```

The *num_ints* component specifies the number of GDP 3 identifiers in the list. The *ints* component is a pointer to a list, *num_ints* long, of the GDP 3 identifiers.

The application must allocate memory for *length* elements in the list of integers.

*total_length*

A pointer to an integer in which the total number of elements in the specified workstation type list of available GDP 3s is returned. This is the value required for *length* if all GDP 3 identifiers are to be returned.

**ERRORS**

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

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

052 Ignoring function, workstation type not recognized by the implementation

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

**SEE ALSO**

PHIGS WORKSTATION DESCRIPTION TABLE (7P)
INQUIRE GENERALIZED DRAWING PRIMITIVE 3 (3P)

**NAME**

INQUIRE LIST OF AVAILABLE WORKSTATION TYPES– inquire the current list of workstation types defined in the PHIGS description table

**SYNOPSIS**

**C Syntax**

**void**
**pinq_list_avail_ws_types ( length, start, error_ind, types, length_list)**

| | | |
|---|---|---|
| **Pint** | **length;** | *length of application list* |
| **Pint** | **start;** | *starting position* |
| **Pint** | ∗**error_ind;** | *OUT error indicator* |
| **Pint_list** | ∗**types;** | *OUT list of ws types* |
| **Pint** | ∗**length_list;** | *OUT length of list in PHIGS* |

**Required PHIGS Operating States**

(PHOP, ∗, ∗, ∗)

**DESCRIPTION**

**Purpose**

Use INQUIRE LIST OF AVAILABLE WORKSTATION TYPES to obtain a list of the currently available workstation types stored in the PHIGS description table. This list contains the predefined workstation types described in OPEN WORKSTATION and user-created workstation types created with WORKSTATION TYPE CREATE. User-created types are removed from this list by WORKSTATION TYPE DESTROY.

**C Input Parameters**

*length* The number of *ints* in the *types* output parameter for which the application has allocated memory. *length* is the number of list elements that the system can return in *types→ints*. If a value of 0 is used here, no data will be returned in the *types→ints* list, but the total number of elements will be returned in *length_list*.

*start* Starting position of inquiry. The elements in the list, beginning with the item number specified by *start*, are copied sequentially into *types→ints* until *types→ints* is full or all the elements have been copied.

**C Output Parameters**

*error_ind*

A pointer to the location for storing the error number of any error detected by this function. See the *Errors* section below for possible return values.

*types* A pointer to a Pint_list structure in which to return the number and list of available workstation types. Pint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint    num_ints;       /* number of Pints in list */
        Pint    *ints;          /* list of Pints */
} Pint_list;
```

The *num_ints* component specifies the number of elements in the list. The *ints* component is a pointer to a list *num_ints* long.

The application must allocate memory for *length* elements in the list of *ints*.

*length_list*

A pointer to an integer in which the system returns the total number of elements in the list. This is the value required for *length* if all elements in the list are to be returned.

**Execution**

INQUIRE LIST OF AVAILABLE WORKSTATION TYPES returns the list of available workstation types stored in the PHIGS description table. See OPEN WORKSTATION and WORKSTATION TYPE CREATE for a description of the available workstation types and information on how to create new ones.

If this function detects an error, then the *error indicator* indicates the error number of the detected error and no other output data is returned, except in the cases mentioned in the C Parameters section above. If the function detects no error, then the *error indicator* is set to zero and the inquired information is available in the output parameters. Since this is an inquiry function, ERROR HANDLING is not invoked when this function detects an error.

**ERRORS**

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

2200 Buffer overflow in input or inquiry function

**SEE ALSO**

OPEN WORKSTATION (3P)
PHIGS WORKSTATION DESCRIPTION TABLE (7P)
WORKSTATION TYPE CREATE (3P)
WORKSTATION TYPE DESTROY (3P)
WORKSTATION TYPE SET (3P)

**NAME**

INQUIRE LIST OF COLOUR INDICES – inquire list of colour indices defined on a workstation

**SYNOPSIS**

**C Syntax**

**void**
**pinq_list_colr_inds ( ws, length, start, error_ind, indices, total_length )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **length;** | *length of application list* |
| **Pint** | **start;** | *starting position* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pint_list** | ***indices;** | *OUT list of colour indices* |
| **Pint** | ***total_length;** | *OUT length of list in PHIGS* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE LIST OF COLOUR INDICES to obtain a list of the currently defined colour indices on the specified workstation. The defined colour table indices are *conceptually sparse*: a PHIGS implementation could allow definition of *n* colours with arbitrary indices. However, colour indices are *nonsparse* in PEX-SI: the *n* colour representations necessarily have indices 0 through *n - 1*.

**C Input Parameters**

*ws* Workstation identifier of the workstation whose colour indices are to be returned.

*length* Number of items for which the application has allocated memory in the output parameter indices. Specify 0 to get the total length of the list.

*start* Starting position in the list at which to begin the inquiry.

**C Output Parameters**

*error_ind*

Pointer to the location for storing the error number of any error this function detects.

*indices* A pointer to a Pint_list in which the system returns the portion of the list of currently defined colour indices, starting at the entry specified with *start*. Pint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint    num_ints;    /* number of Pints in list */
        Pint    *ints;       /* list of integers */
} Pint_list;
```

The pointer *indices→ints* must be initialized to an array of *length* Pint elements.

*total_length*

Pointer to an integer in which to return the total length of the list. This is the value required for *length* if all the items in the list are to be returned.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, specified workstation is not open

059 Ignoring function, specified workstation does not have output capability (in other words, the workstation category is neither OUTPUT, OUTIN, nor MO)

**SEE ALSO**

SET COLOUR REPRESENTATION (3P)
INQUIRE PREDEFINED COLOUR REPRESENTATION (3P)

**NAME**

INQUIRE LIST OF COLOUR MAPPING INDICES – inquire the list of defined colour mapping indices on a workstation

**SYNOPSIS**

**C Syntax**

**void**
**pinq_list_colr_map_inds ( ws, length, start, error_ind, colr_map_inds, total_length )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **length;** | *length of application list* |
| **Pint** | **start;** | *starting position* |
| **Pint** | ∗**error_ind;** | *OUT error indicator* |
| **Pint_list** | ∗**colr_map_inds;** | *OUT list of polyline indices* |
| **Pint** | ∗**total_length;** | *OUT length of list in PHIGS* |

**Required PHIGS Operating States**

(PHOP, WSOP, ∗, ∗)

**DESCRIPTION**

**Purpose**

INQUIRE LIST OF COLOUR MAPPING INDICES obtains a list of the currently defined workstation colour mapping indices.

**C Input Parameters**

*ws* The workstation identifier of the workstation whose colour mapping indices are to be returned.

*length* The number of *ints* in the *colr_map_inds* output parameter for which the application has allocated memory. *length* is the number of list elements that the system can return in *colr_map_inds→ints*. If a value of 0 is used here, no data will be returned in the *colr_map_inds→ints* list, but the total number of elements will be returned in *total_length*.

*start* Starting position of inquiry. The elements in the list, beginning with the item number specified by *start*, are copied sequentially into *colr_map_inds→ints* until *colr_map_inds→ints* is full or all the elements have been copied.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*colr_map_inds*

A pointer to a Pint_list data structure containing the list of colour mapping indices. A Pint_list structure is defined as:

```
typedef struct {
        Pint    num_ints;    /* number of Pints in list */
        Pint    *ints;       /* list of integers */
} Pint_list;
```

The *num_ints* component specifies the number of elements in the list. The *ints* component is a pointer to a list *num_ints* long.

The application must allocate memory for *length* elements in the list of *ints*.

*total_length*

A pointer to an integer which returns the total length of the list. This is the value required for *length* if all items in the list are to be returned.

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the following error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

003 Ignoring function, function requires state (PHOP,WSOP,*,*)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT nor OUTIN)

**SEE ALSO**

SET COLOUR MAPPING REPRESENTATION (3P+)
SET COLOUR MAPPING INDEX (3P+)
INQUIRE COLOUR MAPPING REPRESENTATION (3P+)

**NAME**

INQUIRE LIST OF DEPTH CUE INDICES – inquire the list of depth cue indices defined on a workstation

**SYNOPSIS**

**C Syntax**

**void**
**pinq_list_dcue_inds ( ws, length, start, error_ind, indices, total_length )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **length;** | *length of application list* |
| **Pint** | **start;** | *starting position* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pint_list** | ***indices;** | *OUT list of polyline indices* |
| **Pint** | ***total_length;** | *OUT length of list in PHIGS* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

INQUIRE LIST OF DEPTH CUE INDICES obtains a list of the currently defined workstation depth cue indices. Objects are depth cued toward a specified depth cue colour. Depth cueing applies to all primitives.

**C Input Parameters**

*ws* The workstation identifier.

*length* The number of *ints* in the *indices* output parameter for which the application has allocated memory. *length* is the number of list elements that the system can return in *indices→ints*. If a value of 0 is used here, no data will be returned in the *indices→ints* list, but the total number of elements will be returned in *total_length*.

*start* Starting position of inquiry. The elements in the list, beginning with the item number specified by *start*, are copied sequentially into *indices→ints* until *indices→ints* is full or all the elements have been copied.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*indices* A pointer to a Pint_list data structure containing the list of depth cue indices. A Pint_list structure is defined as:

```
typedef struct {
        Pint    num_ints;    /* number of Pints in list */
        Pint    *ints;       /* list of integers */
} Pint_list;
```

The *num_ints* component specifies the number of elements in the list. The *ints* component is a pointer to a list *num_ints* long.

The application must allocate memory for *length* elements in the list of *ints*.

*total_length*

A pointer to an integer in which the system returns the total number of elements in the list. This is the value required for *length* if all elements in the list are to be returned.

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the following error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

003 Ignoring function, function requires state (PHOP,WSOP,*,*)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT, OUTIN, nor MO)

**SEE ALSO**

SET DEPTH CUE REPRESENTATION (3P+)
SET DEPTH CUE INDEX (3P+)
INQUIRE DEPTH CUE REPRESENTATION (3P+)

**NAME**

INQUIRE LIST OF EDGE INDICES– inquire list of edge indices defined on a workstation

**SYNOPSIS**

**C Syntax**

**void**
**pinq_list_edge_inds ( ws, length, start, error_ind, indices, total_length )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **length;** | *length of application list* |
| **Pint** | **start;** | *starting position* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pint_list** | ***indices;** | *OUT list of edge indices* |
| **Pint** | ***total_length;** | *OUT length of list in PHIGS* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE LIST OF EDGE INDICES to obtain a list of the currently defined edge indices on the specified workstation. The defined edge table indices are *conceptually sparse*: a PHIGS implementation could allow definition of *n* edge representations with arbitrary indices. However, edge indices are *nonsparse* in PEX-SI: the *n* edge representations necessarily have indices 1 through *n*.

**C Input Parameters**

*ws* *Workstation identifier* of the workstation whose edge indices are to be returned.

*length* Number of items for which the application has allocated memory in the output parameter *indices*. Specify 0 to get the *total length* of the list.

*start* Starting position in the list at which to begin the inquiry.

**C Output Parameters**

*error_ind*

Pointer to the location for storing the error number of any error this function detects.

*indices* Pointer to a Pint_list structure in which the system returns the portion of the list of currently defined edge indices, starting at the entry specified with *start*.

Pint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint    num_ints;    /* number of Pints in list */
        Pint    *ints;       /* list of integers */
} Pint_list;
```

Pointer *indices→ints* must be initialized to an array of *length Pint* elements.

*total_length*

Pointer to an integer in which to return the total length of the list. This is the value required for *length* if all the items in the list are to be returned.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, specified workstation is not open

059 Ignoring function, specified workstation does not have output capability (in other words, the workstation category is neither OUTPUT, OUTIN, nor MO)

**SEE ALSO**

SET EDGE REPRESENTATION (3P)
INQUIRE EDGE REPRESENTATION (3P)

**NAME**

INQUIRE LIST OF INTERIOR INDICES– inquire list of interior indices defined on a workstation

**SYNOPSIS**

**C Syntax**

**void**
**pinq_list_int_inds ( ws, length, start, error_ind, indices, total_length )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **length;** | *length of application list* |
| **Pint** | **start;** | *starting position* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pint_list** | ***indices;** | *OUT list of interior indices* |
| **Pint** | ***total_length;** | *OUT length of list in PHIGS* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE LIST OF INTERIOR INDICES to obtain a list of the currently defined interior indices on the specified workstation. The defined interior table indices are *conceptually sparse*: a PHIGS implementation could allow definition of *n* interior representations with arbitrary indices. However, interior indices are *non-sparse* in PEX-SI; the *n* interior representations necessarily have indices 1 through *n*.

**C Input Parameters**

*ws* The *workstation identifier* of the workstation whose interior indices are to be returned.

*length* Number of items for which the application has allocated memory in the output parameter *indices*. 0 can be specified, in order to get the *total length* of the list.

*start* Starting position in the list at which to begin the inquiry.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*indices* A pointer to a Pint_list structure in which the system returns the portion of the list of currently defined interior indices, starting at the entry specified with *start*.

Pint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint    num_ints;    /* number of Pints in list */
        Pint    *ints;       /* list of integers */
} Pint_list;
```

The pointer *indices->ints* must be initialized to an array of *length Pint* elements.

*total_length*

A pointer to an integer in which to return the *total length* of the list. This is the value required for *length* when all the items in the list are to be returned.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (in other words, the workstation category is neither OUTPUT, OUTIN, nor MO)

**SEE ALSO**

SET INTERIOR REPRESENTATION (3P)
SET INTERIOR INDEX (3P)

INQUIRE INTERIOR REPRESENTATION (3P)

**NAME**

INQUIRE LIST OF LIGHT SOURCE INDICES – inquire the list of light source indices defined on a workstation

**SYNOPSIS**

**C Syntax**

**void**
**pinq_list_light_src_inds ( ws, length, start, error_ind, indices, total_length )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **length;** | *length of application list* |
| **Pint** | **start;** | *starting position* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pint_list** | ***indices;** | *OUT list of light source indices* |
| **Pint** | ***total_length;** | *OUT length of list in PHIGS* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

INQUIRE LIST OF LIGHT SOURCE INDICES inquires a list of light source indices defined on a workstation.

**C Input Parameters**

*ws* The workstation identifier.

*length* The number of *ints* in the *indices* output parameter for which the application has allocated memory. *length* is the number of list elements that the system can return in *indices→ints*. If a value of 0 is used here, no data will be returned in the *indices→ints* list, but the total number of elements will be returned in *total_length*.

*start* Starting position of inquiry. The elements in the list, beginning with the item number specified by *start*, are copied sequentially into *indices→ints* until *indices→ints* is full or all the elements have been copied.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*indices* This is a pointer to a Pint_list structure. This structure specifies both an integer value indicating the number of light source representations and a pointer to a list specifying the index numbers of the light source representations. Pint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint    num_ints;    /* number of light sources */
        Pint    *ints;       /* list of light source indices */
} Pint_list;
```

The *num_ints* component specifies the number of elements in the list. The *ints* component is a pointer to a list *num_ints* long.

The application must allocate memory for *length* elements in the list of *ints*.

*total_length*

A pointer to an integer in which the system returns the total number of elements in the list of light source indices. This is the value required for *length* if all elements in the list are to be returned.

**Execution**

If the inquired information is available, the error indicator is returned as zero, and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the following error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

003 Ignoring function, function requires state (PHOP,WSOP,*,*)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT, OUTIN, nor MO)

**SEE ALSO**

INQUIRE LIGHT SOURCE REPRESENTATION (3P+)
SET LIGHT SOURCE REPRESENTATION (3P+)

**NAME**

INQUIRE LIST OF PATTERN INDICES – inquire list of pattern indices defined on a workstation

**SYNOPSIS**

**C Syntax**

```
void
pinq_list_pat_inds ( ws, length, start, error_ind, indices, total_length )
Pint        ws;             workstation identifier
Pint        length;         length of application list
Pint        start;          starting position
Pint        *error_ind;     OUT error indicator
Pint_list   *indices;       OUT list of pattern indices
Pint        *total_length;  OUT length of list in PHIGS
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE LIST OF PATTERN INDICES to obtain a list of the currently defined pattern indices on the specified workstation. The defined pattern table indices are *conceptually sparse*: a PHIGS implementation could allow definition of *n* pattern representations with arbitrary indices. However, pattern indices are *non-sparse* in PEX-SI; the *n* pattern representations necessarily have indices 1 through *n*.

**C Input Parameters**

*ws* The *workstation identifier* of the workstation whose pattern indices are to be returned.

*length* Number of items for which the application has allocated memory in the output parameter *indices*. 0 can be specified, in order to get the *total length* of the list.

*start* Starting position in the list at which to begin the inquiry.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*indices* A pointer to a Pint_list structure in which the system returns the portion of the list of currently defined pattern indices, starting at the entry specified with *start*.

Pint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint    num_ints;   /* number of Pints in list */
        Pint    *ints;      /* list of integers */
} Pint_list;
```

The pointer *indices→ints* must be initialized to an array of *length Pint* elements.

*total_length*

A pointer to an integer in which to return the *total length* of the list. This is the value required for *length* if all the items in the list are to be returned.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (in other words, the workstation category is neither OUTPUT, OUTIN, nor MO)

**SEE ALSO**

SET PATTERN REPRESENTATION (3P)
INQUIRE PATTERN REPRESENTATION (3P)

INQUIRE PREDEFINED PATTERN REPRESENTATION (3P)

**NAME**

INQUIRE LIST OF POLYLINE INDICES– inquire list of polyline indices defined on a workstation

**SYNOPSIS**

**C Syntax**

**void**
**pinq_list_line_inds ( ws, length, start, error_ind, indices, total_length )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **length;** | *length of application list* |
| **Pint** | **start;** | *starting position* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pint_list** | ***indices;** | *OUT list of polyline indices* |
| **Pint** | ***total_length;** | *OUT length of list in PHIGS* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE LIST OF POLYLINE INDICES to obtain a list of the currently defined polyline indices on the specified workstation. The defined polyline table indices are *conceptually sparse*; a PHIGS implementation could allow definition of *n* polyline representations with arbitrary indices. However, polyline indices are *nonsparse* in PEX-SI; the *n* polyline representations necessarily have indices 1 through *n*.

**C Input Parameters**

*ws* The *workstation identifier* of the workstation whose polyline indices are to be returned.

*length* Number of items for which the application has allocated memory in the output parameter *indices*. 0 can be specified, in order to get the *total length* of the list.

*start* Starting position in the list at which to begin the inquiry.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*indices* A pointer to a Pint_list structure in which the system returns the portion of the list of currently defined polyline indices, starting at the entry specified with *start*.

Pint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint    num_ints;    /* number of Pints in list */
        Pint    *ints;       /* list of integers */
} Pint_list;
```

The pointer *indices→ints* must be initialized to an array of *length Pint* elements.

*total_length*

A pointer to an integer in which to return the *total length* of the list. This is the value required for *length* if all the items in the list are to be returned.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (in other words, the workstation category is neither OUTPUT, OUTIN, nor MO)

**SEE ALSO**

SET POLYLINE REPRESENTATION (3P)
INQUIRE POLYLINE REPRESENTATION (3P)

INQUIRE PREDEFINED POLYLINE REPRESENTATION (3P)

**NAME**

INQUIRE LIST OF POLYMARKER INDICES– inquire list of polymarker indices defined on a workstation

**SYNOPSIS**

**C Syntax**

**void**
**pinq_list_marker_inds ( ws, length, start, error_ind, indices, total_length )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **length;** | *length of application list* |
| **Pint** | **start;** | *starting position* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pint_list** | ***indices;** | *OUT list of polymarker indices* |
| **Pint** | ***total_length;** | *OUT length of list in PHIGS* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE LIST OF POLYMARKER INDICES to obtain a list of the currently defined polymarker indices on the specified workstation. The defined polymarker table indices are *conceptually sparse*; a PHIGS implementation could allow definition of *n* polymarker representations with arbitrary indices. However, polymarker indices are *nonsparse* in PEX-SI; the *n* polymarker representations necessarily have indices 1 through *n*.

**C Input Parameters**

*ws* The *workstation identifier* of the workstation whose polymarker indices are to be returned.

*length* Number of items for which the application has allocated memory in the output parameter *indices*. 0 can be specified, in order to get the total length of the list.

*start* Starting position in the list at which to begin the inquiry.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*indices* A pointer to a Pint_list structure in which the system returns the portion of the list of currently defined polymarker indices, starting at the entry specified with *start*.

Pint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint   num_ints;    /* number of Pints in list */
        Pint   *ints;       /* list of integers */
} Pint_list;
```

The pointer *indices→ints* must be initialized to an array of *length Pint* elements.

*total_length*

A pointer to an integer in which to return the total length of the list. This is the value required for *length* if all the items in the list are to be returned.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (in other words, the workstation category is neither OUTPUT, OUTIN, nor MO)

**SEE ALSO**

SET POLYMARKER REPRESENTATION (3P)
INQUIRE POLYMARKER REPRESENTATION (3P)
INQUIRE PREDEFINED POLYMARKER REPRESENTATION (3P)

**NAME**

INQUIRE LIST OF TEXT INDICES– inquire list of text indices defined on a workstation

**SYNOPSIS**

**C Syntax**

**void**
**pinq_list_text_inds ( ws, length, start, error_ind, indices, total_length )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **length;** | *length of application list* |
| **Pint** | **start;** | *starting position* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pint_list** | ***indices;** | *OUT list of text indices* |
| **Pint** | ***total_length;** | *OUT length of list in PHIGS* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE LIST OF TEXT INDICES to obtain a list of the currently defined text indices on the specified workstation. The defined text table indices are *conceptually sparse*; a PHIGS implementation could allow definition of *n* text representations with arbitrary indices. However, text indices are *nonsparse* in PEX-SI; the *n* text representations necessarily have indices 1 through *n*.

**C Input Parameters**

*ws* The *workstation identifier* of the workstation whose text indices are to be returned.

*length* Number of items for which the application has allocated memory in the output parameter *indices*. 0 can be specified, in order to get the *total length* of the list.

*start* Starting position in the list at which to begin the inquiry.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*indices* A pointer to a Pint_list structure in which the system returns the portion of the list of currently defined text indices, starting at the entry specified with *start*.

Pint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint   num_ints;    /* number of Pints in list */
        Pint   *ints;       /* list of integers */
} Pint_list;
```

The pointer *indices→ints* must be initialized to an array of *length Pint* elements.

*total_length*

A pointer to an integer in which to return the *total length* of the list. This is the value required for *length* if all the items in the list are to be returned.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (in other words, the workstation category is neither OUTPUT, OUTIN, nor MO)

**SEE ALSO**

SET TEXT REPRESENTATION (3P)
INQUIRE TEXT REPRESENTATION (3P)

INQUIRE PREDEFINED TEXT REPRESENTATION (3P)

**NAME**

INQUIRE LIST OF VIEW INDICES– inquire list of view indices defined on a workstation

**SYNOPSIS**

**C Syntax**

```
void
pinq_list_view_inds ( ws, length, start, error_ind, indices, total_length )
Pint        ws;          workstation identifier
Pint        length;      length of application list
Pint        start;       starting position
Pint        *error_ind;  OUT error indicator
Pint_list   *indices;    OUT list of view indices
Pint        *total_length; OUT length of list in PHIGS
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE LIST OF VIEW INDICES to obtain a list of the currently defined view indices on the specified workstation. The list is in order of decreasing view transformation input priority.

The defined view table indices are *conceptually sparse*; a PHIGS implementation could allow definition of *n* view representations with arbitrary indices. However, view indices are *nonsparse* in PEX-SI; the *n* settable view representations necessarily have indices 1 through *n*.

**C Input Parameters**

*ws* The *workstation identifier* of the workstation whose view indices are to be returned.

*length* Number of items for which the application has allocated memory in the output parameter *indices*. 0 can be specified, in order to get the *total length* of the list.

*start* Starting position in the ordered list at which to begin the inquiry.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*indices* A pointer to a Pint_list structure in which the system returns the portion of the list of currently defined view indices, starting at the entry specified with *start*.

Pint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint   num_ints;   /* number of Pints in list */
        Pint   *ints;      /* list of integers */
} Pint_list;
```

The pointer *indices→ints* must be initialized to an array of *length Pint* elements.

*total_length*

A pointer to an integer in which to return the *total length* of the list. This is the value required for *length* if all the items in the list are to be returned.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

057 Ignoring function, specified workstation is of category MI

**SEE ALSO**

SET VIEW REPRESENTATION (3P)
INQUIRE VIEW REPRESENTATION (3P)
INQUIRE PREDEFINED VIEW REPRESENTATION (3P)

**NAME**

INQUIRE LOCATOR DEVICE STATE – inquire state of a locator device

**SYNOPSIS**

**C Syntax**

**void**
**pinq_loc_st ( ws, dev, type, store, err, op_mode, echo_switch, init_view_ind, init_loc_pos, prompt_echo, echo_area, loc_data)**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **dev;** | *locator device number* |
| **Pinq_type** | **type;** | *type of returned value* |
| **Pstore** | **store;** | *pointer to buffer* |
| **Pint** | ***err;** | *OUT error indicator* |
| **Pop_mode** | ***op_mode;** | *OUT operating mode* |
| **Pecho_switch** | ***echo_switch;** | *OUT echo switch* |
| **Pint** | ***init_view_ind;** | *OUT initial view index* |
| **Ppoint** | ***init_loc_pos;** | *OUT initial locator position* |
| **Pint** | ***prompt_echo;** | *OUT prompt/echo type* |
| **Plimit** | ***echo_area;** | *OUT echo area* |
| **Ploc_data** | ****loc_data;** | *OUT data record* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE LOCATOR DEVICE STATE to determine the current state of the specified locator device.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing data associated with the device state. This buffer is passed as the *store* argument.

The store buffer is a data area managed by PHIGS API. While the application is responsible for creating the inital buffer through a call to STORE, the implementation manages this area such that there is sufficient memory for the specific inquiry. The locator device data record within the store buffer is accessed by the pointer pointed to by *loc_data*.

*ws* Workstation identifier. An integer specifying the workstation with which the specified locator device is associated.

*dev* The device number of the locator device. See the *Available Devices* section of INITIALIZE LOCATOR for a description of the available devices.

*type* An enumerated value specifying whether the values to be returned are those originally specified by the application (SET), or those resulting after PHIGS mapped them to ones available on the workstation (REALIZED). A Pinq_type structure is defined as:

```
typedef enum {
        PINQ_SET,
        PINQ_REALIZED
} Pinq_type;
```

*store* The memory buffer PHIGS is to use for storing the information returned. This buffer must exist prior to calling this function (see CREATE STORE (3P)).

**C Output Parameters**

*err* The error indicator. See the *Execution* section below for a description of its use. See the *Errors* section below for the possible values it may return.

*op_mode*

The operating mode. Pop_mode is an enumerated type with the following values:

POP_REQ
POP_SAMPLE
POP_EVENT

*echo_switch*

The echo state. Pecho_switch is an enumerated type with the following values:

PSWITCH_ECHO
PSWITCH_NO_ECHO

*init_view_ind*

Initial view index.

*init_loc_pos*

Intitial locator position.

*prompt_echo*

The prompt/echo type desired. See the *Available Devices* Section of INITIALIZE LOCATOR for a description of the available types.

*echo_area*

A pointer to a variable of type Plimit that contains the echo area of the device. Plimit is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          x_min;          /* x min */
        Pfloat          x_max;          /* x max */
        Pfloat          y_min;          /* y min */
        Pfloat          y_max;          /* y max */
} Plimit;
```

*loc_data*

Pointer to a pointer to the locator device record within the *store* data buffer.

**Execution**

INQUIRE LOCATOR DEVICE STATE returns the current state of the specified locator device, which is stored in the workstation state list of the workstation associated with the device. The current state includes the operating mode, echo switch, initial locator value, prompt/echo type, echo area, and data record. See SET LOCATOR MODE for a description of the operating mode and the echo switch and how to set these values. See INITIALIZE LOCATOR for a description of the initial locator value, prompt/echo type, echo area, and data record contents and how to set these values.

In the event of an error, the *error indicator* indicates the error number of the error detected, and no other output data is returned. If no error is detected, the *error indicator* is set to zero, and the inquired information will be available in the output parameters. Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

**SEE ALSO**

INITIALIZE LOCATOR (3P)
SET LOCATOR MODE (3P)
INQUIRE LOCATOR DEVICE STATE 3 (3P)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

INQUIRE LOCATOR DEVICE STATE 3 – inquire state of a locator device

**SYNOPSIS**

**C Syntax**

**void**
**pinq_loc_st3 ( ws, dev, type, store, err, op_mode, echo_switch, init_view_ind, init_loc_pos, prompt_echo, echo_vol, loc_data)**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **dev;** | *locator device number* |
| **Pinq_type** | **type;** | *type of returned value* |
| **Pstore** | **store;** | *store handle* |
| **Pint** | ***err;** | *OUT error indicator* |
| **Pop_mode** | ***op_mode;** | *OUT operating mode* |
| **Pecho_switch** | ***echo_switch;** | *OUT echo switch* |
| **Pint** | ***init_view_ind;** | *OUT initial view index* |
| **Ppoint3** | ***init_loc_pos;** | *OUT initial locator position* |
| **Pint** | ***prompt_echo;** | *OUT prompt/echo type* |
| **Plimit3** | ***echo_vol;** | *OUT echo volume* |
| **Ploc_data3** | ****loc_data;** | *OUT data record* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE LOCATOR DEVICE STATE 3 to retrieve the current state of a locator device.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing data associated with the device state. This buffer is passed as the *store* argument.

The store buffer is a data area managed by PHIGS. While the application is responsible for creating the inital buffer through a call to STORE, PHIGS manages this area such that there is sufficient memory for the specific inquiry. The locator device data record within the store buffer is accessed by the pointer pointed to by loc_data.

*ws* Workstation identifier. An integer specifying the workstation with which the specified locator device is associated.

*dev* The device number of the locator device. See the *Available Devices* section of INITIALIZE LOCATOR for a description of the available devices.

*type* An enumerated value specifying whether the values to be returned are those originally specified by the application (PINQ_SET), or those resulting after PHIGS mapped them to ones available on the workstation (PINQ_REALIZED). A Pinq_type structure is defined as:

```
typedef enum {
        PINQ_SET,
        PINQ_REALIZED
} Pinq_type;
```

*store* The memory buffer PHIGS is to use for storing the information returned. This buffer must exist prior to calling this function (see CREATE STORE (3P)).

**C Output Parameters**

*err* The error indicator. See the *Execution* section below for a description of its use. See the *Errors* section below for the possible values it may return.

*op_mode*

A pointer to a variable of type Pop_mode, which contains the current operating mode of the device. Pop_mode is enumerated in phigs.h as follows:

POP_REQ
POP_SAMPLE
POP_EVENT

*echo_switch*

A pointer to a variable of type Pecho_switch, which contains the state of the devices echo switch. The value returned for *echo_switch* will be one of the following:

PSWITCH_ECHO
PSWITCH_NO_ECHO

*init_view_ind*

Initial view index.

*init_loc_pos*

Initial locator position.

*prompt_echo*

The prompt/echo type desired. See the *Available Devices* Section of INITIALIZE LOCATOR for a description of the available types.

*echo_vol*

A pointer to a Plimit3 structure defining the *x*, *y*, and *z* components of the echo volume, in Device Coordinates. Plimit3 is defined in phigs.h as follows:

```
typedef struct {
        Pfloat      x_min;      /* minimum x coordinate value */
        Pfloat      x_max;      /* maximum x coordinate value */
        Pfloat      y_min;      /* minimum y coordinate value */
        Pfloat      y_max;      /* maximum y coordinate value */
        Pfloat      z_min;      /* minimum z coordinate value */
        Pfloat      z_max;      /* maximum z coordinate value */
} Plimit3;
```

*loc_data*

Points to a pointer to the locator device data within the store buffer.

**Execution**

INQUIRE LOCATOR DEVICE STATE 3 returns the current state of the specified locator device, which is stored in the workstation state list of the workstation associated with the device. The current state includes the operating mode, echo switch, initial locator value, prompt/echo type, echo volume, and data record. See SET LOCATOR MODE for a description of the operating mode and the echo switch, and how to set these values. See INITIALIZE LOCATOR 3 for a description of the initial locator value, prompt/echo type, echo volume, and data record contents, and how to set these values.

In the event of an error, the *error indicator* indicates the error number of the error detected, and no other output data is returned. If no error is detected, the *error indicator* is set to zero, and the inquired information will be available in the output parameters. Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

**SEE ALSO**

INITIALIZE LOCATOR 3 (3P)
SET LOCATOR MODE (3P)
INQUIRE LOCATOR DEVICE STATE (3P)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

INQUIRE MODELLING CLIPPING FACILITIES – inquire number of modelling clipping planes and operators available

**SYNOPSIS**

**C Syntax**

**void**
**pinq_model_clip_facs ( length, start, error_ind, num, ops, total_length )**

| | | |
|---|---|---|
| **Pint** | **length;** | *length of application list* |
| **Pint** | **start;** | *starting position* |
| **Pint** | ∗**error_ind;** | *OUT error indicator* |
| **Pint** | ∗**num;** | *OUT number of distinct planes in modelling clipping volume* |
| **Pint_list** | ∗**ops;** | *OUT list of operators* |
| **Pint** | ∗**total_length;** | *OUT length of list in PHIGS* |

**Required PHIGS Operating States**

(PHOP, ∗, ∗, ∗)

**DESCRIPTION**

**Purpose**

INQUIRE MODELLING CLIPPING FACILITIES obtains the number of planes and available operators that may be used to determine the modelling clipping volume in the PHIGS implementation.

**C Input Parameters**

*length* Number of items for which the application has allocated memory in the output parameter *ops*. 0 can be specified, in order to get the *total length* of the list of operators.

*start* Starting position in the list of operators at which to begin the inquiry.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*num* A pointer to a location in which the system return the number of distinct planes (half-spaces) that can be used to determine the modelling clipping volume.

*ops* A pointer to a *Pint_list* in which the system returns the portion of the list of supported modelling clipping operators, starting at the entry specified with *start*.

Pint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint   num_ints;    /* number of Pints in list */
        Pint   *ints;       /* list of integers */
} Pint_list;
```

The pointer *ops→ints* must be initialized to an array of *length Pint* elements.

*total_length*

A pointer to an integer in which to return the *total length* of the list of supported operators. This is the value required for *length* if all the items in the list are to be returned.

**ERRORS**

002 Ignoring function, function requires state (PHOP, ∗, ∗, ∗)

**SEE ALSO**

PHIGS DESCRIPTION TABLE (7P)
SET MODELLING CLIPPING INDICATOR (3P)
SET MODELLING CLIPPING VOLUME (3P)

**NAME**

INQUIRE MORE SIMULTANEOUS EVENTS – see if there are more simultaneous events on the input queue

**SYNOPSIS**

**C Syntax**

```
void
pinq_more_simult_events ( error_ind, events )
Pint                *error_ind;          OUT error indicator
Pmore_simult_events *events;             OUT simultaneous events
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

Use INQUIRE MORE SIMULTANEOUS EVENTS to determine if there are more simultaneous events on the input queue. Simultaneous events are generated when two or more input devices in EVENT mode share a trigger and that trigger is fired. One event for each device sharing the trigger is generated in such a case, and the events are marked as simultaneous.

**C Input Parameters**

*error_ind*

A pointer to the error indicator. If an error is detected, the number of the error is stored in this variable and the contents of *events* is undefined.

*events* A pointer to a Pmore_simult_events type defined in phigs.h as:

```
typedef enum {
        PSIMULT_NO_MORE,
        PSIMULT_MORE
} Pmore_simult_events;
```

**Execution**

INQUIRE MORE SIMULTANEOUS EVENTS examines the *more simultaneous events* field in the PHIGS state list to see if the next event is the next in a group of simultaneous events. When the first event in a group of simultaneous events reaches the front of the input event queue and is removed by AWAIT EVENT, the *more simultaneous events* field is set to MORE. When the last event in the group is removed from the event queue the value is set to NO MORE.

Simultaneous events are generated by input devices that share the same trigger. If these input devices are in EVENT mode and the trigger is fired, then an input event will be enqueued for each of the devices and each will be marked as one in a group of simultaneous events. The order in which they are enqueued is undefined. If there is not enough space on the input queue to hold this entire group of events, then none of them will be enqueued and one of them will be placed in the *identification of one of the logical input devices that caused an overflow* field in the PHIGS error state list.

PEX-SI has many devices that can potentially share triggers. See the corresponding INITIALIZE *input class* functions for a description of the triggers each device uses.

**ERRORS**

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

**SEE ALSO**

AWAIT EVENT (3P)
GET CHOICE (3P)
GET LOCATOR (3P)
GET PICK (3P)
GET STRING (3P)
GET STROKE (3P)
GET VALUATOR (3P)

**NAME**

INQUIRE NUMBER OF AVAILABLE LOGICAL INPUT DEVICES – inquire the number of available logical input devices for a specified workstation type

**SYNOPSIS**

**C Syntax**

```
void
pinq_num_avail_in ( type, error_ind, numbers )
Pint        type;          workstation type
Pint        *error_ind;    OUT error indicator
Pnum_in     *numbers;      OUT number of input devices
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE NUMBER OF AVAILABLE LOGICAL INPUT DEVICES to determine the number of available logical input devices for a specified workstation type.

**C Input Parameters**

*type* Type of workstation. See OPEN WORKSTATION for a list of those available.

*error_ind*

The error indicator. See the *Execution* section below for a description of its use. See the *Errors* section below for the possible values it may return.

*numbers*

A pointer to a Pnum_in data structure in which to return the number of devices available. Pnum_in is defined in phigs.h as follows:

```
typedef struct {
        Pint    loc;        /* locators */
        Pint    stroke;     /* strokes */
        Pint    val;        /* valuators */
        Pint    choice;     /* choices */
        Pint    pick;       /* picks */
        Pint    string;     /* strings */
} Pnum_in;
```

**Execution**

INQUIRE NUMBER OF AVAILABLE LOGICAL INPUT DEVICES returns the number of input devices available of each input class for a given workstation type. If no errors are detected the numbers are returned. If an error is detected, the *error indicator* will be set to one of the values specified in the *Errors* section below and the input device information will not be returned.

**ERRORS**

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

052 Ignoring function, workstation type not recognized by the implementation

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

**SEE ALSO**

INQUIRE WORKSTATION CONNECTION AND TYPE (3P)
INITIALIZE CHOICE (3P)
INITIALIZE LOCATOR (3P)

INITIALIZE PICK (3P)
INITIALIZE STRING (3P)
INITIALIZE STROKE (3P)
INITIALIZE VALUATOR (3P)

**NAME**

INQUIRE NUMBER OF DISPLAY PRIORITIES SUPPORTED – inquire number of display priorities supported by a workstation type

**SYNOPSIS**

**C Syntax**

```
void
pinq_num_disp_pris ( wst, error_ind, num_pri )
Pint        wst;        workstation type
Pint        *error_ind; OUT error indicator
Pint        *num_pri;   OUT number of display priorities
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE NUMBER OF DISPLAY PRIORITIES SUPPORTED to determine the number of display priorities supported by a specified workstation type.

If the number of priorities returned is zero, this means the workstation type can support an arbitrary number of priorities.

**C Input Parameters**

*wst* Workstation type.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*num_pri*

A pointer to an integer that stores the number of display priorities supported on workstations of type *wst*.

**ERRORS**

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

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

052 Ignoring function, workstation type not recognized by the implementation

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

**SEE ALSO**

POST STRUCTURE (3P)

**NAME**

INQUIRE OPEN STRUCTURE – inquire the status of the currently opened structure

**SYNOPSIS**

**C Syntax**

```
void
pinq_open_struct ( error_ind, status, struct_id )
Pint                *error_ind;          OUT error indicator
Popen_struct_status *status;             OUT status of open structure
Pint                *struct_id;          OUT structure identifier
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE OPEN STRUCTURE to determine the status of the currently open structure.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*status* A pointer to a Popen_struct_status enumerated type in which the system returns the status of the currently open structure. Values of Popen_struct_status are defined in phigs.h as follows:

| | |
|---|---|
| PSTRUCT_NONE | *No Structure Is Open* |
| PSTRUCT_OPEN | *A Structure Is Open* |

*struct_id*

An integer pointer in which the system returns the structure identifier of the currently open structure. This value is undefined if *status* is PSTRUCT_NONE.

**ERRORS**

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

**SEE ALSO**

OPEN STRUCTURE (3P)
CLOSE STRUCTURE (3P)
INQUIRE STRUCTURE STATUS (3P)
INQUIRE STRUCTURE STATE VALUE (3P)

**NAME**

INQUIRE PATHS TO ANCESTORS – inquire the ancestors of the specified structure

**SYNOPSIS**

**C Syntax**

**void**
**pinq_paths_ances ( struct_id, order, depth, store, error_ind, paths )**

| | | |
|---|---|---|
| **Pint** | **struct_id;** | *structure identifier* |
| **Ppath_order** | **order;** | *path order* |
| **Pint** | **depth;** | *path depth* |
| **Pstore** | **store;** | *handle to Store object* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pelem_ref_list_list** | ****paths;** | *OUT structure path list* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

INQUIRE PATHS TO ANCESTORS determines the path or paths in the Central Structure Store that reference the specified structure. It performs the same function in PEX-SI as INQUIRE PATHS TO ANCESTORS does in standard PHIGS.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing data associated with the device state. This buffer is passed as the *store* argument.

The store buffer is a data area managed by PHIGS. While the application is responsible for creating the inital buffer through a call to STORE, PHIGS manages this area such that there is sufficient memory for the specific inquiry. The data record within the store buffer is accessed by the pointer pointed to by *paths*.

*struct_id*
Identifier of the structure to be inquired for ancestors.

*order* Order in which the path lists are to be returned. This is an enumerated type defined in phigs.h to have the following values:

| | |
|---|---|
| PORDER_TOP_FIRST | *top first* |
| PORDER_BOTTOM_FIRST | *bottom first* |

*depth* Depth (maximum number of references) of path lists to return.

*store* The memory buffer PHIGS is to use for storing the information returned. This buffer must exist prior to calling this function (see CREATE STORE (3P)).

**C Output Parameters**

*error_ind*
A pointer to the location to store the error number of any error detected by this function.

*paths* A pointer to a pointer to a Pelem_ref_list_list structure which contains a list of the specified structure's structure path lists. Pelem_ref_list_list is defined in phigs.h as follows:

```
typedef struct{
        Pint            num_elem_ref_lists;     /* number of execute reference lists */
        Pelem_ref_list  *ex_ref_list;           /* list of execute reference lists */
} Pelem_ref_list_list;
```

The *num_elem_ref_lists* component specifies the number of structure path lists, or execute reference lists. The *ex_ref_list* component is a pointer to a list, *num_elem_ref_lists* long, of Pelem_ref_list structures, containing the specified structure's execute reference lists.

Pelem_ref_list is defined in phigs.h as follows:

```
typedef struct{
        Pint        num_elem_refs;   /* number of execute references */
        Pelem_ref   *elem_refs;      /* list of execute references */
} Pelem_ref_list;
```

*elem_refs* is a pointer to a list *num_elem_refs* long of Pelem_ref structures containing the structure identifier and element number of each execute reference structure element in the execute reference list. Pelem_ref is defined in phigs.h as follows:

```
typedef struct {
        Pint    struct_id;    /* structure identifier */
        Pint    elem_pos;     /* element number */
} Pelem_ref;
```

**Execution**

When INQUIRE PATHS TO ANCESTORSS is called, *paths* is filled with list(s) identifying the EXECUTE STRUCTURE structure elements which refer to *struct_id*, in the order of traversal. These EXECUTE STRUCTURE structure elements are represented as (structure identifier, element position) pairs, giving the parent structure identifier and the position of the EXECUTE STRUCTURE structure element. Whenever *struct_id* itself is included at the bottom of a returned path of ancestors, it is represented by a (structure identifier, element position) pair with an element position of 0. *order* and *depth* are used to determine the portion of each path returned. The number of references returned in each path is specified by *depth*; a *depth* of 0 returns all the references in the path. In case of truncation, *order* determines whether the head (PORDER_TOP_FIRST) or the tail (PORDER_BOTTOM_FIRST) portion of a path is returned. If a path truncation results in two or more partial paths with the same set of element references, only one of the identical path portions is returned.

For example, specifying PORDER_TOP_FIRST and a depth of 0 returns all paths to *struct_id*. Specifying PORDER_TOP_FIRST and a depth of 1 returns the root of each structure network that references *struct_id*. Specifying PORDER_BOTTOM_FIRST and a depth of 2 returns all the parents of *struct_id*.

**ERRORS**

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

201 Ignoring function, the specified structure does not exist

207 Ignoring function, the specified path depth is less than zero

**SEE ALSO**

EXECUTE STRUCTURE (3P)
INQUIRE PATHS DESCENDENTS (3P)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

INQUIRE PATHS TO DESCENDANTS – inquire the descendants of the specified structure

**SYNOPSIS**

**C Syntax**

**void**
**pinq_paths_descs ( struct_id, order, depth, store, error_ind, paths )**

| | | |
|---|---|---|
| **Pint** | **struct_id;** | *structure identifier* |
| **Ppath_order** | **order;** | *path order* |
| **Pint** | **depth;** | *path depth* |
| **Pstore** | **store;** | *handle to Store object* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pelem_ref_list_list** | ****paths;** | *OUT structure path list* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE PATHS DESCENDENTS to determine the path or paths in the Central Structure Store that are referenced by the specified structure. It performs the same function in PEX-SI as INQUIRE PATHS TO DESCENDANTS does in standard PHIGS.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing data associated with the device state. This buffer is passed as the *store* argument.

The store buffer is a data area managed by PHIGS. While the application is responsible for creating the inital buffer through a call to STORE, PHIGS manages this area such that there is sufficient memory for the specific inquiry. The data record within the store buffer is accessed by the pointer pointed to by *paths*.

*struct_id*
Identifier of the structure to be inquired for descendants.

*order* Order in which the path lists are to be returned. This is an enumerated type defined in phigs.h to have the following values:

| | |
|---|---|
| PORDER_TOP_FIRST | *top first* |
| PORDER_BOTTOM_FIRST | *bottom first* |

*depth* Depth (maximum number of references) of path lists to return.

*store* The memory buffer PHIGS is to use for storing the information returned. This buffer must exist prior to calling this function (see CREATE STORE (3P)).

**C Output Parameters**

*error_ind*
A pointer to the location to store the error number of any error detected by this function.

*paths* A pointer to a pointer to a structure Pelem_ref_list_list, containing a list of the specified structure's structure path lists. Pelem_ref_list_list is defined in phigs.h as follows:

```
typedef struct{
        Pint            num_elem_ref_lists;   /* number of execute reference lists */
        Pelem_ref_list  *elem_ref_lists;      /* list of execute reference lists */
} Pelem_ref_list_list;
```

The *num_elem_ref_lists* component specifies the number of structure path lists or execute reference lists. The *elem_ref_lists* component is a pointer to a list, *num_elem_ref_lists* long, of Pelem_ref_list structures containing the specified structure's execute reference lists. Pelem_ref_list is defined in phigs.h as follows:

```
typedef struct{
        Pint        num_elem_refs;   /* number of execute references */
        Pelem_ref   *elem_refs;      /* list of execute references */
} Pelem_ref_list;
```

*elem_refs* is a pointer to a list *num_elem_refs* long of Pelem_ref structures containing the structure identifier and element number of each execute reference structure element in the execute reference list. Pelem_ref is defined in phigs.h as follows:

```
typedef struct {
        Pint    struct_id;   /* structure identifier */
        Pint    elem_pos;    /* element number */
} Pelem_ref;
```

**Execution**

When INQUIRE PATHS DESCENDENTS is called, *paths* is filled with list(s) identifying the EXECUTE STRUCTURE structure elements which are referenced by *struct_id*, in the order of traversal. These EXECUTE STRUCTURE structure elements are represented as (structure identifier, element position) pairs, giving the parent structure identifier and the position of the EXECUTE STRUCTURE structure element. The bottom-most element of a structure network, if included in a returned path, is indicated by a (structure identifier, element position) pair containing the identifier of the bottom-most structure and an element position of 0. *order* and *depth* are used to determine the portion of each path returned. The number of references returned in each path is specified by *depth*; a *depth* of 0 returns all the references in the path. In case of truncation, *order* determines whether the head (PORDER_TOP_FIRST) or the tail (PORDER_BOTTOM_FIRST) portion of a path is returned. If a path truncation results in two or more partial paths with the same set of element references, only one of the identical path portions is returned.

For example, specifying PORDER_TOP_FIRST and a depth of 0 returns all paths from *struct_id*. Specifying PORDER_TOP_FIRST and a depth of 1 returns each EXECUTE STRUCTURE structure element in *struct_id* as a separate path list. Specifying PORDER_BOTTOM_FIRST and a depth of 1 returns all the bottom-most structures of the network.

**ERRORS**

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

201 Ignoring function, the specified structure does not exist

207 Ignoring function, the specified path depth is less than zero

**SEE ALSO**

EXECUTE STRUCTURE (3P)
INQUIRE PATHS TO ANCESTORS (3P)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

INQUIRE PATTERN FACILITIES – inquire pattern facilities available on a workstation type

**SYNOPSIS**

**C Syntax**

```
void
pinq_pat_facs ( type, error_ind, predefined )
Pint        type;          workstation type
Pint        *error_ind;    OUT error indicator
Pint        *predefined;   OUT number of predefined pattern indices
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE PATTERN FACILITIES to determine the number of predefined pattern indices available on a specified workstation type.

**C Input Parameters**

*type* Workstation type.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*predefined*

A pointer to an integer that returns the number of predefined pattern indices available on workstations of type *type*.

**ERRORS**

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

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

052 Ignoring function, workstation type not recognized by the implementation

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

**SEE ALSO**

SET PATTERN REPRESENTATION (3P)
INQUIRE PATTERN REPRESENTATION (3P)

**NAME**

INQUIRE PATTERN REPRESENTATION – inquire a pattern representation on a workstation

**SYNOPSIS**

**C Syntax**

**void**
**pinq_pat_rep ( ws, index, type, store, error_ind, rep, )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **index;** | *pattern index* |
| **Pinq_type** | **type;** | *type of returned value* |
| **Pstore** | **store;** | *handle to Store object* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Ppat_rep** | ****rep;** | *OUT pattern representation* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE PATTERN REPRESENTATION to determine the pattern representation on a specified workstation for a given pattern index.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing data associated with the device state. This buffer is passed as the *store* argument.

The store buffer is a data area managed by PHIGS. While the application is responsible for creating the inital buffer through a call to STORE, PHIGS manages this area such that there is sufficient memory for the specific inquiry. The data record within the store buffer is accessed by the pointer pointed to by *rep*.

*ws* Workstation identifier.

*index* Entry to be returned from the workstation's table of pattern representations; if this entry is not present in the table and the type of returned value parameter is REALIZED, the representation for pattern index 1 is returned.

*type* An enumerated value specifying whether the inquired values are to be returned as the values originally specified by the application (SET), or as the values actually being used by the workstation if any of the application-specified values had to be mapped to ones available on the workstation (REALIZED). Valid values are defined in phigs.h as:

| | |
|---|---|
| PINQ_SET | *return application-specified value* |
| PINQ_REALIZED | *return value available on the workstation* |

*store* The memory buffer PHIGS is to use for storing the information returned. This buffer must exist prior to calling this function (see CREATE STORE (3P)).

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*rep* A pointer to a pointer to a Ppat_rep structure in which the system returns the pattern representation at index in the workstation table of pattern representations. Ppat_rep is defined in phigs.h as follows:

```
typedef struct {
        Pint_size   dims;       /* pattern's dimensions */
        Pint        *colr_array; /* colour index array *
} Ppat_rep;
```

The Pint_size structure used to define the pattern dimensions is defined in phigs.h as follows:

```
typedef struct {
        Pint    size_x;    /* dimension (number of divisions) along X */
        Pint    size_y;    /* dimension (number of divisions) along Y */
} Pint_size;
```

The *colr_array* component is a pointer to an array of the colour indices defining the pattern, of the dimensions defined by the *dims* component.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT, OUTIN, nor MO)

112 Ignoring function, the pattern index value is less than one

101 Ignoring function, the specified representation has not been defined

109 Ignoring function, interior style PATTERN is not supported on the workstation

**SEE ALSO**

INQUIRE LIST OF PATTERN INDICES (3P)
SET PATTERN REPRESENTATION (3P)
INQUIRE PREDEFINED PATTERN REPRESENTATION (3P)
INQUIRE PATTERN REPRESENTATION PLUS (3P+)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

INQUIRE PATTERN REPRESENTATION PLUS – inquire an extended pattern representation on a workstation

**SYNOPSIS**

**C Syntax**

**void**
**pinq_pat_rep_plus (ws, index, type, store, error_ind, rep )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **index;** | *pattern index* |
| **Pinq_type** | **type;** | *type of returned value* |
| **Pstore** | **store;** | *OUT pointer to buffer* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Ppat_rep_plus** | ****rep;** | *OUT pattern representation* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

INQUIRE PATTERN REPRESENTATION PLUS inquires an extended pattern representation on a workstation.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing data associated with the device state. This buffer is passed as the *store* argument.

The store buffer is a data area managed by PHIGS. While the application is responsible for creating the inital buffer through a call to STORE, PHIGS manages this area such that there is sufficient memory for the specific inquiry. The data record within the store buffer is accessed by the pointer pointed to by *rep*.

*ws* The workstation identifier.

*index* The pattern index.

*type* An enumerated value specifying whether the values to be returned are those originally specified by the application (PINQ_SET), or those resulting after PHIGS mapped them to ones available on the workstation (PINQ_REALIZED). A Pinq_type structure is defined as:

```
typedef enum {
        PINQ_SET,
        PINQ_REALIZED
} Pinqtype;
```

*store* The memory buffer PHIGS is to use for storing the information returned. This buffer must exist prior to calling this function (see CREATE STORE (3P)).

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*rep* A pointer to a pointer to a Ppat_rep_plus structure which returns the specified extended pattern representation. Ppat_rep_plus is defined as:

```
typedef struct {
        Pint_size   dims;       /* pattern's dimensions */
        Pint        type;       /* colour model */
        Pcoval      *colr_array; /* array of colours */
} Ppat_rep_plus;
```

See SET PATTERN REPRESENTATION PLUS for a full description of the contents of this structure.

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the specified pattern index is not present in the pattern bundle table on the workstation and the specified type of returned value is *Realized*, the representation for pattern index one is returned.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the following error numbers to indicate the reason for non-availability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT, OUTIN, nor MO)

101 Ignoring function, the specified representation has not been defined

109 Ignoring function, interior style PATTERN is not supported on the workstation

112 Ignoring function, the pattern index value is less than one

**SEE ALSO**

SET PATTERN REPRESENTATION PLUS (3P)
INQUIRE PATTERN REPRESENTATION (3P)
INQUIRE PREDEFINED PATTERN REPRESENTATION PLUS (3P+)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

INQUIRE PHIGS FACILITIES– inquire the current list of PHIGS facilities defined in the PHIGS description table

**SYNOPSIS**

**C Syntax**

**void**
**pinq_phigs_facs ( length, start, error_ind, open_ws, open_ar, num_names, char_sets, length_list, norm_filt, inv_filt )**

| | | |
|---|---|---|
| **Pint** | **length;** | *length of application list* |
| **Pint** | **start;** | *starting position* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pint** | ***open_ws;** | *OUT max. num. simultaneously open ws* |
| **Pint** | ***open_ar;** | *OUT max. num. simultaneously open archive files* |
| **Pint** | ***num_names;** | *OUT number of available names for name sets* |
| **Pint_list** | ***char_sets;** | *OUT list of character sets* |
| **Pint** | ***length_list;** | *OUT length of list in PHIGS* |
| **Pint** | ***norm_filt;** | *OUT maximum length of norm filter list for ISS* |
| **Pint** | ***inv_filt;** | *OUT maximum length of inverted filter list for ISS* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE PHIGS FACILITIES to obtain a list of the available PHIGS facilities from the PHIGS description table.

**C Input Parameters**

*length* The number of list elements for which the application has allocated memory in the output parameter *char_sets→ints*. 0 may be specified, in order to have *length_list* return the total number of elements in the list of character sets.

*start* The starting position in the list at which the inquiry is to begin.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*open_ws*

A pointer to an integer that returns the *maximum number of simultaneously open workstations* supported.

*open_ar*

A pointer to an integer that returns the *maximum number of simultaneously open archive files* supported.

*num_names*

A pointer to an integer that returns the number of available names for name sets.

*char_sets*

A pointer to a data structure that returns a portion of the list of available character sets, starting with the *start* number item. Pint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint    num_ints;       /* number of Pints in list */
        Pint    *ints;          /* list of integers */
} Pint_list;
```

Prior to calling this function, the *ints* field of the Pint_list structure must contain a pointer to an application supplied buffer. This buffer must be at least as large as the *length* parameter.

*length_list*
A pointer to an integer that returns the length of the list in PHIGS.

*norm_filt*
A pointer to an integer that returns the maximum length of the norm filter list for Incremental Spatial Search.

*inv_filt* A pointer to an integer that returns the maximum length of the inverted filter list for Incremental Spatial Search.

**ERRORS**

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

**SEE ALSO**

PHIGS DESCRIPTION TABLE (7P)

**NAME**

INQUIRE PICK DEVICE STATE – inquire state of a pick device

**SYNOPSIS**

**C Syntax**

**void**
**pinq_pick_st ( ws, dev, type, store, err, op_mode, echo_switch, filter, init_status,**
**init_pick, prompt_echo, echo_area, path_data, pick_order )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **dev;** | *pick device number* |
| **Pinq_type** | **type;** | *type of returned value* |
| **Pstore** | **store;** | *handle to store object* |
| **Pint** | ***err;** | *OUT error indicator* |
| **Pop_mode** | ***op_mode;** | *OUT operating mode* |
| **Pecho_switch** | ***echo_switch;** | *OUT echo switch* |
| **Pfilter** | ****filter;** | *OUT pick filter* |
| **Pin_status** | ***init_status;** | *OUT initial pick status* |
| **Ppick_path** | ****init_pick;** | *OUT initial pick path* |
| **Pint** | ***prompt_echo;** | *OUT prompt/echo type* |
| **Plimit** | ***echo_area;** | *OUT echo area* |
| **Ppick_data** | ****pick_data;** | *OUT data record* |
| **Ppath_order** | ***path_order;** | *OUT path order* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE PICK DEVICE STATE to retrieve the current state of the specified pick device.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing data associated with the device state. This buffer is passed as the *store* argument.

The store buffer is a data area managed by PHIGS. While the application is responsible for creating the inital buffer through a call to STORE, PHIGS manages this area such that there is sufficient memory for the specific inquiry. The pick device data record within the store buffer is accessed by the pointer pointed to by pick_data.

*ws* Workstation identifier. An integer specifying the workstation with which the specified pick device is associated.

*dev* The device number of the pick device. See the *Available Devices* section of INITIALIZE PICK for a description of the available devices.

*type* An enumerated value specifying whether the values to be returned are those originally specified by the application (PINQ_SET), or those resulting after PHIGS mapped them to ones available on the workstation (PINQ_REALIZED). A Pinq_type structure is defined as:

```
typedef enum {
        PINQ_SET,
        PINQ_REALIZED
} Pinq_type;
```

*store* The memory buffer PHIGS is to use for storing the information returned. This buffer must exist prior to calling this function (see CREATE STORE (3P)).

**C Output Parameters**

*err* The error indicator. See the *Execution* section below for a description of its use. See the *Errors* section below for the possible values it may return.

*op_mode*

The operating mode. Pop_mode is an enumerated type with the following values:

POP_REQ
POP_SAMPLE
POP_EVENT

*echo_switch*

The echo state. Pecho_switch is an enumerated type with the following values:

PSWITCH_ECHO
PSWITCH_NO_ECHO

*filter* A pointer to a pointer to the location to store the device's detectability filter. Pfilter is defined in phigs.h as follows:

```
typedef struct {
        Pint_list       incl_set;       /* inclusion set */
        Pint_list       excl_set;       /* exclusion set */
} Pfilter;
```

Pint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint            num_ints;       /* number of Pints in list */
        Pint            *ints;          /* list of integers        */
} Pint_list;
```

The *num_ints* component specifies the number of elements in the list. The *ints* component is a pointer to a list *num_ints* long.

*init_status*

A pointer to the initial pick status. Pin_status is an enumerated type with the following values:

```
typedef enum {
        PIN_STATUS_NONE,
        PIN_STATUS_OK,
        PIN_STATUS_NO_IN
} Pin_status;
```

*init_pick*

A pointer to the initial pick path. Ppick_path is defined in phigs.h as follows:

```
typedef struct {
        Pint            depth;          /* pick path_list depth */
        Ppick_path_elem *path_list;     /* pick path */
} Ppick_path;
```

Ppick_path_elem is defined in phigs.h as follows:

```
typedef struct {
        Pint            struct_id;      /* structure identifier */
        Pint            pick_id;        /* hierarchical pick identifier */
        Pint            elem_pos;       /* element sequence number */
} Ppick_path_elem;
```

*prompt_echo*

The prompt/echo type desired. See the *Available Devices* Section of INITIALIZE PICK for a description of the available types.

*echo_area*

A pointer to a variable of type Plimit that contains the echo area of the device. Plimit is defined in

phigs.h as follows:

```
typedef struct {
        Pfloat          x_min;          /* x min */
        Pfloat          x_max;          /* x max */
        Pfloat          y_min;          /* y min */
        Pfloat          y_max;          /* y max */
} Plimit;
```

*pick_data*

Pointer to a pointer that points to the pick device state within store. Ppick_data is defined in phigs.h as follows:

```
typedef struct {
    union {
        struct {
                Pint    unused;
        } pet_r1;
    } pets;
} Ppick_data;
```

*path_order*

Pointer to the path order. Ppath_order is an enumerated type with the following values:

```
typedef enum {
        PORDER_TOP_FIRST,
        PORDER_BOTTOM_FIRST
} Ppath_order;
```

**Execution**

INQUIRE PICK DEVICE STATE returns the current state of the specified pick device, which is stored in the workstation state list of the workstation associated with the device. The current state includes the operating mode, echo switch, initial pick, prompt/echo type, echo area, pick path order, data record, and detectability filter. See SET PICK MODE for a description of the operating mode and the echo switch, and how to set these values. See INITIALIZE PICK for a description of the initial pick, prompt/echo type, echo area, pick path order, and data record contents, and how to set these values. See SET PICK FILTER for a description of the pick filter and how to set it.

In the event of an error, the *error indicator* indicates the error number of the error detected, and no other output data is returned. If no error is detected, the *error indicator* is set to zero, and the inquired information will be available in the output parameters. Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

060 Ignoring function, specified workstation is not of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

**SEE ALSO**

INITIALIZE PICK (3P)
SET PICK MODE (3P)
SET PICK FILTER (3P)
INQUIRE PICK DEVICE STATE 3 (3P)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

INQUIRE PICK DEVICE STATE 3 – inquire state of a pick device

**SYNOPSIS**

**C Syntax**

**void**
**pinq_pick_st3 ( ws, dev, type, store, err, op_mode, echo_switch, filter, init_status, init_pick, prompt_echo, echo_vol, path_data, pick_order )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **dev;** | *pick device number* |
| **Pinq_type** | **type;** | *type of returned value* |
| **Pstore** | **store;** | *handle to store object* |
| **Pint** | ***err;** | *OUT error indicator* |
| **Pop_mode** | ***op_mode;** | *OUT operating mode* |
| **Pecho_switch** | ***echo_switch;** | *OUT echo switch* |
| **Pfilter** | ****filter;** | *OUT pick filter* |
| **Pin_status** | ***init_status;** | *OUT initial pick status* |
| **Ppick_path** | ****init_pick;** | *OUT initial pick path* |
| **Pint** | ***prompt_echo;** | *OUT prompt/echo type* |
| **Plimit3** | ***echo_vol;** | *OUT echo area* |
| **Ppick_data3** | ****pick_data;** | *OUT data record* |
| **Ppath_order** | ***path_order;** | *OUT path order* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE PICK DEVICE STATE 3 to retrieve the current state of the specified pick device.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing data associated with the device state. This buffer is passed as the *store* argument.

The store buffer is a data area managed by PHIGS. While the application is responsible for creating the inital buffer through a call to STORE, PHIGS manages this area such that there is sufficient memory for the specific inquiry. The pick device data record within the store buffer is accessed by the pointer pointed to by pick_data.

*ws* Workstation identifier. An integer specifying the workstation with which the specified pick device is associated.

*dev* The device number of the pick device. See the *Available Devices* section of INITIALIZE PICK for a description of the available devices.

*type* An enumerated value specifying whether the values to be returned are those originally specified by the application (PINQ_SET), or those resulting after PHIGS mapped them to ones available on the workstation (PINQ_REALIZED). A Pinq_type structure is defined as:

```
typedef enum {
        PINQ_SET,
        PINQ_REALIZED
} Pinq_type;
```

*store* The memory buffer PHIGS is to use for storing the information returned. This buffer must exist prior to calling this function (see CREATE STORE (3P)).

**C Output Parameters**

*err* The error indicator. See the *Execution* section below for a description of its use. See the *Errors* section below for the possible values it may return.

*op_mode*

The operating mode. Pop_mode is an enumerated type with the following values:

POP_REQ
POP_SAMPLE
POP_EVENT

*echo_switch*

The echo state. Pecho_switch is an enumerated type with the following values:

PSWITCH_ECHO
PSWITCH_NO_ECHO

*filter* A pointer to a pointer to the location to store the device's detectability filter. Pfilter is defined in phigs.h as follows:

```
typedef struct {
        Pint_list       incl_set;       /* inclusion set */
        Pint_list       excl_set;       /* exclusion set */
} Pfilter;
```

Pint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint            num_ints;       /* number of Pints in list */
        Pint            *ints;          /* list of integers        */
} Pint_list;
```

*init_status*

A pointer to the initial pick status. Pin_status is an enumerated type with the following values:

```
typedef enum {
        PIN_STATUS_NONE,
        PIN_STATUS_OK,
        PIN_STATUS_NO_IN
} Pin_status;
```

*init_pick*

A pointer to the initial pick path. Ppick_path is defined in phigs.h as follows:

```
typedef struct {
        Pint            depth;          /* pick path_list depth */
        Ppick_path_elem *path_list;     /* pick path */
} Ppick_path;
```

Ppick_path_elem is defined in phigs.h as follows:

```
typedef struct {
        Pint            struct_id;      /* structure identifier */
        Pint            pick_id;        /* hierarchical pick identifier */
        Pint            elem_pos;       /* element sequence number */
} Ppick_path_elem;
```

*prompt_echo*

The prompt/echo type desired. See the *Available Devices* Section of INITIALIZE PICK for a description of the available types.

*echo_vol*

A pointer to a Plimit3 structure defining the *x*, *y*, and *z* components of the echo volume, in Device Coordinates. Plimit3 is defined in phigs.h as follows:

```
typedef struct {
```

```
        Pfloat      x_min;      /* minimum x coordinate value */
        Pfloat      x_max;      /* maximum x coordinate value */
        Pfloat      y_min;      /* minimum y coordinate value */
        Pfloat      y_max;      /* maximum y coordinate value */
        Pfloat      z_min;      /* minimum z coordinate value */
        Pfloat      z_max;      /* maximum z coordinate value */
    } Plimit3;
```

*pick_data*

Pointer to a pointer that points to the pick device state within the store. Pick_data3 is defined in phigs.h as follows:

```
typedef struct {
    union {
        struct {
                Pint    unused;
        } pet_r1;
    } pets;
} Ppick_data3;
```

*path_order*

A pointer to the path order. Ppath_order is an enumerated type with the following values:

```
typedef enum {
        PORDER_TOP_FIRST,
        PORDER_BOTTOM_FIRST
} Ppath_order;
```

**Execution**

INQUIRE PICK DEVICE STATE 3 returns the current state of the specified pick device, which is stored in the workstation state list of the workstation associated with the device. The current state includes the operating mode, echo switch, initial pick, prompt/echo type, echo volume, pick path order, data record, and pick filter. See SET PICK MODE for a description of the operating mode and the echo switch, and how to set these values. See INITIALIZE PICK 3 for a description of the initial pick, prompt/echo type, echo volume, pick path order, and data record contents, and how to set these values. See SET PICK FILTER for a description of the pick filter and how to set it.

In the event of an error, the *error indicator* indicates the error number of the error detected, and no other output data is returned. If no error is detected, the *error indicator* is set to zero, and the inquired information will be available in the output parameters. Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

060 Ignoring function, specified workstation is not of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

**SEE ALSO**

INITIALIZE PICK 3 (3P)
SET PICK MODE (3P)
SET PICK FILTER (3P)
INQUIRE PICK DEVICE STATE (3P)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

INQUIRE POLYLINE FACILITIES – inquire list of polyline facilities from workstation description table

**SYNOPSIS**

**C Syntax**

```
void
pinq_line_facs ( type, length, start, error_ind, facilities, total_length )
Pint          type;          workstation type
Pint          length;        length of application list
Pint          start;         starting position
Pint          *error_ind;    OUT error indicator
Pline_facs    *facilities;   OUT polyline facilities
Pint          *total_length; OUT length of list in PHIGS
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

INQUIRE POLYLINE FACILITIES obtains a list of the polyline facilities supported on the specified type of workstation.

**C Input Parameters**

*type* Get the polyline facilities for this workstation type.

*length* The number of *ints* in the *facilities→types→*ints output parameter for which the application has allocated memory. *length* is the number of list elements that the system can return in *types→*ints. If a value of 0 is used here, no data will be returned in the *types→ints* list, but the total number of elements will be returned in *total_length*.

*start* Starting position of inquiry. The elements in the list, beginning with the item number specified by *start*, are copied sequentially into *types→ints* until *types→ints* is full or all the elements have been copied.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*facilities*

A pointer to a structure in which the system returns the portion of the list of polyline facilities from the workstation description table, starting with start. Pline_facs is defined in phigs.h as:

```
typedef struct {
        Pint_list   types;          /* list of line types */
        Pint        widths;         /* number of available line widths */
        Pint        num_widths;     /* number of available line widths */
        Pfloat      nom_width;      /* nominal line width */
        Pfloat      min_width;      /* minimum line width */
        Pfloat      max_width;      /* maximum line width */
        Pint        num_pred_inds;  /* number of predefined bundles */
} Pline_facs;
```

And Pint_list is defined in phigs.h as:

```
typedef struct {
        Pint    num_ints;   /* number of integers */
        Pint    *ints;      /* list of integers */
} Pint_list;
```

Prior to calling this function, the *ints* field of the Pint_list structure must contain a pointer to an

application supplied buffer. This buffer must be at least as large as the corresponding length parameter.

*total_length*
A pointer to an integer in which to return the total length of the list. This is the value required for *length* if all the items in the list are to be returned.

**ERRORS**

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

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

052 Ignoring function, workstation type not recognized by the implementation

059 Ignoring function, the specified workstation does not have output capability (in other words, the workstation category is neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

**SEE ALSO**

PHIGS WORKSTATION DESCRIPTION TABLE (7P)

**NAME**

INQUIRE POLYLINE FACILITIES PLUS– inquire workstation type extended polyline facilities

**SYNOPSIS**

**C Syntax**

**void**
**pinq_line_facs_plus ( type, lt_length, lt_start, sm_length, sm_start, error_ind, tot_lt_length, tot_sm_length, facilities)**

| | | |
|---|---|---|
| **Pint** | **type;** | *workstation type* |
| **Pint** | **lt_length;** | *number of line types to return* |
| **Pint** | **lt_start;** | *starting position* |
| **Pint** | **sm_length;** | *number of shading methods to return* |
| **Pint** | **sm_start;** | *starting position* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pint** | ***tot_lt_length;** | *OUT length of line type list in PHIGS* |
| **Pint** | ***tot_sm_length;** | *OUT length of shading method list in PHIGS* |
| **Pline_facs_plus** | ***facilities;** | *OUT extended polyline facilities* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

INQUIRE POLYLINE FACILITIES PLUS returns the extended polyline support available from a specified workstation type.

**C Input Parameters**

*type* The workstation type.

*lt_length*
The number of line types to return.

*lt_start* The index of first line type to return.

*sm_length*
The number of shading methods to return.

*sm_start*
The index of first shading method to return.

**C Output Parameters**

*error_ind*
A pointer to the location to store the error number of any error detected by this function.

*tot_lt_length*
A pointer to the location to store the total number of line types available.

*tot_sm_length*
A pointer to the location to store the total number of shading methods available.

*facilities*
A pointer to the Pline_facs_plus structure that returns the requested information. Pline_facs_plus is defined as:

```
typedef struct {
        Pint_list   types;          /* list of available line types */
        Pint        num_widths;     /* number of available line widths */
        Pfloat      nom_width;      /* nominal line width */
        Pfloat      min_width;      /* minimum line width */
        Pfloat      max_width;      /* maximum line width */
        Pint        num_pred_inds;  /* number of predefined bundles */
        Pint_list   shads;          /* list of shading methods */
```

```
} Pline_facs_plus;
```

Pint_list is defined as:

```
typedef struct {
        Pint    num_ints;    /* number of Pints in list */
        Pint    *ints;       /* list of integers */
} Pint_list;
```

Prior to calling this function, the *ints* field of the Pint_list structures must contain a pointer to an application supplied buffer. This buffer must be at least as large as the corresponding *length* parameter.

Defined shading methods are:

| | | |
|---|---|---|
| 1 | PSD_NONE | *No Shading* |
| 2 | PSD_COLOUR | *Colour Interpolation Shading* |

**Execution**

If the inquired information is available, the error indicator is returned as zero and the requested information is returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the below error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

052 Ignoring function, workstation type not recognized by the implementation

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

059 Ignoring function, the specified workstation does not have output capability (i.e. the workstation category is neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

**SEE ALSO**

SET POLYLINE REPRESENTATION PLUS (3P+)
INQUIRE POLYLINE REPRESENTATION PLUS (3P+)
INQUIRE PREDEFINED POLYLINE REPRESENTATION PLUS (3P+)

**NAME**

INQUIRE POLYLINE REPRESENTATION – inquire a polyline representation on a workstation

**SYNOPSIS**

**C Syntax**

**void**
**pinq_line_rep ( ws, index, type, error_ind, rep )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **index;** | *polyline index* |
| **Pinq_type** | **type;** | *type of returned value* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pline_bundle** | ***rep;** | *OUT polyline representation* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE POLYLINE REPRESENTATION to determine the current attribute values for a specified entry in a specified workstation's table of defined polyline representations.

See the description of the subroutine SET POLYLINE REPRESENTATION for information on the meaning of these attribute values.

**C Input Parameters**

*ws* Workstation identifier.

*index* Entry to be returned from the workstation's table of polyline representations; if this entry is not present in the table and the *type of returned value* parameter is REALIZED, the representation for polyline index 1 is returned.

*type* An enumerated value specifying whether the inquired values are to be returned as the values originally specified by the application (SET), or as the values actually being used by the workstation if any of the application-specified values had to be mapped to ones available on the workstation (REALIZED). Valid values are defined in phigs.h as:

| | |
|---|---|
| PINQ_SET | *return application-specified value* |
| PINQ_REALIZED | *return value available on the workstation* |

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*rep* A pointer to a Pline_bundle structure in which the system returns the polyline representation at *index* in the workstation's table of polyline representations. Pline_bundle is defined in phigs.h as follows:

```
typedef struct {
        Pint    type;       /* line type */
        Pfloat  width;      /* linewidth scale factor */
        Pint    colr_ind;   /* colour index */
} Pline_bundle;
```

Values for *type* are:

| | |
|---|---|
| 1 | PLINE_SOLID |
| 2 | PLINE_DASH |
| 3 | PLINE_DOT |
| 4 | PLINE_DOT_DASH |

**Execution**

If the inquired information is available, the error indicator is returned as zero and the requested information is returned in the output parameters.

If the specified polyline index is not present in the polyline bundle table on the workstation and the specified type of returned value is *Realized*, the representation for polyline index one is returned.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the below error numbers to indicate the reason for non-availability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (in other words, the workstation category is neither OUTPUT, OUTIN, nor MO)

100 Ignoring function, the bundle index value is less than one

101 Ignoring function, the specified representation has not been defined

**SEE ALSO**

SET POLYLINE REPRESENTATION (3P)
INQUIRE LIST OF POLYLINE INDICES (3P)
INQUIRE PREDEFINED POLYLINE REPRESENTATION (3P)
INQUIRE POLYLINE REPRESENTATION PLUS (3P+)

**NAME**

INQUIRE POLYLINE REPRESENTATION PLUS – inquire the values of a workstation's extended polyline representation

**SYNOPSIS**

**C Syntax**

**void**
**pinq_line_rep_plus ( ws, index, type, error_ind, rep )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **index;** | *polyline index* |
| **Pinq_type** | **type;** | *type of returned value* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pline_bundle_plus** | ***rep;** | *OUT polyline representation* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

INQUIRE POLYLINE REPRESENTATION PLUS returns the attribute values of a specified entry in a workstation extended polyline bundle table. See SET POLYLINE REPRESENTATION PLUS for a full description of the contents of the bundle table entry and a description of the extended polyline bundle table.

**C Input Parameters**

*ws* Workstation identifier.

*index* Index of the entry to be returned from the workstation extended polyline bundle table. If this entry is not present in the table, and the type of returned value parameter is PINQ_REALIZED, the representation for polyline index one is returned.

*type* An enumerated value specifying whether the values to be returned are those originally specified by the application (PINQ_SET), or those resulting after PHIGS mapped them to ones available on the workstation (PINQ_REALIZED). A Pinq_type structure is defined as:

```
typedef enum {
        PINQ_SET,
        PINQ_REALIZED
} Pinq_type;
```

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*rep* A pointer to a Pline_bundle_plus structure that returns the specified extended polyline representation. Pline_bundle_plus is defined as:

```
typedef struct {
        Pint    type;           /* line type */
        Pfloat  width;          /* linewidth scale factor */
        Pgcolr  colr;           /* polyline colour */
        Pint    shad_meth;      /* polyline shading method */
        Pint    approx_type;    /* curve approximation type */
        Pfloat  approx_val;     /* curve approximation value */
} Pline_bundle_plus;
```

See SET POLYLINE REPRESENTATION PLUS for a full description of the contents of this structure.

**Execution**

If the inquired information is available, the error indicator is returned as zero and the requested information is returned in the output parameters.

If the specified polyline index is not present in the polyline bundle table on the workstation and the specified type of returned value is *Realized*, the representation for polyline index one is returned.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the below error numbers to indicate the reason for non-availability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT, OUTIN, nor MO)

100 Ignoring function, the bundle index value is less than one

101 Ignoring function, the specified representation has not been defined

**SEE ALSO**

SET POLYLINE REPRESENTATION PLUS (3P+)
INQUIRE POLYLINE REPRESENTATION (3P)
INQUIRE PREDEFINED POLYLINE REPRESENTATION PLUS (3P+)

**NAME**

INQUIRE POLYMARKER FACILITIES – inquire list of workstation polymarker facilities

**SYNOPSIS**

**C Syntax**

**void**
**pinq_marker_facs ( type, length, start, error_ind, facilities, total_length )**

| | | |
|---|---|---|
| **Pint** | **type;** | *workstation type* |
| **Pint** | **length;** | *length of application list* |
| **Pint** | **start;** | *starting position* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pmarker_facs** | ***facilities;** | *OUT polymarker facilities* |
| **Pint** | ***total_length;** | *OUT length of list in PHIGS* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

INQUIRE POLYMARKER FACILITIES obtains a list of the polymarker facilities available on the specified type of workstation.

**C Input Parameters**

*type* Get the polymarker facilities for this workstation type.

*length* The number of *ints* in the *facilities→types* output parameter for which the application has allocated memory. *length* is the number of list elements that the system can return in *facilities→types.ints*. If a value of 0 is used here, no data will be returned in the *facilities→types.ints* list, but the total number of elements will be returned in *total_length*.

*start* Starting position of inquiry. The elements in the list, beginning with the item number specified by *start*, are copied sequentially into *facilities→types.ints* until *facilities→types.ints* is full or all the elements have been copied.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*facilities*

A pointer to a structure in which the system returns the portion of the list of polymarker facilities from the workstation description table, starting with start. Pmarker_facs is defined in phigs.h as:

```
typedef struct {
        Pint_list   types;          /* list of marker types */
        Pint        num_sizes;      /* number of available marker sizes */
        Pfloat      nom_size;       /* nominal marker size */
        Pfloat      min_size;       /* minimum marker size */
        Pfloat      max_size;       /* maximum marker size */
        Pint        num_pred_inds;  /* number of predefined bundles */
} Pmarker_facs;
```

And Pint_list is defined as:

```
typedef struct {
        Pint    num_ints;   /* number of integers */
        Pint    *ints;      /* list of integers */
} Pint_list;
```

Prior to calling this function, the *ints* field of the Pint_list structure must contain a pointer to an

application supplied buffer. This buffer must be at least as large as the *length* parameter.

*total_length*
A pointer to an integer in which the system returns the total number of elements in the list. This is the value required for *length* if all elements in the list are to be returned.

**ERRORS**

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

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

052 Ignoring function, workstation type not recognized by the implementation

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

**SEE ALSO**

PHIGS WORKSTATION DESCRIPTION TABLE (7P)

**NAME**

INQUIRE POLYMARKER REPRESENTATION – inquire a polymarker representation on a workstation

**SYNOPSIS**

**C Syntax**

**void**
**pinq_marker_rep ( ws, index, type, error_ind, rep )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **index;** | *polymarker index* |
| **Pinq_type** | **type;** | *type of returned value* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pmarker_bundle** | ***rep;** | *OUT polymarker representation* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE POLYMARKER REPRESENTATION to determine the current attribute values for a specified entry in a specified workstation's table of defined polymarker representations.

See the description of the subroutine SET POLYMARKER REPRESENTATION for information on the meaning of these attribute values.

**C Input Parameters**

*ws* Workstation identifier.

*index* Entry to be returned from the workstation's table of polymarker representations; if this entry is not present in the table and the *type of returned value* parameter is REALIZED, the representation for polymarker index 1 is returned.

*type* An enumerated value specifying whether the inquired values are to be returned as the values originally specified by the application (SET), or as the values actually being used by the workstation if any of the application-specified values had to be mapped to ones available on the workstation (REALIZED). Valid values are defined in phigs.h as:

| | |
|---|---|
| PINQ_SET | *Return Application-specified Value* |
| PINQ_REALIZED | *Return Value Available on the Workstation* |

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*rep* A pointer to a Pmarker_bundle structure in which the system returns the polymarker representation at *index* in the workstation's table of polymarker representations. Pmarker_bundle is defined in phigs.h as follows:

```
typedef struct {
        Pint    type;       /* marker type */
        Pfloat  size;       /* marker size scale factor */
        Pint    colr_ind;   /* colour index */
} Pmarker_bundle;
```

Values for *type* are:

| | |
|---|---|
| 1 | PMARKER_DOT |
| 2 | PMARKER_PLUS |
| 3 | PMARKER_ASTERISK |
| 4 | PMARKER_CIRCLE |
| 5 | PMARKER_CROSS |

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the specified polymarker index is not present in the polymarker bundle table on the workstation and the specified type of returned value is *Realized*, the representation for polymarker index one is returned.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the following error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (in other words, the workstation category is neither OUTPUT, OUTIN, nor MO)

100 Ignoring function, the bundle index value is less than one

101 Ignoring function, the specified representation has not been defined

**SEE ALSO**

SET POLYMARKER REPRESENTATION (3P)
INQUIRE LIST OF POLYMARKER INDICES (3P)
INQUIRE PREDEFINED POLYMARKER REPRESENTATION (3P)
INQUIRE POLYMARKER REPRESENTATION PLUS (3P+)

**NAME**

INQUIRE POLYMARKER REPRESENTATION PLUS – inquire an extended polymarker representation on a workstation

**SYNOPSIS**

**C Syntax**

**void**
**pinq_marker_rep_plus ( wsid, index, type, error_ind, rep )**

| | | |
|---|---|---|
| **Pint** | **wsid;** | *workstation identifier* |
| **Pint** | **index;** | *polymarker index* |
| **Pinq_type** | **type;** | *type of returned value* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pmarker_bundle_plus** | ***rep;** | *OUT polymarker representation* |

**Required PHIGS Operating States**

(PHOP, WSOP, ∗, ∗)

**DESCRIPTION**

**Purpose**

INQUIRE POLYMARKER REPRESENTATION PLUS returns the attribute values of a specified entry in a workstation extended polymarker bundle table. See SET POLYMARKER REPRESENTATION PLUS for a full description of the contents of the bundle table entry and a description of the extended polymarker bundle table.

**C Input Parameters**

*wsid* Workstation identifier.

*index* Index of the entry to be returned from the workstation extended polymarker bundle table. If this entry is not present in the table, and the type of returned value parameter is PINQ_REALIZED, the representation for polymarker index one is returned.

*type* An enumerated value specifying whether the values to be returned are those originally specified by the application (PINQ_SET), or those resulting after PHIGS mapped them to ones available on the workstation (PINQ_REALIZED). A Pinq_type structure is defined as:

```
typedef enum {
        PINQ_SET,
        PINQ_REALIZED
} Pinq_type;
```

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*rep* A pointer to a Pmarker_bundle_plus structure which returns the specified extended polymarker representation. Pmarker_bundle_plus is defined as:

```
typedef struct {
        Pint    type;   /* marker type */
        Pfloat  size;   /* marker size scale factor */
        Pgcolr  colr;   /* marker colour */
} Pmarker_bundle_plus;
```

Pgcolr is defined as:

```
typedef struct {
        Pint      type;     /* indirect, RGB, CIE, HSV, HLS */
        union {
           Pint   ind;      /* index in workstation colour bundle table */
           struct {
              Pfloat      x;       /* red, hue, etc. */
              Pfloat      y;       /* green, saturation, lightness, etc. */
              Pfloat      z;       /* blue, value, saturation, etc. */
           } general;
        } val;
} Pgcolr;
```

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the specified polymarker index is not present in the polymarker bundle table on the workstation and the specified type of returned value is *Realized*, the representation for polymarker index one is returned.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the following error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT, OUTIN, nor MO)

100 Ignoring function, the bundle index value is less than one

101 Ignoring function, the specified representation has not been defined

**SEE ALSO**

SET POLYMARKER REPRESENTATION PLUS (3P+)
INQUIRE POLYMARKER REPRESENTATION (3P)
INQUIRE PREDEFINED POLYMARKER REPRESENTATION PLUS (3P+)

**NAME**

INQUIRE POSTED STRUCTURES – inquire the list of structures posted to a workstation

**SYNOPSIS**

**C Syntax**

```
void
pinq_posted_structs ( ws, length, start, error_ind, list, total_length )
Pint                ws;              workstation identifier
Pint                length;          length of application list
Pint                start;           starting position
Pint                *error_ind;      OUT error indicator
Pposted_struct_list *list;           OUT list of posted structures
Pint                *total_length;   OUT length of list in PHIGS
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE POSTED STRUCTURES to obtain the list of currently posted structures from a workstation's state list.

**C Input Parameters**

*ws* The *workstation identifier* of the workstation whose state list is queried.

*length* The number of *postings* in the *list* output parameter for which the application has allocated memory. *length* is the number of list elements (posted structures) that the system can return in *list→postings*. If a value of 0 is used here, no data will be returned in the *list→postings* list, but the total number of posted structures in the PHIGS state list will be returned in *total_length*.

*start* Starting position of inquiry into the PHIGS state list of current structures. The elements of the list of posted structures, beginning with the item number specified by *start*, are copied sequentially into *list→postings* until *list→postings* is full or all the posted structures have been copied.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*list* A pointer to a Pposted_struct_list in which the system returns the portion of the list of currently posted structures starting at the entry specified with *start*. Pposted_struct_list is defined in phigs.h as follows:

```
typedef struct {
        Pint            num_postings;   /* number of structure postings */
        Pposted_struct  *postings;      /* list of postings */
} Pposted_struct_list;
```

The pointer *list→postings* must be initialized to an array of *length* Pstructpost elements. Pposted_struct is defined in phigs.h as follows:

```
typedef struct {
        Pint    id;         /* structure id */
        Pfloat  disp_pri;   /* structure priority */
} Pposted_struct;
```

*total_length*

A pointer to an integer in which the system returns the total number of elements in the PHIGS state list of currently used posted structures. This is the value required for *length* if all posted structures are to be returned.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT, OUTIN, nor MO)

**ALSO**

POST STRUCTURE (3P)
INQUIRE SET OF WORKSTATIONS TO WHICH POSTED (3P)
INQUIRE PATHS DESCENDENTS (3P)

**NAME**

INQUIRE PREDEFINED COLOUR MAPPING REPRESENTATION – inquire a predefined colour mapping representation for the specified workstation type

**SYNOPSIS**

**C Syntax**

**void**
**pinq_pred_colr_map_rep ( type, index, store, error_ind, map_method, map_data )**

| | | |
|---|---|---|
| **Pint** | **type;** | *workstation type* |
| **Pint** | **index;** | *predefined index* |
| **Pstore** | **store;** | *Handle to store data* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pint** | ***map_method** | *OUT colour mapping method* |
| **Pcolr_map_data** | ****map_data** | *OUT colour mapping data* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

INQUIRE PREDEFINED COLOUR MAPPING REPRESENTATION returns the values of a specified entry in the workstation type's table of predefined colour mapping representations.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing data associated with the device state. This buffer is passed as the *store* argument.

The store buffer is a data area managed by PHIGS. While the application is responsible for creating the inital buffer through a call to STORE, PHIGS manages this area such that there is sufficient memory for the specific inquiry. The data record within the store buffer is accessed by the pointer pointed to by *map_data*.

*type* The type of workstation whose capabilities are inquired.

*index* Entry in the workstation table of predefined colour mapping representations to be returned.

*store* The memory buffer PHIGS is to use for storing the information returned. This buffer must exist prior to calling this function (see CREATE STORE (3P)).

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*map_data*

A pointer to a pointer to a Pcolr_map_data structure that contains the information specific to the data mapping method. PHIGS will allocate space for the structure in the Pstore object and return the pointer to that allocated space in this parameter. Pcolr_map_data is defined in phigs.h as follows:

```
typedef union {
        struct {
                Pint            colr_model;
                Pfloat_list     weights;
                Pcolr_rep_list  colrs;
        } method_r2
        struct {
                Pint            colr_model;
                Pfloat_list_list colr_lists;
        } method_r3
} Pcolr_map_data;
```

Pfloat_list is defined in phigs.h as follows:

```
typedef struct {
        Pint            num_floats;     /* number of Pfloats in list */
        Pfloat          *floats;        /* list of floats *
} Pfloat_list;
```

Pfloat_list_list is defined in phigs.h as follows:

```
typedef struct {
        Pint            num_lists;      /* number of lists in list */
        Pfloat_list     *lists;         /* list of float lists */
} Pfloat_list_list;
```

Pcolr_rep_list is defined in phigs.h as follows:

```
typedef struct {
        Pint            num_colr_reps;  /* number of colours */
        Pcolr_rep       *colr_reps;     /* array of colours */
} Pcolr_rep_list;
```

Pcolr_rep is defined in phigs.h as follows:

```
typedef union {
        Prgb            rgb;            /* Red Green Blue colour specification */
        Pcieluv         cieluv;         /* CIE L*U*V* colour specification */
        Phls            hls;            /* Hue Lightness Saturation colour specification */
        Phsv            hsv;            /* Hue Saturation Value colour specification */
        Pdata           unsupp;         /* Colour in unsupported colour model */
} Pcolr_rep;
```

Prgb is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          red;            /* red, hue, etc */
        Pfloat          green;          /* green, saturation, lightness, etc */
        Pfloat          blue;           /* blue, value, saturation, etc */
} Prgb;
```

Pcieluv is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          cieluv_x;       /* x coefficient */
        Pfloat          cieluv_y;       /* y coefficient */
        Pfloat          cieluv_y_lum;   /* y luminance */
} Pcieluv;
```

Phsv is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          hue;            /* hue */
        Pfloat          satur;          /* saturation */
        Pfloat          value;          /* value */
} Phsv;
```

Phls is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          hue;            /* hue */
        Pfloat          lightness;      /* lightness */
        Pfloat          satur;          /* saturation */
```

```
} Phls;
```

Pdata is defined in phigs.h as follows:

```
typedef struct {
        size_t          size;           /* size of data */
        char            *data           /*pointer to data */
} Pdata;
```

There is no data associated with mapping method PCOLR_MAP_TRUE. The data for method PCOLR_MAP_PSEUDO is a list of weights and a list of direct colours. *colr_model* specifies the type of colours given in the list of colours, *colrs*. The colour models are defined in phigs.h as follows:

1 PMODEL_RGB
2 PMODEL_CIELUV
3 PMODEL_HSV
4 PMODEL_HLS

The data for mapping method PCOLR_MAP_PSEUDO_N is *N* lists of direct colours. These colours may be of the same types as for PCOLR_MAP_PSEUDO.

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the following error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

051 Ignoring function, this information unavailable for this workstation type

052 Ignoring function, workstation type not recognized by the implementation

059 Ignoring function, the specified workstation does not have output capability (i.e. the workstation category is neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

102 Ignoring function, specified representation has not been predefined on this workstation

121 Ignoring function, the colour mapping index is less than 0.

**SEE ALSO**

SET COLOUR MAPPING REPRESENTATION (3P+)
INQUIRE COLOUR MAPPING REPRESENTATION (3P+)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

INQUIRE PREDEFINED COLOUR REPRESENTATION – inquire a predefined colour representation for a workstation type

**SYNOPSIS**

**C Syntax**

```
void
pinq_pred_colr_rep ( type, index, error_ind, bundle )
Pint        type;          workstation type
Pint        index;         predefined index
Pint        *error_ind;    OUT error indicator
Pcolr_rep   *bundle;       OUT predefined colour rep
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE PREDEFINED COLOUR REPRESENTATION to determine the predefined colour representation for a specified workstation type at a given colour index.

**C Input Parameters**

*type* Type of workstation.

*index* Entry in the workstation table of predefined colour representations to be returned.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*bundle* A pointer to a Pcolr_rep structure in which the system returns the colour representation at *index* on the workstation table of predefined colour representations. Pcolr_rep is defined in phigs.h as:

```
typedef union {
        Prgb        rgb;        /* Red Green Blue colour specification */
        Pcieluv     cieluv;     /* CIE L*U*V* colour specification */
        Phls        hls;        /* Hue Lightness Saturation colour specification */
        Phsv        hsv;        /* Hue Saturation Value colour specification */
        Pdata       unsupp;     /* Colour in unsupported colour model */
} Pcolr_rep;
```

Prgb is defined in phigs.h as follows:

```
typedef struct {
        Pfloat      red;        /* red, hue, etc */
        Pfloat      green;      /* green, saturation, lightness, etc */
        Pfloat      blue;       /* blue, value, saturation, etc */
} Prgb;
```

Pcieluv is defined in phigs.h as follows:

```
typedef struct {
        Pfloat      cieluv_x;       /* x coefficient */
        Pfloat      cieluv_y;       /* y coefficient */
        Pfloat      cieluv_y_lum;   /* y luminance */
} Pcieluv;
```

Phsv is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          hue;            /* hue */
        Pfloat          satur;          /* saturation */
        Pfloat          value;          /* value */
} Phsv;
```

Phls is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          hue;            /* hue */
        Pfloat          lightness;      /* lightness */
        Pfloat          satur;          /* saturation */
} Phls;
```

Pdata is defined in phigs.h as follows:

```
typedef struct {
        size_t          size;           /* size of data */
        char            *data           /*pointer to data */
} Pdata;
```

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the below error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

052 Ignoring function, workstation type not recognized by the implementation

059 Ignoring function, the specified workstation does not have output capability (in other words, the workstation category is neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

113 Ignoring function, the colour index value is less than zero

102 Ignoring function, the specified representation has not been predefined on this workstation

**SEE ALSO**

INQUIRE COLOUR FACILITIES (3P)
SET COLOUR REPRESENTATION (3P)
INQUIRE COLOUR REPRESENTATION (3P)
INQUIRE LIST OF COLOUR INDICES (3P)

**NAME**

INQUIRE PREDEFINED DEPTH CUE REPRESENTATION – inquire predefined depth cue representation for the specified workstation type

**SYNOPSIS**

**C Syntax**

```
void
pinq_pred_dcue_rep ( type, index, error_ind, bundle )
Pint            type;           workstation type
Pint            index;          predefined index
Pint            *error_ind;     OUT error indicator
Pdcue_bundle    *bundle;        OUT predefined depth cue rep
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

INQUIRE PREDEFINED DEPTH CUE REPRESENTATION returns the attribute values of a specified entry in the workstation type's table of predefined depth cue representations. See SET DEPTH CUE REPRESENTATION for a full description of the contents of the bundle table entry and a description of the depth cue representation table.

**C Input Parameters**

*type* The workstation type.

*index* The index of the predefined entry to be returned.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*bundle* A pointer to a Pdcue_bundle data structure containing attribute values defining a predefined depth cue representation. A Pdcue_bundle structure is defined in phigs.h as follows:

```
typedef struct {
        Pdcue_mode      mode;           /* depth cue mode */
        Pfloat          ref_planes[2];  /* depth cue reference planes */
        Pfloat          scaling[2];     /* depth cue scaling */
        Pgcolr          colr;           /* depth cue colour */
} Pdcue_bundle;
```

Entry 0 in ref_planes is the back reference plane. Entry 0 in scaling is back scaling.

Pdcue_mode is defined as follows:

```
typedef enum {
        PSUPPRESSED,
        PALLOWED
} Pdcue_mode;
```

Pgcolr is defined as:

```
typedef struct {
        Pint    type;   /* indirect, RGB, CIE, HSV, HLS */
        union {
                Pint    ind;    /* index in workstation colour bundle table */
                struct {
                        Pfloat  x;      /* red, hue, etc. */
                        Pfloat  y;      /* green, saturation, lightness, etc. */
                        Pfloat  z;      /* blue, value, saturation, etc. */
                } general;
        } val;
} Pgcolr;
```

Constants defined for colour type are:

| | | |
|---|---|---|
| 0 | PINDIRECT | *Indirect* |
| 1 | PMODEL_RGB | *Red, Green, Blue* |
| 2 | PMODEL_CIELUV | *CIE* |
| 3 | PMODEL_HSV | *Hue, Saturation, Value* |
| 4 | PMODEL_HLS | *Hue, Lightness, Saturation* |

The index member of the *val* union is used for type PINDIRECT. The general member is used for the other types.

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the following error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

051 Ignoring function, this information unavailable for this workstation type

052 Ignoring function, workstation type not recognized by the implementation

059 Ignoring function, the specified workstation does not have output capability (i.e., the workstation category is neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

102 Ignoring function, specified representation has not been predefined on this workstation

603 Ignoring function, depth cue index is less than zero

**SEE ALSO**

SET DEPTH CUE REPRESENTATION (3P+)
INQUIRE DEPTH CUE REPRESENTATION (3P+)
INQUIRE DEPTH CUE FACILITIES (3P+)
INQUIRE LIST OF DEPTH CUE INDICES (3P+)

**NAME**

INQUIRE PREDEFINED EDGE REPRESENTATION – inquire a predefined edge representation for a workstation type

**SYNOPSIS**

**C Syntax**

```
void
pinq_pred_edge_rep ( type, index, error_ind, bundle )
Pint            type;           workstation type
Pint            index;          predefined index
Pint            *error_ind;     OUT error indicator
Pedge_bundle    *bundle;        OUT predefined edge rep
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE PREDEFINED EDGE REPRESENTATION to determine the predefined edge representation for a specified workstation type at a given edge index.

**C Input Parameters**

*type* Type of workstation.

*index* Entry to be returned from the workstation table of predefined edge representations.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*bundle* A pointer to a Pedge_bundle structure in which the system returns the edge representation at *index* in the workstation table of predefined edge representations. Pedge_bundle is defined in phigs.h as:

```
typedef struct {
        Pedge_flag      flag;       /* edge flag */
        Pint            type;       /* edge type */
        Pfloat          width;      /* edge width scale factor */
        Pint            colr_ind;   /* edge colour index */
} Pedge_bundle;
```

Valid values for the Pedge_flag enumerated type are defined in phigs.h as:

```
typedef enum {
        PEDGE_OFF,
        PEDGE_ON
} Pedge_flag
```

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the below error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

052 Ignoring function, workstation type not recognized by the implementation

059 Ignoring function, the specified workstation does not have output capability (in other words, the workstation category is neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

100 Ignoring function, the bundle index value is less than one

102 Ignoring function, the specified representation has not been predefined on this workstation

**SEE ALSO**

INQUIRE EDGE FACILITIES (3P)
SET EDGE REPRESENTATION (3P)
INQUIRE EDGE REPRESENTATION (3P)
INQUIRE PREDEFINED EDGE REPRESENTATION PLUS (3P+)

**NAME**

INQUIRE PREDEFINED EDGE REPRESENTATION PLUS – inquire a predefined extended edge representation for a specified workstation type

**SYNOPSIS**

**C Syntax**

```
void
pinq_pred_edge_rep_plus ( type, index, error_ind, bundle )
Pint                type;           workstation type
Pint                index;          predefined index
Pint                *error_ind;     OUT error indicator
Pedge_bundle_plus   *bundle;        OUT predefined edge rep
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

INQUIRE PREDEFINED EDGE REPRESENTATION PLUS returns the attribute values of a specified predefined entry in the extended edge bundle table of a workstation type's workstation description table. See SET EDGE REPRESENTATION PLUS for a full description of the contents of the bundle table entry and a description of the extended edge bundle table.

**C Input Parameters**

*type* The workstation type.

*index* The index of the predefined entry to be returned.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*bundle*

A pointer to a Pedge_bundle_plus structure that returns the requested representation. Pedge_bundle_plus is defined as:

```
typedef struct {
        Pedge_flag      flag;       /* edge flag */
        Pint            type;       /* edgetype */
        Pfloat          scale;      /* edgewidth scale factor */
        Pgcolr          colr;       /* edge colour */
} Pedge_bundle_plus;
```

Valid values for the Pedge_flag enumerated type are defined in phigs.h as:

```
typedef enum {
        PEDGE_OFF,
        PEDGE_ON
} Pedge_flag;
```

Pgcolr is defined as:

```
typedef struct {
        Pint    type;   /* indirect, RGB, CIE, HSV, HLS */
        union {
                Pint    ind;    /* index in workstation colour bundle table */
                struct {
                        Pfloat  x;      /* red, hue, etc. */
                        Pfloat  y;      /* green, saturation, lightness, etc. */
                        Pfloat  z;      /* blue, value, saturation, etc. */
                } general;
        } val;
} Pgcolr;
```

Constants defined for colour type are:

| | | |
|---|---|---|
| 0 | PINDIRECT | *Indirect* |
| 1 | PMODEL_RGB | *Red, Green, Blue* |
| 2 | PMODEL_CIELUV | *CIE* |
| 3 | PMODEL_HSV | *Hue, Saturation, Value* |
| 4 | PMODEL_HLS | *Hue, Lightness, Saturation* |

The index member of the *val* union is used for type PINDIRECT. The general member is used for the other types.

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the below error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

051 Ignoring function, this information unavailable for this workstation type

052 Ignoring function, workstation type not recognized by the implementation

059 Ignoring function, the specified workstation does not have output capability (i.e., the workstation category is neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

100 Ignoring function, the bundle index value is less than one

102 Ignoring function, the specified representation has not been predefined on this workstation

**SEE ALSO**

SET EDGE REPRESENTATION PLUS (3P+)
INQUIRE EDGE REPRESENTATION PLUS (3P+)
INQUIRE PREDEFINED EDGE REPRESENTATION (3P)

**NAME**

INQUIRE PREDEFINED INTERIOR REPRESENTATION – inquire a predefined interior representation for a workstation type

**SYNOPSIS**

**C Syntax**

**void**
**pinq_pred_int_rep ( type, index, error_ind, bundle )**

| | | |
|---|---|---|
| **Pint** | **type;** | *workstation type* |
| **Pint** | **index;** | *predefined index* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pint_bundle** | ***bundle;** | *OUT predefined interior rep* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE PREDEFINED INTERIOR REPRESENTATION to determine the predefined interior representation for a specified workstation type at a given interior index.

**C Input Parameters**

*type* Type of workstation.

*index* Entry to be returned from the workstation table of predefined interior representations.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*bundle* A pointer to a Pint_bundle structure in which the system returns the interior representation at *index* in the workstation table of predefined interior representations. Pint_bundle is defined in phigs.h as:

```
typedef struct {
        Pint_style    style;      /* interior style */
        Pint          style_ind;  /* interior style index */
        Pint          colr_ind;   /* interior colour index */
} Pint_bundle;
```

Pint_style is defined in phigs.h as:

```
typedef enum {
        PSTYLE_HOLLOW,
        PSTYLE_SOLID,
        PSTYLE_PAT,
        PSTYLE_HATCH,
        PSTYLE_EMPTY
} Pint_style;
```

See SET INTERIOR STYLE for a description of each style.

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the below error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

052 Ignoring function, workstation type not recognized by the implementation

059 Ignoring function, the specified workstation does not have output capability (in other words, the workstation category is neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

100 Ignoring function, the bundle index value is less than one

102 Ignoring function, the specified representation has not been predefined on this workstation

**SEE ALSO**

INQUIRE INTERIOR FACILITIES (3P)
SET INTERIOR REPRESENTATION (3P)
INQUIRE INTERIOR REPRESENTATION (3P)
INQUIRE PREDEFINED INTERIOR REPRESENTATION PLUS (3P+)

**NAME**

INQUIRE PREDEFINED INTERIOR REPRESENTATION PLUS – inquire the values of a predefined extended interior representation for a specified workstation type

**SYNOPSIS**

**C Syntax**

**pinq_pred_int_rep_plus ( type, index, error_ind, bundle )**

| | | |
|---|---|---|
| **Pint** | **type;** | *workstation type* |
| **Pint** | **index;** | *interior index* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pint_bundle_plus** | ***bundle;** | *OUT predefined interior rep* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

INQUIRE PREDEFINED INTERIOR REPRESENTATION PLUS returns the attribute values of a specified predefined entry in the extended interior bundle table of a workstation type's workstation description table. See SET INTERIOR REPRESENTATION PLUS for a full description of the contents of the bundle table entry and a description of the extended interior bundle table.

**C Input Parameters**

*type* The workstation type.

*index* Index of the predefined entry to be returned.

**C Output Parameters**

*error_ind*

A pointer to the location for storing the error number of any error this function detects.

*bundle* A pointer to a Pint_bundle_plus structure in which the system returns the specified predefined extended interior representation. Pint_bundle_plus is defined as:

```
typedef struct {
        Pint_style      style;          /* interior style */
        Pint            style_ind;      /* interior style index */
        Pgcolr          colr;           /* interior colour */
        Pint            refl_eqn;       /* reflectance equation */
        Pint            shad_meth;      /* shading method */
        Prefl_props     refl_props;     /* area properties */
        Pint_style      back_style;     /* interior style */
        Pint            back_style_ind; /* interior style index */
        Pgcolr          back_colr;      /* interior colour */
        Pint            back_refl_eqn;  /* back reflectance equation */
        Pint            back_shad_meth; /* back shading method */
        Prefl_props     back_refl_props; /* back area properties */
        Pint            approx_type;    /* approximation method */
        Pfloat          approx_val[2];  /* approximation values, u and v */
} Pint_bundle_plus;
```

Pint_style is defined as:

```
typedef enum {
        PSTYLE_HOLLOW,
        PSTYLE_SOLID,
        PSTYLE_PAT,
        PSTYLE_HATCH,
        PSTYLE_EMPTY
} Pint_style;
```

See SET INTERIOR STYLE for a description of each style.

Pgcolr is defined as:

```
typedef struct {
        Pint    type;     /* indirect, RGB, CIE, HSV, HLS */
        union {
                Pint    ind;      /* index in workstation colour bundle table */
                struct {
                        Pfloat  x;        /* red, hue, etc. */
                        Pfloat  y;        /* green, saturation, lightness, etc. */
                        Pfloat  z;        /* blue, value, saturation, etc. */
                } general;
        } val;
} Pgcolr;
```

Constants defined for colour type are:

| | | |
|---|---|---|
| 0 | PINDIRECT | *Indirect* |
| 1 | PMODEL_RGB | *Red, Green, Blue* |
| 2 | PMODEL_CIELUV | *CIE* |
| 3 | PMODEL_HSV | *Hue, Saturation, Value* |
| 4 | PMODEL_HLS | *Hue, Lightness, Saturation* |

The index member of the *val* union is used for type PINDIRECT. The general member is used for the other types.

Prefl_props is defined as:

```
typedef struct {
        Pfloat  ambient_coef;   /* ambient reflectance coefficient */
        Pfloat  diffuse_coef;   /* diffuse reflectance coefficient */
        Pfloat  specular_coef;  /* specular reflectance coefficient */
        Pgcolr  specular_colr;  /* specular colour */
        Pfloat  specular_exp;   /* specular exponent */
        Pfloat  transpar_coef;  /* transparency coefficient */
} Prefl_props;
```

The values for ambient, diffuse, specular, and transparency coefficients must all be in the range [0,1]. The specular exponent must be greater than zero.

Defined reflectance equation values are:

| | | |
|---|---|---|
| 1 | PREFL_NONE | *No Reflectance Calculation Performed* |
| 2 | PREFL_AMBIENT | *Use Ambient Term* |
| 3 | PREFL_AMB_DIFF | *Use Ambient and Diffuse Terms* |
| 4 | PREFL_AMB_DIFF_SPEC | *Use Ambient, Diffuse, and Specular Terms* |

Defined constants for interior shading method are:

| | | |
|---|---|---|
| 1 | PSD_NONE | *No Shading* |
| 2 | PSD_COLOUR | *Colour Interpolation Shading* |
| 3 | PSD_DOT_PRODUCT | *Dot Product Interpolation Shading* |
| 4 | PSD_NORMAL | *Normal Interpolation Shading* |

Constants defined for surface approximation type are:

| | |
|---|---|
| 1 | PSURF_WS_DEP |
| 2 | PSURF_CONSTANT_PARAMETRIC_BETWEEN_KNOTS |
| 3 | PSURF_CHORDAL_SIZE_WC |
| 4 | PSURF_CHORDAL_SIZE_NPC |
| 5 | PSURF_CHORDAL_SIZE_DC |
| 6 | PSURF_PLANAR_DEVIATION_WC |
| 7 | PSURF_PLANAR_DEVIATION_NPC |
| 8 | PSURF_PLANAR_DEVIATION_DC |

See SET SURFACE APPROXIMATION CRITERIA for a description of the meaning of these constants.

**Execution**

If the inquired information is available, the error indicator is returned as zero, and information is returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are undefined, and the error indicator is set to one of the error numbers given below to indicate the reason for nonavailability.

The extended interior bundle table contains bundled entries of the PHIGS and PHIGS PLUS interior attributes. During traversal, the attribute values of these bundles are used when the corresponding aspect source flag in the workstation state list is set to BUNDLED.

The PHIGS interior bundle table is a portion of the extended table. The function INQUIRE PREDEFINED INTERIOR REPRESENTATION returns the following attributes from the extended table: interior style, interior style index, and colour index.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

051 Ignoring function, the information unavailable for this workstation type

052 Ignoring function, workstation type not recognized by the implementation

059 Ignoring function, the specified workstation does not have output capability (i.e., the workstation category is neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

100 Ignoring function, the bundle index value is less than one

102 Ignoring function, the specified representation has not been predefined on this workstation

**SEE ALSO**

SET INTERIOR REPRESENTATION PLUS (3P+)
INQUIRE INTERIOR REPRESENTATION PLUS (3P+)
INQUIRE INTERIOR FACILITIES PLUS (3P+)
INQUIRE PREDEFINED INTERIOR REPRESENTATION (3P)

**NAME**

INQUIRE PREDEFINED LIGHT SOURCE REPRESENTATION – inquire a predefined light source representation

**SYNOPSIS**

**C Syntax**

**void**
**pinq_pred_light_src_rep ( type, index, error_ind, bundle )**

| | | |
|---|---|---|
| **Pint** | **type;** | *workstation type* |
| **Pint** | **index;** | *predefined index* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Plight_src_bundle** | ***bundle;** | *OUT predefined light source rep* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

INQUIRE PREDEFINED LIGHT SOURCE REPRESENTATION returns the attribute values of a specified predefined entry in the light source bundle table of a workstation type's workstation description table. See SET LIGHT SOURCE REPRESENTATION for a full description of the contents of the bundle table entry and a description of the extended light source bundle table.

**C Input Parameters**

*type* The workstation type.

*index* Index of the predefined entry to be returned.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*bundle* A pointer to the location in which to return the light source representation. The Plight_src_bundle data structure contains values needed to define the specified type of light source representation. Plight_src_bundle is defined as:

```
typedef struct {
        Pint            type;   /* light source type */
        Plight_src_rec  rec;    /* light source data record */
} Plight_src_bundle;
```

Plight_src_rec is defined as:

```
typedef union {
        Pamb_light_src_rec      ambient;
        Pdir_light_src_rec      directional;
        Ppos_light_src_rec      positional;
        Pspot_light_src_rec     spot;
} Plight_src_rec;
```

Pamb_light_src_rec is defined as:

```
typedef struct {
        Pgcolr  colr;   /* light source colour */
} Pamb_light_src_rec;
```

Pdir_light_src_rec is defined as:

```
typedef struct {
```

```
        Pgcolr    colr;    /* light source colour */
        Pvec3     dir;     /* light source direction */
} Pdir_light_src_rec;
```

Ppos_light_src_rec is defined as:

```
typedef struct {
        Pgcolr    colr;      /* light source colour */
        Ppoint3   pos;       /* light source position */
        Pfloat    coef[2];   /* attenuation coefficients */
} Ppos_light_src_rec;
```

Pspot_light_src_rec is defined as:

```
typedef struct {
        Pgcolr    colr;      /* light source colour */
        Ppoint3   pos;       /* light source position */
        Pvec3     dir;       /* light source direction */
        Pfloat    exp;       /* concentration exponent */
        Pfloat    coef[2];   /* attenuation coefficients */
        Pfloat    angle;     /* spread angle */
} Pspot_light_src_rec;
```

Pgcolr is defined as:

```
typedef struct {
        Pint                type;          /* indirect, RGB, CIE, HSV, HLS */
        union {
            Pint            ind;           /* index in workstation colour bundle table */
            struct {
                Pfloat      x;             /* red, hue, etc. */
                Pfloat      y;             /* green, saturation, lightness, etc. */
                Pfloat      z;             /* blue, value, saturation, etc. */
            } general;
        } val;
} Pgcolr;
```

Pvec3 is defined as:

```
typedef struct {
        Pfloat    delta_x;   /* x magnitude */
        Pfloat    delta_y;   /* y magnitude */
        Pfloat    delta_z;   /* z magnitude */
} Pvec3;
```

Ppoint3 is defined as:

```
typedef struct {
        Pfloat    x;    /* x coordinate */
        Pfloat    y;    /* y coordinate */
        Pfloat    z;    /* z coordinate */
} Ppoint3;
```

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

See SET LIGHT SOURCE REPRESENTATION for the defined data record formats that may be returned in the data record parameter.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the following error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

051 Ignoring function, this information unavailable for this workstation type

052 Ignoring function, workstation type not recognized by the implementation

059 Ignoring function, the specified workstation does not have output capability (i.e., the workstation category is neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

102 Ignoring function, specified representation has not been predefined on this workstation

614 Ignoring function, the general interior index is less than one

**SEE ALSO**

SET LIGHT SOURCE REPRESENTATION (3P+)
SET LIGHT SOURCE STATE (3P+)
INQUIRE LIST OF LIGHT SOURCE INDICES (3P+)
INQUIRE LIGHT SOURCE REPRESENTATION (3P+)
INQUIRE LIGHT SOURCE FACILITIES (3P+)

**NAME**

INQUIRE PREDEFINED PATTERN REPRESENTATION – inquire a predefined pattern representation for a workstation type

**SYNOPSIS**

**C Syntax**

```
void
pinq_pred_pat_rep ( type, index, store, error_ind, rep )
Pint        type;        workstation type
Pint        index;       predefined index
Pstore      store;       handle to Store object
Pint        *error_ind;  OUT error indicator
Ppat_rep    **rep;       OUT predefined pattern rep
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE PREDEFINED PATTERN REPRESENTATION to determine the predefined pattern representation for a specified workstation type at a given pattern index.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing data associated with the device state. This buffer is passed as the *store* argument.

The store buffer is a data area managed by PHIGS. While the application is responsible for creating the inital buffer through a call to STORE, PHIGS manages this area such that there is sufficient memory for the specific inquiry. The data record within the store buffer is accessed by the pointer pointed to by *rep*.

*type* Workstation type.

*index* Entry to be returned from the workstation table of predefined pattern representations.

*store* The memory buffer PHIGS is to use for storing the information returned. This buffer must exist prior to calling this function (see CREATE STORE (3P)).

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*rep* A pointer to a pointer to a Ppat_rep structure in which the system returns the pattern representation at *index* in the workstation table of predefined pattern representations. Ppat_rep is defined in phigs.h as:

```
typedef struct {
        Pint_size   dims;       /* pattern's dimensions */
        Pint        *colr_array; /* colour index array *
} Ppat_rep;
```

The Pint_size structure used to define the pattern dimensions is defined in phigs.h as:

```
typedef struct {
        Pint    size_x;   /* dimension (number of divisions) along X */
        Pint    size_y;   /* dimension (number of divisions) along Y */
} Pint_size;
```

The *colr_array* component of Ppat_rep is a pointer to an array of the colour indices defining the pattern of the dimensions defined by the *dims* component.

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the below error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

052 Ignoring function, workstation type not recognized by the implementation

059 Ignoring function, the specified workstation does not have output capability (i.e., the workstation category is neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

102 Ignoring function, the specified representation has not been predefined on this workstation

112 Ignoring function, the pattern index value is less than one

**SEE ALSO**

INQUIRE PATTERN FACILITIES (3P)
SET PATTERN REPRESENTATION (3P)
INQUIRE PATTERN REPRESENTATION (3P)
INQUIRE PREDEFINED PATTERN REPRESENTATION PLUS (3P+)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

INQUIRE PREDEFINED PATTERN REPRESENTATION PLUS – inquire a predefined extended pattern representation for a specified workstation type

**SYNOPSIS**

**C Syntax**

```
void
pinq_pred_pat_rep_plus ( type, index, store, error_ind, rep )
Pint            type;          workstation type
Pint            index;         predefined index
Pstore          store;         handle to Store object
Pint            *error_ind;    OUT error indicator
Ppat_rep_plus   **rep;         OUT predefined pattern rep
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

INQUIRE PREDEFINED PATTERN REPRESENTATION PLUS inquires a predefined extended pattern representation for a specified workstation type.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing data associated with the device state. This buffer is passed as the *store* argument.

The store buffer is a data area managed by PHIGS. While the application is responsible for creating the inital buffer through a call to STORE, PHIGS manages this area such that there is sufficient memory for the specific inquiry. The data record within the store buffer is accessed by the pointer pointed to by *rep*.

*type* The workstation type.

*index* Index of the predefined entry to be returned.

*store* The memory buffer PHIGS is to use for storing the information returned. This buffer must exist prior to calling this function (see CREATE STORE (3P)).

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*rep* A pointer to a pointer to a Ppat_rep_plus structure that returns the requested representation. Ppat_rep_plus is defined in phigs.h as follows:

```
typedef struct {
        Pint_size       dims;           /* pattern's dimensions */
        Pint            type;           /* colour type */
        Pcoval          *colr_array;    /* array of colours */
} Ppat_rep_plus;
```

Pint_size is definied in phigs.h as follows:

```
typedef struct {
        Pint            size_x;
        Pint            size_y;
} Pint_size;
```

Pcoval is definied in phigs.h as follows:

```
typedef union {
        Pint            ind;            /* index in workstation colour bundle table */
```

```
        Pcolr_rep       direct;             /* direct colour components */
} Pcoval;
```

Pcolr_rep is defined in phigs.h as follows:

```
typedef union {
        Prgb            rgb;            /* Red Green Blue colour specification */
        Pcieluv         cieluv;         /* CIE L*U*V* colour specification */
        Phls            hls;            /* Hue Lightness Saturation colour specification */
        Phsv            hsv;            /* Hue Saturation Value colour specification */
        Pdata           unsupp;         /* Colour in unsupported colour model */
} Pcolr_rep;
```

Prgb is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          red;            /* red, hue, etc */
        Pfloat          green;          /* green, saturation, lightness, etc */
        Pfloat          blue;           /* blue, value, saturation, etc */
} Prgb;
```

Pcieluv is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          cieluv_x;       /* x coefficient */
        Pfloat          cieluv_y;       /* y coefficient */
        Pfloat          cieluv_y_lum;   /* y luminance */
} Pcieluv;
```

Phsv is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          hue;            /* hue */
        Pfloat          satur;          /* saturation */
        Pfloat          value;          /* value */
} Phsv;
```

Phls is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          hue;            /* hue */
        Pfloat          lightness;      /* lightness */
        Pfloat          satur;          /* saturation */
} Phls;
```

Pdata is defined in phigs.h as follows:

```
typedef struct {
        size_t          size;           /* size of data */
        char            *data           /*pointer to data */
} Pdata;
```

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the following error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

051 Ignoring function, this information unavailable for this workstation type

052 Ignoring function, workstation type not recognized by the implementation

059 Ignoring function, the specified workstation does not have output capability (i.e., the workstation category is neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

100 Ignoring function, the bundle index value is less than one

102 Ignoring function, the specified representation has not been predefined on this workstation

**SEE ALSO**

SET PATTERN REPRESENTATION PLUS (3P+)
INQUIRE PATTERN REPRESENTATION PLUS (3P+)
INQUIRE PREDEFINED PATTERN REPRESENTATION (3P)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

INQUIRE PREDEFINED POLYLINE REPRESENTATION – inquire a predefined polyline representation for a workstation type

**SYNOPSIS**

**C Syntax**

**void**
**pinq_pred_line_rep ( type, index, error_ind, bundle )**

| | | |
|---|---|---|
| **Pint** | **type;** | *workstation type* |
| **Pint** | **index;** | *predefined index* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pline_bundle** | ***bundle;** | *OUT predefined polyline rep* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE PREDEFINED POLYLINE REPRESENTATION to determine the predefined polyline representation for a specified workstation type at a given polyline index.

**C Input Parameters**

*type* Type of workstation.

*index* Entry to be returned from the workstation table of predefined polyline representations.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*bundle* A pointer to a Pline_bundle data structure in which the system returns the polyline representation at *index* in the workstation table of predefined polyline representations. Pline_bundle is defined in phigs.h as:

```
typedef struct {
        Pint    type;       /* line type */
        Pfloat  width;      /* linewidth scale factor */
        Pint    colr_ind;   /* colour index */
} Pline_bundle;
```

Values for *type* are:

| | |
|---|---|
| 1 | PLINE_SOLID |
| 2 | PLINE_DASH |
| 3 | PLINE_DOT |
| 4 | PLINE_DOT_DASH |

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the below error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

052 Ignoring function, workstation type not recognized by the implementation

059 Ignoring function, the specified workstation does not have output capability (in other words, the workstation category is neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

100 Ignoring function, the bundle index value is less than one

102 Ignoring function, the specified representation has not been predefined on this workstation

**SEE ALSO**

INQUIRE POLYLINE FACILITIES (3P)
SET POLYLINE REPRESENTATION (3P)
INQUIRE POLYLINE REPRESENTATION (3P)
INQUIRE PREDEFINED POLYLINE REPRESENTATION PLUS (3P+)

**NAME**

INQUIRE PREDEFINED POLYLINE REPRESENTATION PLUS – inquire a predefined extended polyline representation from a specified workstation type

**SYNOPSIS**

**C Syntax**

```
void
pinq_pred_line_rep_plus ( type, index, error_ind, bundle )
Pint                type;           workstation type
Pint                index;          predefined index
Pint                *error_ind;     OUT error indicator
Pline_bundle_plus   *bundle;        OUT predefined polyline rep
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

INQUIRE PREDEFINED POLYLINE REPRESENTATION PLUS returns the attribute values of a specified predefined entry in the extended polyline bundle table of a workstation type's workstation description table. See SET POLYLINE REPRESENTATION PLUS for a full description of the contents of the bundle table entry and a description of the extended polyline bundle table.

**C Input Parameters**

*type* The workstation type.

*index* Index of the predefined entry to be returned.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*bundle* A pointer to a Pline_bundle_plus structure that returns the requested representation. Pline_bundle_plus is defined as:

```
typedef struct {
        Pint    type;           /* line type */
        Pfloat  width;          /* linewidth scale factor */
        Pgcolr  colr;           /* polyline colour */
        Pint    shad_meth;      /* polyline shading method */
        Pint    approx_type;    /* curve approximation criteria */
        Pfloat  approx_val;     /* curve approximation criteria */
} Pline_bundle_plus;
```

See SET POLYLINE REPRESENTATION PLUS for a full description of the contents of this structure.

**Execution**

If the inquired information is available, the error indicator is returned as zero and the requested information is returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the below error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

051 Ignoring function, this information unavailable for this workstation type

052 Ignoring function, workstation type not recognized by the implementation

059 Ignoring function, the specified workstation does not have output capability (i.e., the workstation category is neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

100 Ignoring function, the bundle index value is less than one

102 Ignoring function, the specified representation has not been predefined on this workstation

**SEE ALSO**

SET POLYLINE REPRESENTATION PLUS (3P+)
INQUIRE POLYLINE REPRESENTATION PLUS (3P+)
INQUIRE PREDEFINED POLYLINE REPRESENTATION (3P)

**NAME**

INQUIRE PREDEFINED POLYMARKER REPRESENTATION – inquire a predefined polymarker representation for a workstation type

**SYNOPSIS**

**C Syntax**

**void**
**pinq_pred_marker_rep ( type, index, error_ind, bundle )**

| | | |
|---|---|---|
| **Pint** | **type;** | *workstation type* |
| **Pint** | **index;** | *predefined index* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pmarker_bundle** | ***bundle;** | *OUT predefined polymarker rep* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE PREDEFINED POLYMARKER REPRESENTATION to determine the predefined polymarker representation for a specified workstation type at a given polymarker index.

**C Input Parameters**

*type* Type of workstation.

*index* Entry to be returned from the workstation table of predefined polymarker representations.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*bundle* A pointer to a Pmarker_bundle structure, in which the system returns the polymarker representation at *index* in the workstation table of predefined polymarker representations. Pmarker_bundle is defined in phigs.h as:

```
typedef struct {
        Pint    type;       /* marker type */
        Pfloat  size;       /* marker size scale factor */
        Pint    colr_ind;   /* colour index */
} Pmarker_bundle;
```

Values for *type* are:

1 PMARKER_DOT
2 PMARKER_PLUS
3 PMARKER_ASTERISK
4 PMARKER_CIRCLE
5 PMARKER_CROSS

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the below error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

051 Ignoring function, this information is not yet available for this workstation type; open a

workstation of this type and use the specific workstation type

052 Ignoring function, workstation type not recognized by the implementation

059 Ignoring function, the specified workstation does not have output capability (in other words, the workstation category is neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

100 Ignoring function, the bundle index value is less than one

102 Ignoring function, the specified representation has not been predefined on this workstation

**SEE ALSO**

INQUIRE POLYMARKER FACILITIES (3P)
SET POLYMARKER REPRESENTATION (3P)
INQUIRE POLYMARKER REPRESENTATION (3P)
INQUIRE PREDEFINED POLYMARKER REPRESENTATION PLUS (3P+)

**NAME**

INQUIRE PREDEFINED POLYMARKER REPRESENTATION PLUS – inquire a predefined extended polymarker representation for a specified workstation type

**SYNOPSIS**

**C Syntax**

```
void
pinq_pred_marker_rep_plus ( type, index, error_ind, bundle )
Pint                  type;          workstation type
Pint                  index;         predefined index
Pint                  *error_ind;    OUT error indicator
Pmarker_bundle_plus   *bundle;       OUT predefined polymarker rep
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

INQUIRE PREDEFINED POLYMARKER REPRESENTATION PLUS returns the attribute values of a specified predefined entry in the extended polymarker bundle table of a workstation type's workstation description table. See SET POLYMARKER REPRESENTATION PLUS for a full description of the contents of the bundle table entry and a description of the extended polymarker bundle table.

This is a PHIGS Extension function based on PHIGS PLUS and is not part of the PHIGS standard.

**C Input Parameters**

*type* The workstation type.

*index* The index of the predefined entry to be returned.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*bundle* A pointer to a Pmarker_bundle_plus structure that returns the requested representation. Pmarker_bundle_plus is defined in phigs.h as:

```
typedef struct {
        Pint    type;   /* marker type */
        Pfloat  size;   /* marker size scale factor */
        Pgcolr  colr;   /* marker colour */
} Pmarker_bundle_plus;
```

See SET POLYMARKER REPRESENTATION PLUS for a full description of the contents of this structure.

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the below error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

051 Ignoring function, this information unavailable for this workstation type

052 Ignoring function, workstation type not recognized by the implementation

059 Ignoring function, the specified workstation does not have output capability (i.e., the workstation

category is neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

100 Ignoring function, the bundle index value is less than one

102 Ignoring function, the specified representation has not been predefined on this workstation

**SEE ALSO**

SET POLYMARKER REPRESENTATION PLUS (3P+)
INQUIRE POLYMARKER REPRESENTATION PLUS (3P+)
INQUIRE PREDEFINED POLYMARKER REPRESENTATION (3P)

**NAME**

INQUIRE PREDEFINED TEXT REPRESENTATION – inquire a predefined text representation for a workstation type

**SYNOPSIS**

**C Syntax**

```
void
pinq_pred_text_rep ( type, index, error_ind, bundle )
Pint            type;           workstation type
Pint            index;          predefined index
Pint            *error_ind;     OUT error indicator
Ptext_bundle    *bundle;        OUT predefined text rep
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE PREDEFINED TEXT REPRESENTATION to determine the predefined text representation for a specified workstation type at a given text index.

**C Input Parameters**

*type* Type of workstation.

*index* Entry to be returned from the workstation table of predefined text representations.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*bundle* A pointer to a Ptext_bundle structure, in which the system returns the text representation at *index* in the workstation table of predefined text representations. Ptext_bundle is defined in phigs.h as:

```
typedef struct {
        Pint        font;       /* text font */
        Ptext_prec  prec;       /* text precision */
        Pfloat      char_expan; /* character expansion factor */
        Pfloat      char_space; /* character spacing */
        Pint        colr_ind;   /* text colour index */
} Ptext_bundle;
```

Ptext_prec is defined in phigs.h as:

```
typedef enum {
        PPREC_STRING,   /* string precision */
        PPREC_CHAR,     /* character precision */
        PPREC_STROKE    /* stroke precision */
} Ptext_prec;
```

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the below error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

052 Ignoring function, workstation type not recognized by the implementation

059 Ignoring function, the specified workstation does not have output capability (in other words, the workstation category is neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

100 Ignoring function, the bundle index value is less than one

102 Ignoring function, the specified representation has not been predefined on this workstation

**SEE ALSO**

INQUIRE TEXT FACILITIES (3P)
SET TEXT REPRESENTATION (3P)
INQUIRE TEXT REPRESENTATION (3P)
INQUIRE PREDEFINED TEXT REPRESENTATION PLUS (3P+)

**NAME**

INQUIRE PREDEFINED TEXT REPRESENTATION PLUS – inquire a predefined extended text representation for a specified workstation type

**SYNOPSIS**

**C Syntax**

```
void
pinq_pred_text_rep_plus (type, index, error_ind, bundle)
Pint                type;           workstation type
Pint                index;          predefined index
Pint                *error_ind;     OUT error indicator
Ptext_bundle_plus   *bundle;        OUT predefined text rep
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

INQUIRE PREDEFINED TEXT REPRESENTATION PLUS returns the attribute values of a specified predefined entry in the extended text bundle table of a workstation type's workstation description table. See SET TEXT REPRESENTATION PLUS for a full description of the contents of the bundle table entry and a description of the extended text bundle table.

**C Input Parameters**

*type* The workstation type.

*index* Index of the predefined entry to be returned.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*bundle* A pointer to a Ptext_bundle_plus structure that returns the specified extended text representation. Ptext_bundle_plus is defined as:

```
typedef struct {
        Pint        font;           /* text font */
        Ptext_prec  prec;           /* text precision */
        Pfloat      char_expan;     /* character expansion factor */
        Pfloat      char_space;     /* character spacing */
        Pgcolr      colr;           /* text colour */
} Ptext_bundle_plus;
```

See SET TEXT REPRESENTATION PLUS for a full description of the contents of this structure.

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the following error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

051 Ignoring function, this information unavailable for this workstation type

052 Ignoring function, workstation type not recognized by the implementation

059 Ignoring function, the specified workstation does not have output capability (i.e., the workstation

category is neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

100 Ignoring function, the bundle index value is less than one

102 Ignoring function, the specified representation has not been predefined on this workstation

**SEE ALSO**

SET TEXT REPRESENTATION PLUS (3P+)
INQUIRE TEXT REPRESENTATION PLUS (3P+)
INQUIRE PREDEFINED TEXT REPRESENTATION (3P)

**NAME**

INQUIRE PREDEFINED VIEW REPRESENTATION – inquire a predefined view representation for a workstation type

**SYNOPSIS**

**C Syntax**

```
void
pinq_pred_view_rep ( type, index, error_ind, rep )
Pint            type;           workstation type
Pint            index;          predefined view index
Pint            *error_ind;     OUT error indicator
Pview_rep3      *rep;           OUT view representation
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

INQUIRE PREDEFINED VIEW REPRESENTATION determines the predefined view representation for a specified workstation type at a given view index.

**C Input Parameters**

*type* Type of workstation.

*index* Entry to be returned from the workstation table of predefined view representations.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*rep* A pointer to a Pview_rep3 structure in which the system returns the view representation at *index* in the workstation table of predefined view representations. Pview_rep3 is defined in phigs.h as follows:

```
typedef struct {
        Pmatrix3    ori_matrix;     /* orientation matrix */
        Pmatrix3    map_matrix;     /* mapping matrix */
        Plimit3     clip_limit;     /* clipping limits */
        Pclip_ind   xy_clip;        /* x-y clipping indicator */
        Pclip_ind   back_clip;      /* back clipping indicator */
        Pclip_ind   front_clip;     /* front clipping indicator */
} Pview_rep3;
```

The Pmatrix3 type definition is a 4x4 matrix, defined in phigs.h as:

```
typedef  Pfloat     Pmatrix3[4][4];
```

The Plimit3 structure used to define the clipping limits is defined in phigs.h as follows:

```
typedef struct {
        Pfloat  x_min;   /* x min */
        Pfloat  x_max;   /* x max */
        Pfloat  y_min;   /* y min */
        Pfloat  y_max;   /* y max */
        Pfloat  z_min;   /* z min */
        Pfloat  z_max;   /* z max */
} Plimit3;
```

The clipping indicators control whether the clipping limits for the associated plane are active or inactive. Valid values for the Pclip enumerated type are defined in phigs.h as:

```
typedef enum {
        PIND_NO_CLIP,       /* Do Not Clip */
        PIND_CLIP           /* Perform Clipping */
} Pclip_ind;
```

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the below error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

051 Ignoring function, this information is not yet available for this generic workstation type; open a workstation of this type and use the specific workstation type

052 Ignoring function, workstation type not recognized by the implementation

057 Ignoring function, specified workstation is of category MI

062 Ignoring function, this information is not available for this MO workstation type

101 Ignoring function, the specified representation has not been defined

114 Ignoring function, the view index value is less than zero

**SEE ALSO**

INQUIRE VIEW FACILITIES (3P)
SET VIEW REPRESENTATION 3 (3P)
INQUIRE VIEW REPRESENTATION (3P)

**NAME**

INQUIRE RENDERING COLOUR MODEL FACILITIES – inquire list of available direct colour models

**SYNOPSIS**

**C Syntax**

**void**
**pinq_rendering_colr_model_facs( type, length, start, error_ind, models, total_length )**

| | | |
|---|---|---|
| **Pint** | **type;** | *workstation type* |
| **Pint** | **length;** | *length of application's list* |
| **Pint** | **start;** | *starting position* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pint_list** | ***models;** | *OUT rendering colour models* |
| **Pint** | ***total_length;** | *OUT total number of rendering* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE RENDERING COLOUR MODEL FACILITIES to obtain a list of colour model facilities available for rendering for the specified workstation type

**C Input Parameters**

*type* The workstation type.

*length* The number of *ints* in the *models* output parameter for which the application has allocated memory. *length* is the number of list elements that the system can return in *models→ints*. If a value of 0 is used here, no data will be returned in the *models→ints* list, but the total number of elements will be returned in *total_length*.

*start* Starting position of inquiry. The elements in the list, beginning with the item number specified by *start*, are copied sequentially into *models→ints* until *models→ints* is full or all the elements have been copied.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*models* A pointer to a Pint_list that returns the portion of the list of available rendering colour models from the workstation description table, starting with *start*. Pint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint        num_ints;       /* number of Pints in list */
        Pint        *ints;          /* list of integers */
} Pint_list;
```

The *num_ints* component specifies the number of elements in the list. The *ints* component is a pointer to a list *num_ints* long.

The application must allocate memory for *length* elements in the list of *ints*.

*total_length*

A pointer to an integer in which to return the total length of the list. This is the value required for *length* if all the items in the list are to be returned.

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the below error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

052 Ignoring function, workstation type not recognized by the implementation

059 Ignoring function, the specified workstation does not have output capability (i.e., the workstation category is neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, specified workstation category is neither OUTPUT nor OUTIN

-153 List length is less that zero – zero will be used

2201 Start index out of range

**SEE ALSO**

SET RENDERING COLOUR MODEL (3P+)

**NAME**

INQUIRE SET OF OPEN WORKSTATIONS – inquire the current set of open workstations

**SYNOPSIS**

**C Syntax**

**void**
**pinq_open_wss ( length, start, error_ind, idlist, total_length )**

| | | |
|---|---|---|
| **Pint** | **length;** | *length of application list* |
| **Pint** | **start;** | *starting position* |
| **Pint** | ∗**error_ind;** | *OUT error indicator* |
| **Pint_list** | ∗**idlist;** | *OUT list of ws ids* |
| **Pint** | ∗**total_length;** | *OUT length of list in PHIGS* |

**Required PHIGS Operating States**

(PHOP, ∗, ∗, ∗)

**DESCRIPTION**

**Purpose**

Use INQUIRE SET OF OPEN WORKSTATIONS to obtain a list of the identifiers fo the currently open workstations.

**C Input Parameters**

*length* The number of *ints* in the *idlist* output parameter for which the application has allocated memory. *length* is the number of list elements that the system can return in *idlist→ints*. If a value of 0 is used here, no data will be returned in the *idlist→ints* list, but the total number of elements will be returned in *total_length*.

*start* Starting position of inquiry. The elements in the list, beginning with the item number specified by *start*, are copied sequentially into *idlist→ints* until *idlist→ints* is full or all the elements have been copied.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*idlist* A pointer to a Pint_list in which the system returns the portion of the list of currently open workstations starting at the entry specified with *start*. Pint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint   num_ints;    /* number of Pints in list */
        Pint   *ints;       /* list of integers */
} Pint_list;
```

The *num_ints* component specifies the number of workstation identifiers in the list.

The pointer *ints* must be initialized to an array, *length* long, of Pint elements.

*total_length*

A pointer to an integer in which to return the total length of the list. This is the value required for *length* if all the items in the list are to be returned.

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the below error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

2201 Start index out of range

**SEE ALSO**

OPEN WORKSTATION (3P)
CLOSE WORKSTATION (3P)
INQUIRE WORKSTATION STATE VALUE (3P)

**NAME**

INQUIRE SET OF WORKSTATIONS TO WHICH POSTED – inquire list of workstations to which a structure is posted

**SYNOPSIS**

**C Syntax**

**void**
**pinq_wss_posted ( struct_id, length, start, error_ind, ws, total_length )**

| | | |
|---|---|---|
| **Pint** | **struct_id;** | *structure identifier* |
| **Pint** | **length;** | *length of application list* |
| **Pint** | **start;** | *starting position* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pint_list** | ***ws;** | *OUT list of workstations* |
| **Pint** | ***total_length;** | *OUT length of list in PHIGS* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE SET OF WORKSTATIONS TO WHICH POSTED to obtain a list of workstations to which a specified structure is posted.

**C Input Parameters**

*struct_id*

Identifier of the structure for which the list of workstations it is posted to will be returned.

*length* The number of *ints* items in the *ws* output parameter for which the application has allocated memory. *length* is the number of list elements (workstation identifiers) that the system can return in *ints*. If a value of 0 is used here, no data will be returned in the *ints* list, but the total number of workstations the structure is posted to will be returned in *total_length*.

*start* Starting position of inquiry into the list of workstations to which this structure is posted. The elements of the list of workstation identifiers, beginning with the item number specified by *start*, are copied sequentially into *ints* until *ints* is full or all the workstation identifiers have been copied.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*ws* A pointer to a Pint_list structure in which the system returns the list of workstations to which *struct_id* is posted. Pint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint    num_ints;    /* number of Pints in list */
        Pint    *ints:       /* pointer to list of integers */
} Pint_list;
```

The *num_ints* component specifies the number of workstation identifiers in the list. The *ints* component is a pointer to a list, *num_ints* long, of the workstation identifiers.

The application must allocate memory for *length* elements in the *ints* list.

*total_length*

A pointer to an integer in which the system returns the total number of workstations to which the specified structure is posted. This is the value required for *length* if the entire list is to be returned.

**ERRORS**

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

201 Ignoring function, the specified structure does not exist

**SEE ALSO**

POST STRUCTURE (3P)
INQUIRE POSTED STRUCTURES (3P)

**NAME**

INQUIRE STRING DEVICE STATE – inquire state of a STRING device

**SYNOPSIS**

**C Syntax**

**void**
**pinq_string_st( ws, dev, store, err, op_mode, echo_switch, init_string, prompt_echo, echo_area, string_data)**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **dev;** | *string device number* |
| **Pstore** | **store;** | *handle to Store object* |
| **Pint** | ***err;** | *OUT error indicator* |
| **Pop_mode** | ***op_mode** | *OUT operating mode* |
| **Pecho_switch** | ***echo_switch;** | *OUT echo switch* |
| **char** | ****init_string;** | *OUT initial string* |
| **Pint** | ***prompt_echo;** | *OUT prompt/echo type* |
| **Plimit** | ***echo_area;** | *OUT echo area* |
| **Pstring_data** | ****string_data;** | *OUT data record* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE STRING DEVICE STATE to retrieve the current state of the specified STRING device.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing data associated with the device state. This buffer is passed as the *store* argument.

The store buffer is a data area managed by PHIGS. While the application is responsible for creating the inital buffer through a call to STORE, PHIGS manages this area such that there is sufficient memory for the specific inquiry. The string data record within the store buffer is accessed by the pointer pointed to by *string_data*.

*ws* Workstation identifier. An integer specifying the workstation with which the specified STRING device is associated.

*dev* The device number of the string device. See the *Available Devices* section of INITIALIZE STRING for a description of the available devices.

*store* The memory buffer PHIGS is to use for storing the information returned. This buffer must exist prior to calling this function (see CREATE STORE (3P)).

**C Output Parameters**

*err* A pointer to the location to store the error number of any error detected by this function.

*op_mode*

The operating mode. Pop_mode is an enumerated type defined in phigs.h with the following values:

```
typedef enum {
        POP_REQ,
        POP_SAMPLE,
        POP_EVENT
} Pop_mode;
```

*echo_switch*

The echo state. Pecho_switch is an enumerated type defined in phigs.h with the following values:

```
typedef enum {
        PSWITCH_NO_ECHO,
```

```
        PSWITCH_ECHO
} Pecho_switch;
```

*init_string*
A pointer to a char pointer. PHIGS sets the location pointed to by *init_string* to point to the initial string stored in *store*.

*prompt_echo*
The prompt/echo type desired. See the *Available Devices* Section of INITIALIZE STRING for a description of the available types.

*echo_area*
A pointer to a variable of type Plimit that contains the echo area of the device. Plimit is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          x_min;          /* x min */
        Pfloat          x_max;          /* x max */
        Pfloat          y_min;          /* y min */
        Pfloat          y_max;          /* y max */
} Plimit;
```

*string_data*
Pointer to a pointer that points to the string state within *store*. Pstring_data is defined in phigs.h as follows:

```
typedef struct {
        Pint    buffer_size;    /* input buffer size        */
        Pint    init_pos;       /* initial editing position */
        union {
                struct {
                        Pint    unused;
                } pet_r1;
        } pets;
} Pstring_data;
```

**Execution**

INQUIRE STRING DEVICE STATE returns the current state of the specified string device, which is stored in the workstation state list of the workstation associated with the device. The current state includes the operating mode, echo switch, initial string, prompt/echo type, echo area, buffer length, edit position, and data record. See SET STRING MODE for a description of the operating mode and the echo switch, and how to set these values. See INITIALIZE STRING for a description of the initial string, prompt/echo type, echo area, and data record contents, and how to set these values.

In the event of an error, the *error indicator* indicates the error number of the error detected, and no other output data is returned. If no error is detected, the *error indicator* is set to zero, and the inquired information will be available in the output parameters. Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

2200 C : Buffer overflow in input or inquiry function

**SEE ALSO**

SET STRING MODE (3P)
INITIALIZE STRING (3P)
INQUIRE STRING DEVICE STATE 3 (3P)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

INQUIRE STRING DEVICE STATE 3 – inquire state of a STRING device

**SYNOPSIS**

**C Syntax**

**void**
**pinq_string_st3( ws, dev, store, err, op_mode, echo_switch, init_string, prompt_echo, echo_vol, string_data)**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **dev;** | *string device number* |
| **Pstore** | **store;** | *handle to Store object* |
| **Pint** | ***err;** | *OUT error indicator* |
| **Pop_mode** | ***op_mode** | *OUT operating mode* |
| **Pecho_switch** | ***echo_switch;** | *OUT echo switch* |
| **char** | ****init_string;** | *OUT initial string* |
| **Pint** | ***prompt_echo;** | *OUT prompt/echo type* |
| **Plimit3** | ***echo_vol;** | *OUT echo area* |
| **Pstring_data3** | ****string_data;** | *OUT data record* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE STRING DEVICE STATE 3 to retrieve the current state of the specified STRING device.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing data associated with the device state. This buffer is passed as the *store* argument.

The store buffer is a data area managed by PHIGS. While the application is responsible for creating the inital buffer through a call to STORE, PHIGS manages this area such that there is sufficient memory for the specific inquiry. The string data record within the store buffer is accessed by the pointer pointed to by *string_data*.

*ws* Workstation identifier. An integer specifying the workstation with which the specified STRING device is associated.

*dev* The device number of the STRING device. See the *Available Devices* section of INITIALIZE STRING for a description of the available devices.

*store* The memory buffer PHIGS is to use for storing the information returned. This buffer must exist prior to calling this function (see CREATE STORE (3P)).

**C Output Parameters**

*err* A pointer to the location to store the error number of any error detected by this function.

*op_mode*

The operating mode. Pop_mode is an enumerated type defined in phigs.h with the following values:

```
typedef enum {
        POP_REQ,
        POP_SAMPLE,
        POP_EVENT
} Pop_mode;
```

*echo_switch*

The echo state. Pecho_switch is an enumerated type defined in phigs.h with the following values:

```
typedef enum {
        PSWITCH_NO_ECHO,
        PSWITCH_ECHO
} Pecho_switch;
```

*init_string*

A pointer to a char pointer. PHIGS sets the location pointed to by *init_string* to point to the initial string stored in *store*.

*prompt_echo*

The prompt/echo type desired. See the *Available Devices* Section of INITIALIZE STRING for a description of the available types.

*echo_vol*

A pointer to a Plimit3 structure defining the *x*, *y*, and *z* components of the echo volume, in Device Coordinates. Plimit3 is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          x_min;          /* minimum x coordinate value */
        Pfloat          x_max;          /* maximum x coordinate value */
        Pfloat          y_min;          /* minimum y coordinate value */
        Pfloat          y_max;          /* maximum y coordinate value */
        Pfloat          z_min;          /* minimum z coordinate value */
        Pfloat          z_max;          /* maximum z coordinate value */
} Plimit3;
```

*string_data*

Pointer to a pointer that points to the string state within *store*. Pstring_data3 is defined in phigs.h as follows:

```
typedef struct {
        Pint    buffer_size;    /* input buffer size        */
        Pint    init_pos;       /* initial editing position */
        union {
                struct {
                        Pint    unused;
                } pet_r1;
        } pets;
} Pstring_data3;
```

**Execution**

INQUIRE STRING DEVICE STATE 3 returns the current state of the specified string device, which is stored in the workstation state list of the workstation associated with the device. The current state includes the operating mode, echo switch, initial string, prompt/echo type, echo volume, buffer length, edit position, and data record. See SET STRING MODE for a description of the operating mode and the echo switch, and how to set these values. See INITIALIZE STRING 3 for a description of the initial string, prompt/echo type, echo volume, and data record contents, and how to set these values.

In the event of an error, the *error indicator* indicates the error number of the error detected, and no other output data is returned. If no error is detected, the *error indicator* is set to zero, and the inquired information will be available in the output parameters. Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

**SEE ALSO**

SET STRING MODE (3P)
INITIALIZE STRING (3P)
INQUIRE STRING DEVICE STATE (3P)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

INQUIRE STROKE DEVICE STATE – inquire state of a stroke device

**SYNOPSIS**

**C Syntax**

**void**
**pinq_stroke_st( ws, dev, type, store, err, op_mode, echo_switch, init_view_ind, init_stroke, prompt_echo, echo_area, stroke_data)**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **dev;** | *stroke device number* |
| **Pinq_type** | **type;** | *type of returned value* |
| **Pstore** | **store;** | *handle to Store object* |
| **Pint** | ***err;** | *OUT error indicator* |
| **Pop_mode** | ***op_mode** | *OUT operating mode* |
| **Pecho_switch** | ***echo_switch;** | *OUT echo switch* |
| **Pint** | ***init_view_ind;** | *OUT initial view indicator* |
| **Ppoint_list** | ****init_stroke;** | *OUT initial stroke* |
| **Pint** | ***prompt_echo;** | *OUT prompt/echo type* |
| **Plimit** | ***echo_area;** | *OUT echo area* |
| **Pstroke_data** | ****stroke_data;** | *OUT data record* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE STROKE DEVICE STATE to retrieve the current state of the specified stroke device.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing data associated with the device state. This buffer is passed as the *store* argument.

The store buffer is a data area managed by PHIGS. While the application is responsible for creating the initial buffer through a call to CREATE STORE, PHIGS manages this area such that there is sufficient memory for the specific inquiry. The stroke device data record within the store buffer is accessed via the pointer pointed to by *stroke_data*.

*ws* Workstation identifier. An integer specifying the workstation with which the specified stroke device is associated.

*dev* The device number of the stroke device. See the *Available Devices* section of INITIALIZE STROKE for a description of the available devices.

*type* An enumerated value specifying whether the values to be returned are those originally specified by the application (PINQ_SET), or those resulting after PHIGS mapped them to ones available on the workstation (PINQ_REALIZED). A Pinq_type structure is defined as:

```
typedef enum {
        PINQ_SET,
        PINQ_REALIZED
} Pinq_type;
```

*store* The memory buffer PHIGS is to use for storing the information returned. This buffer must exist prior to calling this function (see CREATE STORE (3P)).

**C Output Parameters**

*err* A pointer to the location to store the error number of any error detected by this function.

*op_mode*

The operating mode. Pop_mode is an enumerated type defined in phigs.h with the following

values:

```
typedef enum {
        POP_REQ,
        POP_SAMPLE,
        POP_EVENT
} Pop_mode;
```

*echo_switch*
The echo state. Pecho_switch is an enumerated type defined in phigs.h with the following values:

```
typedef enum {
        PSWITCH_NO_ECHO,
        PSWITCH_ECHO
} Pecho_switch;
```

*init_view_ind*
Initial view indicator.

*init_stroke*
A pointer to the initial stroke. Ppoint_list is defined in phigs.h as follows:

```
typdef struct {
        Pint        num_points;     /* number of Ppoint3s in the list */
        Ppoint      *points;        /* list of points */
} Ppoint_list;
```

Ppoint is defined in phigs.h as follows:

```
typdef struct {
        Pfloat      x;              /* x coordinate */
        Pfloat      y;              /* x coordinate */
} Ppoint;
```

*prompt_echo*
The prompt/echo type desired. See the *Available Devices* Section of INITIALIZE STROKE for a description of the available types.

*echo_area*
A pointer to a variable of type Plimit that contains the echo area of the device. Plimit is defined in phigs.h as follows:

```
typedef struct {
        Pfloat      x_min;          /* x min */
        Pfloat      x_max;          /* x max */
        Pfloat      y_min;          /* y min */
        Pfloat      y_max;          /* y max */
} Plimit;
```

*stroke_data*

Pointer to a pointer that points to the stroke device state within *store*. Pstroke_data is defined in phigs.h as follows:

```
typedef struct {
        Pint    buffer_size;            /* input buffer size */
        Pint    init_pos;               /* initial editing position */
        Pfloat  x_interval;             /* x interval */
        Pfloat  y_interval;             /* y interval */
        Pfloat  time_interval;          /* time interval */
        union {
                struct {
                        Pint            unused;
                } pet_r1;
                struct {
                        Pint            unused;
                } pet_r2;
                struct {
                        Pmarker_attrs   marker_attrs;   /* marker attributes */
                } pet_r3;
                struct {
                        Pline_attrs     line_attrs;     /* line attributes */
                } pet_r4;
        } pets;
} Pstroke_data;
```

**Execution**

INQUIRE STROKE DEVICE STATE returns the current state of the specified stroke device, which is stored in the workstation state list of the workstation associated with the device. The current state includes the operating mode, echo switch, initial stroke value, prompt/echo type, echo area, and data record. See SET STROKE MODE for a description of the operating mode and the echo switch, and how to set these values. See INITIALIZE STROKE for a description of the initial stroke value, prompt/echo type, echo area, and data record contents, and how to set these values.

In the event of an error, the *error indicator* indicates the error number of the error detected, and no other output data is returned. If no error is detected, the *error indicator* is set to zero, and the inquired information will be available in the output parameters. Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

**SEE ALSO**

SET STROKE MODE (3P)
INITIALIZE STROKE (3P)
INQUIRE STROKE DEVICE STATE 3 (3P)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

INQUIRE STROKE DEVICE STATE 3– inquire state of a stroke device

**SYNOPSIS**

**C Syntax**

**void**
**pinq_stroke_st3( ws, dev, type, store, err, op_mode, echo_switch, init_view_ind, init_stroke, prompt_echo, echo_vol, stroke_data)**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **dev;** | *stroke device number* |
| **Pinq_type** | **type;** | *type of returned value* |
| **Pstore** | **store;** | *handle to Store object* |
| **Pint** | ***err;** | *OUT error indicator* |
| **Pop_mode** | ***op_mode** | *OUT operating mode* |
| **Pecho_switch** | ***echo_switch;** | *OUT echo switch* |
| **Pint** | ***init_view_ind;** | *OUT initial view indicator* |
| **Ppoint_list** | ****init_stroke;** | *OUT initial stroke* |
| **Pint** | ***prompt_echo;** | *OUT prompt/echo type* |
| **Plimit3** | ***echo_vol;** | *OUT echo area* |
| **Pstroke_data3** | ****stroke_data;** | *OUT data record* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE STROKE DEVICE STATE 3 to retrieve the current state of the specified stroke device.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing data associated with the device state. This buffer is passed as the *store* argument.

The store buffer is a data area managed by PHIGS. While the application is responsible for creating the initial buffer through a call to CREATE STORE, PHIGS manages this area such that there is sufficient memory for the specific inquiry. The stroke device data record within the store buffer is accessed via the pointer pointed to by *stroke_data*.

*ws* Workstation identifier. An integer specifying the workstation with which the specified stroke device is associated.

*dev* The device number of the stroke device. See the *Available Devices* section of INITIALIZE STROKE for a description of the available devices.

*type* An enumerated value specifying whether the values to be returned are those originally specified by the application (PINQ_SET), or those resulting after PHIGS mapped them to ones available on the workstation (PINQ_REALIZED). A Pinq_type structure is defined as:

```
typedef enum {
        PINQ_SET,
        PINQ_REALIZED
} Pinq_type;
```

*store* The memory buffer PHIGS is to use for storing the information returned. This buffer must exist prior to calling this function (see CREATE STORE (3P)).

**C Output Parameters**

*err* A pointer to the location to store the error number of any error detected by this function.

*op_mode*

The operating mode. Pop_mode is an enumerated type defined in phigs.h with the following

values:

```
typedef enum {
        POP_REQ,
        POP_SAMPLE,
        POP_EVENT
} Pop_mode;
```

*echo_switch*

The echo state. Pecho_switch is an enumerated type defined in phigs.h with the following values:

```
typedef enum {
        PSWITCH_NO_ECHO,
        PSWITCH_ECHO
} Pecho_switch;
```

*init_view_ind*

Initial view indicator

*init_stroke*

A pointer to the initial stroke. Ppoint_list3 is defined in phigs.h as follows:

```
typdef struct {
        Pint            num_points;     /* number of Ppoint3s in the list */
        Ppoint3         *points;        /* list of points */
} Ppoint_list3;
```

Ppoint3 is defined in phigs.h as follows:

```
typdef struct {
        Pfloat          x;              /* x coordinate */
        Pfloat          y;              /* x coordinate */
        Pfloat          z;              /* x coordinate */
} Ppoint3;
```

*prompt_echo*

The prompt/echo type desired. See the *Available Devices* Section of INITIALIZE STROKE for a description of the available types.

*echo_vol*

A pointer to a Plimit3 structure defining the *x*, *y*, and *z* components of the echo volume, in Device Coordinates. Plimit3 is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          x_min;          /* minimum x coordinate value */
        Pfloat          x_max;          /* maximum x coordinate value */
        Pfloat          y_min;          /* minimum y coordinate value */
        Pfloat          y_max;          /* maximum y coordinate value */
        Pfloat          z_min;          /* minimum z coordinate value */
        Pfloat          z_max;          /* maximum z coordinate value */
} Plimit3;
```

*stroke_data*

Pointer to a pointer that points to the stroke device state within *store*. Pstroke_data3 is defined in phigs.h as follows:

```
typedef struct {
        Pint        buffer_size;            /* input buffer size */
        Pint        init_pos;               /* initial editing position */
        Pfloat      x_interval;             /* x interval */
        Pfloat      y_interval;             /* y interval */
        Pfloat      z_interval;             /* z interval */
        Pfloat      time_interval;          /* time interval */
        union {
                struct {
                        Pint            unused;
                } pet_r1;
                struct {
                        Pint            unused;
                } pet_r2;
                struct {
                        Pmarker_attrs   marker_attrs;   /* marker attributes */
                } pet_r3;
                struct {
                        Pline_attrs     line_attrs;     /* line attributes */
                } pet_r4;
        } pets;
} Pstroke_data3;
```

**Execution**

INQUIRE STROKE DEVICE STATE 3 returns the current state of the specified stroke device, which is stored in the workstation state list of the workstation associated with the device. The current state includes the operating mode, echo switch, initial stroke value, prompt/echo type, echo volume, and data record. See SET STROKE MODE for a description of the operating mode and the echo switch, and how to set these values. See INITIALIZE STROKE 3 for a description of the initial stroke value, prompt/echo type, echo volume, and data record contents, and how to set these values.

In the event of an error, the *error indicator* indicates the error number of the error detected, and no other output data is returned. If no error is detected, the *error indicator* is set to zero, and the inquired information will be available in the output parameters. Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

**SEE ALSO**

SET STROKE MODE (3P)
INITIALIZE STROKE (3P)
INQUIRE STROKE DEVICE STATE (3P)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

INQUIRE STRUCTURE IDENTIFIERS– inquire the current structure identifiers

**SYNOPSIS**

**C Syntax**

**void**
**pinq_struct_ids ( length, start, error_ind, struct_ids, length_list )**

| | | |
|---|---|---|
| **Pint** | **length;** | *length of application list* |
| **Pint** | **start;** | *starting position* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pint_list** | ***struct_ids;** | *OUT list of structure ids* |
| **Pint** | ***length_list;** | *OUT length of list in PHIGS* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE STRUCTURE IDENTIFIERS to obtain a list of the current structure identifiers from the PHIGS state list.

**C Input Parameters**

*length* The number of *ints* items in the *struct_ids* output parameter for which the application has allocated memory. *length* is the number of list elements (structure identifiers) that the system can return in *struct_ids→ints*. If a value of 0 is used here, no data will be returned in the *struct_ids→ints* list, but the total number of structure identifiers in the PHIGS state list will be returned in *length_list*.

*start* Starting position of inquiry into the PHIGS state list of current structure identifiers. The elements of the list of structure identifiers, beginning with the item number specified by *start*, are copied sequentially into *struct_ids→ints* until *struct_ids→ints* is full or all the structure identifiers have been copied.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*struct_ids*

A pointer to a Pint_list structure in which the system returns the list of current structure identifiers. Pint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint    num_ints;       /* number of Pints in list */
        Pint    *ints;          /* list of integers */
} Pint_list;
```

The *num_ints* component specifies the number of structure identifiers in the list. The *ints* component is a pointer to a list, *num_ints* long, of the structure identifiers.

The application must allocate memory for *length* elements in the *struct_ids→ints* list prior to calling this procedure.

*length_list*

A pointer to an integer in which the system returns the total number of elements in the PHIGS state list of currently used structure identifiers. This is the value required for *length* if all structure identifiers are to be returned.

**ERRORS**

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

**SEE ALSO**

CHANGE STRUCTURE IDENTIFIER (3P)
INQUIRE STRUCTURE STATUS (3P)

**NAME**

INQUIRE STRUCTURE STATE VALUE – inquire current structure state value

**SYNOPSIS**

**C Syntax**

```
void
pinq_struct_st ( struct_state )
Pstruct_st          *struct_state;          OUT structure state
```

**Required PHIGS Operating States**

(*, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE STRUCTURE STATE VALUE to determine the current structure state. The state value may be ‘‘structure open’’ (STOP), or ‘‘structure closed’’ (STCL).

**C Output Parameters**

*struct_state*

A pointer to a Pstruct_st enumerated variable in which the current structure state value is returned. Values for Pstruct_st are defined in phigs.h as follows:

| | |
|---|---|
| PSTRUCT_ST_STCL | (structure closed) |
| PSTRUCT_ST_STOP | (structure open) |

**ERRORS**

No Error

**SEE ALSO**

OPEN STRUCTURE (3P)
CLOSE STRUCTURE (3P)
INQUIRE OPEN STRUCTURE (3P)
INQUIRE STRUCTURE IDENTIFIERS (3P)
INQUIRE STRUCTURE STATUS (3P)

**NAME**

INQUIRE STRUCTURE STATUS – inquire the status of a specified structure

**SYNOPSIS**

**C Syntax**

```
void
pinq_struct_status ( struct_id, error_ind, status )
Pint            struct_id;          structure identifier
Pint            *error_ind;         OUT error indicator
Pstruct_status  *struct;            OUT existence status
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE STRUCTURE STATUS to determine the existence status of the specified structure (NON_EXISTENT, EMPTY, or NOTEMPTY).

**C Input Parameters**

*struct_id*

Identifier of the structure for which the status is to be returned.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*struct* A pointer to a Pstruct_status enumerated variable in which the system returns the existence status of the specified structure. Values for Pstruct_status are defined in phigs.h as follows:

PSTRUCT_STATUS_NON_EXISTENT
PSTRUCT_STATUS_EMPTY
PSTRUCT_STATUS_NOT_EMPTY

**ERRORS**

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

**SEE ALSO**

DELETE STRUCTURE (3P)
EMPTY STRUCTURE (3P)
INQUIRE STRUCTURE IDENTIFIERS (3P)

**NAME**

INQUIRE SYSTEM STATE VALUE – inquire state value of PHIGS

**SYNOPSIS**

**C Syntax**

```
void
pinq_sys_st ( sys_state )
Psys_st          *sys_state;          OUT the system state
```

**Required PHIGS Operating States**

(*, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE SYSTEM STATE VALUE to determine the current state of the PHIGS system. The state value may be either *PHIGS open* (PHOP) or *PHIGS closed* (PHCL).

PHIGS must be opened with the OPEN PHIGS subroutine before any operations can be performed with the PHIGS subroutines. This function may be used to verify that OPEN PHIGS was successful before performing additional PHIGS operations.

**C Output Parameters**

*sys_state*

A pointer to a Psys_st enumerated variable in which the current state of the PHIGS system is returned. Values for Psys_st are defined in phigs.h as follows:

```
typdef enum {
        PSYS_ST_PHCL,     /* PHIGS closed */
        PSYS_ST_PHOP      /* PHIGS open */
} Psys_st;
```

**ERRORS**

None

**SEE ALSO**

OPEN PHIGS (3P)
CLOSE PHIGS (3P)
EMERGENCY CLOSE PHIGS (3P)
INQUIRE ARCHIVE STATE VALUE (3P)
INQUIRE STRUCTURE STATE VALUE (3P)
INQUIRE WORKSTATION STATE VALUE (3P)

**NAME**

INQUIRE TEXT EXTENT – determine the extent rectangle for a text string

**SYNOPSIS**

**C Syntax**

**void**
**pinq_text_extent ( wst, font, exp, sp, ht, path, hor, ver, str, error_ind, rect, offset )**

| | | |
|---|---|---|
| **Pint** | **wst;** | *workstation type* |
| **Pint** | **font;** | *text font* |
| **Pfloat** | **exp;** | *char expansion factor* |
| **Pfloat** | **sp;** | *char spacing* |
| **Pfloat** | **ht;** | *char height* |
| **Ptext_path** | **path;** | *text path* |
| **Phor_text_align** | **hor;** | *horizontal alignment* |
| **Pvert_text_align** | **ver;** | *vertical alignment* |
| **char** | ***str;** | *text string* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Prect** | ***rect;** | *OUT extent rectangle* |
| **Ppoint** | ***offset;** | *OUT concatenation offset* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

INQUIRE TEXT EXTENT calculates the extent (bounding rectangle) and concatenation offset for a specified text string and set of PHIGS text attributes.

**C Input Parameters**

*wst* The workstation type to use to resolve the font.

*font* The *text font*, specified as an index to the workstation's non-writable table of available fonts. All the font indices have named constants defined in phigs.h, as shown below.

The fonts available for PCS_ASCII are:

| Constant | Value |
|---|---|
| PFONT_MONO | 1 |

*exp* A real value specifying the character expansion factor.

*sp* The character spacing, specified as a real fraction of the font's nominal character height.

*ht* A real value specifying the character height.

*path* The text path is one of the following enumerated values:

```
typedef enum {
        PPATH_RIGHT,        /* Right */
        PPATH_LEFT,         /* Left */
        PPATH_UP,           /* Up */
        PPATH_DOWN          /* Down */
} Ptext_path;
```

*hor* The horizontal alignment. This is an enumerated value defined in phigs.h, and may be one of:

```
typedef enum {
        PHOR_NORM,      /* Normal */
        PHOR_LEFT,      /* Left */
        PHOR_CTR,       /* Center */
        PHOR_RIGHT      /* Right */
} Phor_text_align;
```

*ver* The vertical alignment. This is an enumerated value defined in phigs.h, and may be one of:

```
typedef enum {
        PVERT_NORM,     /* Normal */
        PVERT_TOP,      /* Top */
        PVERT_CAP,      /* Cap */
        PVERT_HALF,     /* Half */
        PVERT_BASE,     /* Base */
        PVERT_BOTTOM,   /* Bottom */
} Pvert_text_align;
```

*str* A pointer to the text string.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*rect* A pointer to a Prect structure in which to return the text extent. Prect is defined in phigs.h as:

```
typedef struct {
        Ppoint  p;      /* point p */
        Ppoint  q;      /* point q */
} Prect;
```

Ppoint is defined in phigs.h as:

```
typedef struct {
        Pfloat  x;      /* x coordinate */
        Pfloat  y;      /* y coordinate */
} Ppoint;
```

*offset* A pointer to a Ppoint structure in which to return the concatenation offset. See the *Execution* section below for its meaning.

**Execution**

The extent of the specified character string in the local 2D text coordinate system is computed using the specified text attributes for the specified workstation type. STROKE precision is assumed. The text position is (0,0) in the text local coordinate system. See TEXT 3 for a description of the text local coordinate system and how it is defined. The text extent encloses the entire text string with the specified attributes applied.

The concatenation offset indicates the text position for the concatenation of a subsequent text output primitive in the local 2D text coordinate system. This includes, for TEXT PATHS RIGHT and LEFT, an adjustment to account for the intercharacter spacing of the last character as specified by the character spacing parameter. It will be necessary for the application to apply a suitable modelling transformation to account for the CHARACTER UP VECTOR if it is other than the default.

If an error is detected by this function the *error indicator* will indicate the error number of the error detected and no other output data will be returned. If no error is detected, the *error indicator* will be set to zero and the inquired information will be available in the output parameters. Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

051 Ignoring function, this information is not yet available for this generic workstation type; open a workstation of this type and use the specific workstation type

052 Ignoring function, workstation type not recognized by the implementation type

062 Ignoring function, this information is not available for this MO workstation type

106 Ignoring function, the specified font is not available for the requested text precision on the specified workstation

**SEE ALSO**

TEXT (3P)
ANNOTATION TEXT RELATIVE (3P)

**NAME**

INQUIRE TEXT FACILITIES – inquire list of workstation text facilities

**SYNOPSIS**

**C Syntax**

**void**
**pinq_text_facs ( type, length, start, error_ind, facilities, total_length )**

| | | |
|---|---|---|
| **Pint** | **type;** | *workstation type* |
| **Pint** | **length;** | *length of application list* |
| **Pint** | **start;** | *starting position* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Ptext_facs** | ***facilities;** | *OUT text facilities* |
| **Pint** | ***total_length;** | *OUT length of list in PHIGS* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE TEXT FACILITIES to obtain a list of the text facilities available on the specified workstation type.

**C Input Parameters**

*type* Workstation type.

*length* The *length* is the number of text font and precision (fps) pairs which the *facilities* output parameter can return in the allocated memory space. A length of zero may be specified in order to have *total_length* return the total number of elements in the workstation list of available font and precision pairs.

*start* Starting position in the list of font and precision pairs in the workstation description table at which to begin the inquiry.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*facilities*

A pointer to a Ptext_facs data structure in which the system returns a listing of the text facilities available on this type of workstation. Ptext_facs is defined in phigs.h as:

```
typedef struct {
        Pint            num_font_precs;     /* number of fonts and precisions */
        Ptext_font_prec *font_precs;        /* list of fonts and precisions */
        Pint            num_char_hts;       /* number of character heights */
        Pfloat          min_char_ht;        /* minimum height */
        Pfloat          max_char_ht;        /* maximum height */
        Pint            num_char_expans;    /* number of character expansion factors */
        Pfloat          min_char_expan;     /* minimum expansion factor */
        Pfloat          max_char_expan;     /* maximum expansion factor */
        Pint            num_pred_inds;      /* number of predefined bundles */
} Ptext_facs;
```

Ptext_font_prec is defined in phigs.h as follows:

```
typedef struct {
        Pint            font;               /* text font */
        Ptext_prec      prec;               /* text precision */
} Ptext_font_prec;
```

Ptext_prec is an enumerated type with the following values:

```
typedef enum {
        PPREC_STRING,
        PPREC_CHAR,
        PPREC_STROKE
} Ptext_prec;
```

*total_length*

An integer pointer in which the system returns the total number of items in the list of available font and precision pairs on the workstation description table. This is the value required for *length* if all the items in the workstation list are to be returned.

The application must allocate memory for the text facilities list of fonts and precisions.

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the below error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

052 Ignoring function, workstation type not recognized by the implementation

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

**SEE ALSO**

INQUIRE PREDEFINED TEXT REPRESENTATION (3P)

**NAME**

INQUIRE TEXT REPRESENTATION – inquire a text representation on a workstation

**SYNOPSIS**

**C Syntax**

```
void
pinq_text_rep ( ws, index, type, error_ind, rep )
Pint            ws;             workstation identifier
Pint            index;          text index
Pinq_type       type;           type of returned value
Pint            *error_ind;     OUT error indicator
Ptext_bundle    *rep;           OUT text representation
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE TEXT REPRESENTATION to determine the current attribute values for a specified entry in a specified workstation's table of defined text representations.

**C Input Parameters**

*ws* Workstation identifier.

*index* Entry to be returned from the workstation's table of text representations; if this entry is not present in the table and the *type of returned value* parameter is REALIZED, the representation for text index 1 is returned.

*type* An enumerated value specifying whether the inquired values are to be returned as the values originally specified by the application (SET), or as the values actually being used by the workstation if any of the application-specified values had to be mapped to ones available on the workstation (REALIZED). Valid values are defined in phigs.h as:

```
typedef enum {
        PINQ_SET,       /* Return Application-specified Value */
        PINQ_REALIZED   /* Return Value Available on the Workstation */
} Pinq_type;
```

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*rep* A pointer to a Ptext_bundle structure in which the system returns the text representation at *index* in the workstation's table of text representations. Ptext_bundle is defined in phigs.h as follows:

```
typedef struct {
        Pint        font;       /* text font */
        Ptext_prec  prec;       /* text precision */
        Pfloat      char_expan; /* character expansion factor */
        Pfloat      char_space; /* character spacing */
        Pint        colr_ind;   /* text colour index */
} Ptext_bundle;
```

Ptext_prec is defined in phigs.h as:

```
typedef enum {
        PPREC_STRING,
        PPREC_CHAR,
        PPREC_STROKE
```

```
} Ptext_prec;
```

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the below error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (in other words, the workstation category is neither OUTPUT, OUTIN, nor MO)

100 Ignoring function, the bundle index value is less than one

101 Ignoring function, the specified representation has not been defined

**SEE ALSO**

INQUIRE PREDEFINED TEXT REPRESENTATION (3P)
SET TEXT REPRESENTATION (3P)
INQUIRE LIST OF TEXT INDICES (3P)
INQUIRE TEXT REPRESENTATION PLUS (3P+)

**NAME**

INQUIRE TEXT REPRESENTATION PLUS – inquire an extended text representation on a workstation

**SYNOPSIS**

**C Syntax**

```
void
pinq_text_rep_plus ( ws, index, type, error_ind, rep )
Pint              ws;           workstation identifier
Pint              index;        text index
Pinq_type         type;         type of returned value
Pint              *error_ind;   OUT error indicator
Ptext_bundle_plus *rep;         OUT extended text representation
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

INQUIRE TEXT REPRESENTATION PLUS returns the attribute values of a specified entry in a workstation extended text bundle table. See SET TEXT REPRESENTATION PLUS for a full description of the contents of the bundle table entry and a description of the extended text bundle table.

**C Input Parameters**

*ws* The workstation identifier.

*index* The text index.

*type* An enumerated value specifying whether the values to be returned are those originally specified by the application (PINQ_SET), or those resulting after PHIGS mapped them to ones available on the workstation (PINQ_REALIZED). A Pinq_type structure is defined in phigs.h as:

```
typedef enum {
        PINQ_SET,
        PINQ_REALIZED
} Pinq_type;
```

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*rep* A pointer to a Ptext_bundle_plus structure which returns the specified extended text representation. Ptext_bundle_plus is defined as:

```
typedef struct {
        Pint        font;       /* text font */
        Ptext_prec  prec;       /* text precision */
        Pfloat      char_expan; /* character expansion factor */
        Pfloat      char_space; /* character spacing */
        Pgcolr      colr;       /* text colour */
} Ptext_bundle_plus;
```

Ptext_prec is defined as:

```
typedef enum {
        PPREC_STRING,
        PPREC_CHAR,
        PPREC_STROKE
} Ptext_prec;
```

Pgcolr is defined as:

```
typedef struct {
        Pint                type;           /* indirect, RGB, CIE, HSV, HLS */
        union {
            Pint            ind;            /* index in workstation colour bundle table */
            struct {
                Pfloat      x;              /* red, hue, etc. */
                Pfloat      y;              /* green, saturation, lightness, etc. */
                Pfloat      z;              /* blue, value, saturation, etc. */
            } general;
        } val;
} Pgcolr;
```

Constants defined for colour type are:

| | | |
|---|---|---|
| 0 | PINDIRECT | *Indirect* |
| 1 | PMODEL_RGB | *Red, Green, Blue* |
| 2 | PMODEL_CIELUV | *CIE* |
| 3 | PMODEL_HSV | *Hue, Saturation, Value* |
| 4 | PMODEL_HLS | *Hue, Lightness, Saturation* |

The index member of the *val* union is used for type PINDIRECT. The general member is used for the other types.

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the specified text index is not present in the text bundle table on the workstation and the specified type of returned value is *Realized*, the representation for text index one is returned.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the following error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT, OUTIN, nor MO)

100 Ignoring function, the bundle index value is less than one

101 Ignoring function, the specified representation has not been defined

**SEE ALSO**

INQUIRE LIST OF TEXT INDICES (3P)
INQUIRE TEXT REPRESENTATION (3P)
INQUIRE PREDEFINED TEXT REPRESENTATION PLUS (3P+)
SET TEXT REPRESENTATION PLUS (3P+)

**NAME**

INQUIRE VALUATOR DEVICE STATE– inquire state of a valuator device

**SYNOPSIS**

**C Syntax**

**void**
**pinq_val_st ( ws, dev, store, err, op_mode, echo_switch, init_value, prompt_echo, echo_area, val_data)**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **dev;** | *valuator device number* |
| **Pstore** | **store;** | *handle to store object* |
| **Pint** | ***err;** | *OUT error indicator* |
| **Pop_mode** | ***op_mode;** | *OUT operating mode* |
| **Pecho_switch** | ***echo_switch;** | *OUT echo switch* |
| **Pfloat** | ***init_value;** | *OUT initial value* |
| **Pint** | ***prompt_echo;** | *OUT prompt/echo type* |
| **Plimit** | ***echo_area;** | *OUT echo area* |
| **Pval_data** | ****val_data;** | *OUT data record* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE VALUATOR DEVICE STATE to retrieve the current state of the specified valuator device.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing data associated with the device state. This buffer is passed as the *store* argument.

The store buffer is a data area managed by PHIGS. While the application is responsible for creating the initial buffer through a call to CREATE STORE, PHIGS manages this area such that there is sufficient memory for the specific inquiry. The valuator device data record within the store buffer is accessed via the pointer pointed to by *val_data*.

*ws* Workstation identifier. An integer specifying the workstation with which the specified valuator device is associated.

*dev* The device number of the valuator device. See the *Available Devices* section of INITIALIZE VALUATOR for a description of the available devices.

*store* The memory buffer PHIGS is to use for storing the information returned. This buffer must exist prior to calling this function (see CREATE STORE (3P)).

**C Output Parameters**

*err* A pointer to the location to store the error number of any error detected by this function.

*op_mode*

The operating mode. Pop_mode is an enumerated type defined in phigs.h with the following values:

```
typedef enum {
        POP_REQ,
        POP_SAMPLE,
        POP_EVENT
} Pop_mode;
```

*echo_switch*

The echo state. Pecho_switch is an enumerated type defined in phigs.h with the following values:

```
typedef enum {
        PSWITCH_NO_ECHO,
        PSWITCH_ECHO
} Pecho_switch;
```

*init_value*

Initial value.

*prompt_echo*

The prompt/echo type desired. See the *Available Devices* Section of INITIALIZE VALUATOR for a description of the available types.

*echo_area*

A pointer to a variable of type Plimit that contains the echo area of the device. Plimit is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          x_min;          /* x min */
        Pfloat          x_max;          /* x max */
        Pfloat          y_min;          /* y min */
        Pfloat          y_max;          /* y max */
} Plimit;
```

*val_data*

Pointer to a pointer that points to the valuator device state within *store*. Pval_data is defined in phigs.h as follows:

```
typedef struct {
        Pfloat  low;    /* low range limit */
        Pfloat  high;   /* high range limit */
        union {
                struct {
                        Pint    unused;
                } pet_r1;
                struct {
                        char    *label;
                        char    *format;
                        char    *low_label;
                        char    *high_label;
                } pet_u1;
        } pets;
} Pval_data;
```

**Execution**

INQUIRE VALUATOR DEVICE STATE returns the current state of the specified valuator device, which is stored in the workstation state list of the workstation associated with the device. The current state includes the operating mode, echo switch, initial value, prompt/echo type, echo area, and data record. See SET VALUATOR MODE for a description of the operating mode and the echo switch, and how to set these values. See INITIALIZE VALUATOR for a description of the initial valuator, prompt/echo type, echo area, and data record contents, and how to set these values.

In the event of an error, the *error indicator* indicates the error number of the error detected, and no other output data is returned. If no error is detected, the *error indicator* is set to zero, and the inquired information will be available in the output parameters. Since this is an inquiry function, ERROR HANDLING is not

invoked when an error is detected by this function.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

**SEE ALSO**

INITIALIZE VALUATOR (3P)
SET VALUATOR MODE (3P)
INQUIRE VALUATOR DEVICE STATE 3 (3P)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

INQUIRE VALUATOR DEVICE STATE 3 – inquire state of a valuator device

**SYNOPSIS**

**C Syntax**

**void**
**pinq_val_st3 ( ws, dev, store, err, op_mode, echo_switch, init_value, prompt_echo, echo_vol, val_data)**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **dev;** | *valuator device number* |
| **Pstore** | **store;** | *handle to store object* |
| **Pint** | ***err;** | *OUT error indicator* |
| **Pop_mode** | ***op_mode;** | *OUT operating mode* |
| **Pecho_switch** | ***echo_switch;** | *OUT echo switch* |
| **Pfloat** | ***init_value;** | *OUT initial value* |
| **Pint** | ***prompt_echo;** | *OUT prompt/echo type* |
| **Plimit3** | ***echo_vol;** | *OUT echo area* |
| **Pval_data3** | ****val_data;** | *OUT data record* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE VALUATOR DEVICE STATE 3 to retrieve the current state of the specified valuator device.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing data associated with the device state. This buffer is passed as the *store* argument.

The store buffer is a data area managed by the PHIGS. While the application is responsible for creating the initial buffer through a call to CREATE STORE, PHIGS manages this area such that there is sufficient memory for the specific inquiry. The valuator device data record within the store buffer is accessed via the pointer pointed to by *val_data*.

*ws* Workstation identifier. An integer specifying the workstation with which the specified valuator device is associated.

*dev* The device number of the valuator device. See the *Available Devices* section of INITIALIZE VALUATOR for a description of the available devices.

*store* The memory buffer PHIGS is to use for storing the information returned. This buffer must exist prior to calling this function (see CREATE STORE (3P)).

**C Output Parameters**

*err* A pointer to the location to store the error number of any error detected by this function.

*op_mode*

The operating mode. Pop_mode is an enumerated type defined in phigs.h with the following values:

```
typedef enum {
        POP_REQ,
        POP_SAMPLE,
        POP_EVENT
} Pop_mode;
```

*echo_switch*

The echo state. Pecho_switch is an enumerated type defined in phigs.h with the following values:

```
typedef enum {
        PSWITCH_NO_ECHO,
        PSWITCH_ECHO
} Pecho_switch;
```

*init_value*

Initial value.

*prompt_echo*

The prompt/echo type desired. See the *Available Devices* Section of INITIALIZE VALUATOR for a description of the available types.

*echo_vol*

A pointer to a Plimit3 structure defining the *x*, *y*, and *z* components of the echo volume, in Device Coordinates. Plimit3 is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          x_min;          /* minimum x coordinate value */
        Pfloat          x_max;          /* maximum x coordinate value */
        Pfloat          y_min;          /* minimum y coordinate value */
        Pfloat          y_max;          /* maximum y coordinate value */
        Pfloat          z_min;          /* minimum z coordinate value */
        Pfloat          z_max;          /* maximum z coordinate value */
} Plimit3;
```

*val_data*

Pointer to a pointer that points to the pick device state within *store*. Pval_data3 is defined in phigs.h as follows:

```
typedef struct {
        Pfloat  low;    /* low range limit */
        Pfloat  high;   /* high range limit */
        union {
                struct {
                        Pint    unused;
                } pet_r1;
                struct {
                        char    *label;
                        char    *format;
                        char    *low_label;
                        char    *high_label;
                } pet_u1;
        } pets;
} Pval_data3;
```

**Execution**

INQUIRE VALUATOR DEVICE STATE 3 returns the current state of the specified valuator device, which is stored in the workstation state list of the workstation associated with the device. The current state includes the operating mode, echo switch, initial value, prompt/echo type, echo volume, and data record. See SET VALUATOR MODE for a description of the operating mode and the echo switch, and how to set these values. See INITIALIZE VALUATOR 3 for a description of the initial valuator, prompt/echo type, echo volume, and data record contents, and how to set these values.

In the event of an error, the *error indicator* indicates the error number of the error detected, and no other output data is returned. If no error is detected, the *error indicator* is set to zero, and the inquired information will be available in the output parameters. Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

061 Ignoring function, specified workstation is not of category INPUT or OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

**SEE ALSO**

INITIALIZE VALUATOR 3 (3P)
SET VALUATOR MODE (3P)
INQUIRE VALUATOR DEVICE STATE (3P)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

INQUIRE VIEW FACILITIES – inquire the view facilities of a specified workstation type

**SYNOPSIS**

**C Syntax**

```
void
pinq_view_facs ( type, error_ind, num )
Pint        type;          workstation type
Pint        *error_ind;    OUT error indicator
Pint        *num;          OUT number of predefined view indices
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE VIEW FACILITIES to determine the number of predefined view representations supported by a specified workstation type.

**C Input Parameter**

*type* *Workstation type* being queried.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*num* An pointer to a location in which the system returns the number of predefined view indices for the workstation.

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the below error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

052 Ignoring function, workstation type not recognized by the implementation

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

057 Ignoring function, specified workstation is of category MI

062 Ignoring function, this information is not available for this MO workstation type

**SEE ALSO**

INQUIRE PREDEFINED VIEW REPRESENTATION (3P)

**NAME**

INQUIRE VIEW REPRESENTATION – inquire a view representation on a workstation

**SYNOPSIS**

**C Syntax**

```
void
pinq_view_rep ( ws, view_index, error_ind, update_state, cur_rep, req_rep )
Pint        ws;          workstation identifier
Pint        view_index;  view index
Pint        *error_ind;  OUT error indicator
Pupd_st     *update_state; OUT transformation update state
Pview_rep3  *cur_rep;    OUT current view representation
Pview_rep3  *req_rep;    OUT requested view representation
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE VIEW REPRESENTATION to obtain the current and requested view representation in an entry of the specified workstation's view table.

**C Input Parameters**

*ws* The *workstation identifier* of the workstation whose state list is queried.

*view_index*

The view index whose representation is desired from the workstation's table of defined view representations.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*update_state*

A pointer to a location in which the system returns the *view transformation update state*. Pupd_st is an enumerated type defined in phigs.h as follows:

```
typedef enum {
        PUPD_NOT_PEND,      /* Not Pending */
        PUPD_PEND           /* Pending */
} Pupd_st;
```

*cur_rep* A pointer to a Pview_rep3 structure in which the system returns the definition of the *current view representation*.

*req_rep* A pointer to a Pview_rep3 structure in which the system returns the definition of the *requested view representation*.

Pview_rep3 is defined in phigs.h as follows:

```
typedef struct {
    Pmatrix3        ori_matrix;     /* view orientation matrix */
    Pmatrix3        map_matrix;     /* mapping matrix */
    Plimit3         clip_limit;     /* clipping limits */
    Pclip_ind       xy_clip;        /* X-Y clipping indicator */
    Pclip_ind       back_clip;      /* back clipping indicator */
    Pclip_ind       front_clip;     /* front clipping indicator */
} Pview_rep3;
```

The Pmatrix3 type definition is a 4x4 matrix, defined in phigs.h as:

```
typedef Pfloat Pmatrix3[4][4];
```

The Plimit3 structure used to define the clipping limits is defined in phigs.h as follows:

```
typedef struct {
        Pfloat    x_min;    /* x min */
        Pfloat    x_max;    /* x max */
        Pfloat    y_min;    /* y min */
        Pfloat    y_max;    /* y max */
        Pfloat    z_min;    /* z min */
        Pfloat    z_max;    /* z max */
} Plimit3;
```

Pclip_ind is an enumerated type defined in phigs.h as follows:

```
typedef enum {
        PIND_CLIP,          /* Clipping */
        PIND_NO_CLIP        /* Not Clipping */
} Pclip_ind;
```

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the below error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

114 Ignoring function, the view index value is less than zero

101 Ignoring function, the specified representation has not been defined

**SEE ALSO**

INQUIRE PREDEFINED VIEW REPRESENTATION (3P)
SET VIEW INDEX (3P)
SET VIEW REPRESENTATION 3 (3P)
INQUIRE VIEW FACILITIES (3P)

**NAME**

INQUIRE WORKSTATION CATEGORY – inquire the category of a specified workstation type

**SYNOPSIS**

**C Syntax**

```
void
pinq_ws_cat ( type, error_ind, category )
Pint        type;          workstation type
Pint        *error_ind;    OUT error indicator
Pws_cat     *category;     OUT workstation category
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE WORKSTATION CATEGORY to determine the category of a specified workstation type.

**C Input Parameters**

*type* Type of workstation.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*category*

A pointer to a Pws_cat enumerated variable, in which the workstation category is returned. Values for Pws_cat are defined in phigs.h as follows:

```
typedef enum {
        PCAT_OUT,       /* Output Only */
        PCAT_IN,        /* Input Only */
        PCAT_OUTIN,     /* Output and Input */
        PCAT_MO,        /* Metafile Output */
        PCAT_MI         /* Metafile Input */
} Pws_cat;
```

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the below error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

052 Ignoring function, workstation type not recognized by the implementation

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

**SEE ALSO**

PHIGS WORKSTATION DESCRIPTION TABLE (7P)
OPEN WORKSTATION (3P)
INQUIRE WORKSTATION CLASSIFICATION (3P)

**NAME**

INQUIRE WORKSTATION CLASSIFICATION – inquire the classification of a specified workstation type

**SYNOPSIS**

**C Syntax**

**void**
**pinq_ws_class ( type, error_ind, class )**

| | | |
|---|---|---|
| **Pint** | **type;** | *workstation type* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pws_class** | ***class;** | *OUT workstation class* |

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE WORKSTATION CLASSIFICATION to determine the classification of a specified workstation type.

**C Input Parameters**

*type* Type of workstation.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*class* A pointer to a Pws_class enumerated variable in which the system returns the workstation class. Values for Pws_class are defined in phigs.h as follows:

```
typedef enum {
        PCLASS_VEC,     /* Vector Device */
        PCLASS_RASTER,  /* Raster Device */
        PCLASS_OTHER    /* Other Device */
} Pws_class;
```

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the below error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

052 Ignoring function, workstation type not recognized by the implementation

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

059 Ignoring function, the specified workstation does not have output capability (i.e., the workstation category is neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

**SEE ALSO**

PHIGS WORKSTATION DESCRIPTION TABLE (7P)
INQUIRE WORKSTATION CATEGORY (3P)
INQUIRE DISPLAY SPACE SIZE (3P)

**NAME**

INQUIRE WORKSTATION CONNECTION AND TYPE– inquire a workstation's connection identifier and type

**SYNOPSIS**

**C Syntax**

```
void
pinq_ws_conn_type ( ws, store, error_ind, conn_id, ws_type )
Pint            ws;             workstation identifier
Pstore store;   handle to Store object
Pint            *error_ind;     OUT error indicator
Pconnid *conn_id;  OUT connection identifier
Pint            *ws_type;       OUT workstation type
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE WORKSTATION CONNECTION AND TYPE to inquire the specific workstation type and connection identifier associated with a given workstation. The connection identifier and workstation type are associated with a workstation when it is opened. At that time a *specific workstation type* is created, based on the generic type supplied by the caller, and the specific workstation type and the user-supplied connection identifier are stored in the workstation's state list. This function returns that specific workstation type and the connection identifier. See OPEN WORKSTATION for descriptions of the possible connection identifiers and generic workstation types.

**C Input Parameters**

*ws* The workstation identifier of the workstation in question.

*string_length*
The size of the buffer (in bytes) pointed to by *connid* for the connection identifier to be returned in.

**C Output Parameters**

*error_ind*
A pointer to the location to store the error number of any error detected by this function.

*conn_id* A pointer to a Pconnid (a pointer to the connection identifier).

*ws_type* A pointer to a Pint variable in which to store the workstation type.

**Execution**

INQUIRE WORKSTATION CONNECTION AND TYPE returns the specific workstation type and connection identifier associated with the specified workstation. See OPEN WORKSTATION for a description of the possible connection identifiers and workstation types.

Except in the case mentioned in the C Parameters sections above, if an error is detected by this function the *error indicator* will indicate the error number of the error detected and no other output data will be returned. If no error is detected, the *error indicator* will be set to zero and the inquired information will be available in the output parameters. Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

2200 Buffer overflow in input or inquiry function

**SEE ALSO**

PHIGS WORKSTATION DESCRIPTION TABLE (7P)
INQUIRE LIST OF AVAILABLE WORKSTATION TYPES (3P)

OPEN WORKSTATION (3P)
WORKSTATION TYPE CREATE (3P)
WORKSTATION TYPE GET (3P)
WORKSTATION TYPE SET (3P)

**NAME**

INQUIRE WORKSTATION STATE TABLE LENGTHS – inquire the lengths of the workstation state tables for a specified workstation type

**SYNOPSIS**

**C Syntax**

```
void
pinq_ws_st_table ( type, error_ind, lengths )
Pint            type;           workstation type
Pint            *error_ind;     OUT error indicator
Pws_st_tables   *lengths;       OUT lengths of workstation tables
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE WORKSTATION STATE TABLE LENGTHS to determine the lengths of the state tables of a specified workstation type.

**C Input Parameters**

*type* Type of workstation.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*lengths* A pointer to a Pws_st_tables data structure which returns the length of the workstation state tables. Pws_st_tables is defined in phigs.h as:

```
typedef struct {
        Pint    line_bundles;   /* polyline tables */
        Pint    mark_bundles;   /* polymarker tables */
        Pint    text_bundles;   /* text tables */
        Pint    int_bundles;    /* interior tables */
        Pint    edge_bundles;   /* edge tables */
        Pint    pat_reps;       /* pattern tables */
        Pint    colr_reps;      /* colour tables */
        Pint    view_reps;      /* view tables */
} Pws_st_tables;
```

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the below error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

051 Ignoring function, this information is not yet available for this workstation type; open a workstation of this type and use the specific workstation type

052 Ignoring function, workstation type not recognized by the implementation

059 Ignoring function, the specified workstation does not have output capability (i.e., the workstation category is neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

**SEE ALSO**

PHIGS WORKSTATION DESCRIPTION TABLE (7P)
INQUIRE WORKSTATION STATE TABLE PLUS LENGTHS (3P+)

**NAME**

INQUIRE WORKSTATION STATE TABLE PLUS LENGTHS – inquire the lengths of the extended workstation state tables for a specified workstation type

**SYNOPSIS**

**C Syntax**

```
void
pinq_ws_st_table_plus ( type, error_ind, lengths )
Pint            type;           workstation type
Pint            *error_ind;     OUT error indicator
Pws_tables_plus *lengths;       OUT lengths of the workstation tables
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

INQUIRE WORKSTATION STATE TABLE PLUS LENGTHS inquires the lengths of the extended workstation state tables for a specified workstation type.

**C Input Parameters**

*type* The workstation type in question.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*lengths* A pointer to a Pws_tables_plus structure which returns the lengths of the workstation's tables. Pws_tables_plus is defined in phigs.h as follows:

```
typedef struct {
        Pint    line_bundles;   /* polyline tables */
        Pint    mark_bundles;   /* polymarker tables */
        Pint    text_bundles;   /* text tables */
        Pint    int_bundles;    /* interior tables */
        Pint    edge_bundles;   /* edge tables */
        Pint    pat_reps;       /* pattern tables */
        Pint    colr_reps;      /* colour tables */
        Pint    view_reps;      /* view tables */
        Pint    dcue_rep;       /* depth cue tables */
        Pint    light_src_rep;  /* light source tables */
        Pint    colr_map_rep;   /* colour mapping tables */
} Pws_tables_plus;
```

**Execution**

If the inquired information is available, the error indicator is returned as zero and values are returned in the output parameters.

If the inquired information is not available, the values returned in the output parameters are undefined and the error indicator is set to one of the below error numbers to indicate the reason for nonavailability.

Since this is an inquiry function, ERROR HANDLING is not invoked when an error is detected by this function.

**ERRORS**

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

051 Ignoring function, this information unavailable for this workstation type

052 Ignoring function, workstation type not recognized by the implementation

059 Ignoring function, the specified workstation does not have output capability (i.e. the workstation category is neither OUTPUT, OUTIN, nor MO)

062 Ignoring function, this information is not available for this MO workstation type

**SEE ALSO**

INQUIRE WORKSTATION STATE TABLE LENGTHS (3P)
PHIGS WORKSTATION DESCRIPTION TABLE (7P)

**NAME**

INQUIRE WORKSTATION STATE VALUE – inquire state value of workstation

**SYNOPSIS**

**C Syntax**

```
void
pinq_ws_st ( ws_state )
Pws_st          *ws_state;          OUT workstation state
```

**Required PHIGS Operating States**

(*, *, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE WORKSTATION STATE VALUE to determine the current state of the workstation. The state value can be either workstation open (PWS_ST_WSOP) or workstation closed (PWS_ST_WSCL).

This function can be used to verify that OPEN WORKSTATION was successful before performing additional PHIGS operations on the workstation.

**C Output Parameters**

*ws_state*

A pointer to an enumerated variable in which the current state of the workstation is returned. Pws_st is defined in phigs.h as follows:

```
typedef enum {
        PWS_ST_WSCL,    /* Workstation Closed */
        PWS_ST_WSOP     /* Workstation Open */
} Pws_st;
```

**ERRORS**

No Error

**SEE ALSO**

INQUIRE SET OF OPEN WORKSTATIONS (3P)
OPEN WORKSTATION (3P)
CLOSE WORKSTATION (3P)
INQUIRE SYSTEM STATE VALUE (3P)

**NAME**

INQUIRE WORKSTATION TRANSFORMATION– inquire a 2D workstation transformation

**SYNOPSIS**

**C Syntax**

**void**
**pinq_ws_tran ( ws, error_ind, upd_st, req_win_lim, cur_win_lim, req_vp_lim, cur_vp_lim )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pupd_st** | ***upd_st;** | *OUT update state* |
| **Plimit** | ***req_win_lim;** | *OUT requested workstation window* |
| **Plimit** | ***cur_win_lim;** | *OUT current workstation window* |
| **Plimit** | ***req_vp_lim;** | *OUT requested workstation viewport* |
| **Plimit** | ***cur_vp_lim;** | *OUT current workstation viewport* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE WORKSTATION TRANSFORMATION to determine the current and requested 2D workstation transformation from a specified workstation's state list.

**C Input Parameter**

*ws* The *workstation identifier* of the workstation whose state list is queried.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*upd_st* The current update state. Pupd_st is defined in phigs.h as follows:

```
typedef enum {
        PUPD_NOT_PEND
        PUPD_PEND
} Pupd_st;
```

*req_win_lim*

The requested workstation window. Plimit is defined in phigs.h as follows:

```
typedef struct {
        Pfloat    x_min;
        Pfloat    x_max;
        Pfloat    y_min;
        Pfloat    y_max;
} Plimit;
```

*cur_win_lim*

The current workstation window. Plimit is defined above.

*req_vp_lim*

The requested workstation viewport. Plimit is defined above.

*cur_vp_lim*

The current workstation viewport. Plimit is defined above.

**Execution**

The requested and current entries may differ if a workstation transformation change has been requested, but has not yet been provided. The update state will be *Pending* in this case, and *Not Pending* otherwise.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

057 Ignoring function, specified workstation is of category MI

**SEE ALSO**

SET WORKSTATION VIEWPORT (3P)
SET WORKSTATION WINDOW (3P)
INQUIRE WORKSTATION TRANSFORMATION 3 (3P)

**NAME**

INQUIRE WORKSTATION TRANSFORMATION 3 – inquire a 3D workstation transformation

**SYNOPSIS**

**C Syntax**

**void**
**pinq_ws_tran3 ( ws, error_ind, upd_st, req_win_lim, cur_win_lim, req_vp_lim, cur_vp_lim)**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | ***error_ind;** | *OUT error indicator* |
| **Pupd_st** | ***upd_st;** | *OUT update state* |
| **Plimit3** | ***req_win_lim;** | *OUT requested workstation window* |
| **Plimit3** | ***cur_win_lim;** | *OUT current workstation window* |
| **Plimit3** | ***req_vp_lim;** | *OUT requested workstation viewport* |
| **Plimit3** | ***cur_vp_lim;** | *OUT current workstation viewport* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use INQUIRE WORKSTATION TRANSFORMATION 3 to determine the current and requested 3D transformation from a specified workstation's state list.

**C Input Parameter**

*ws* The *workstation identifier* of the workstation whose state list is queried.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*upd_st* The current update state. Pupd_st is defined in phigs.h as follows:

```
typedef enum {
        PUPD_NOT_PEND
        PUPD_PEND
} Pupd_st;
```

*req_win_lim*

The requested workstation window. Plimit3 is defined in phigs.h as follows:

```
typedef struct {
        Pfloat    x_min;
        Pfloat    x_max;
        Pfloat    y_min;
        Pfloat    y_max;
        Pfloat    z_min;
        Pfloat    z_max;
} Plimit3;
```

*cur_win_lim*

The current workstation window. Plimit3 is defined above.

*req_vp_lim*

The requested workstation viewport. Plimit3 is defined above.

*cur_vp_lim*

The current workstation viewport. Plimit3 is defined above.

**Execution**

The requested and current entries may differ if a workstation transformation change has been requested, but has not yet been provided. The update state will be *Pending* in this case, and *Not Pending* otherwise.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

057 Ignoring function, specified workstation is of category MI

**SEE ALSO**

SET WORKSTATION VIEWPORT 3 (3P)
SET WORKSTATION WINDOW 3 (3P)
INQUIRE WORKSTATION TRANSFORMATION (3P)

**NAME**

LABEL – create a structure element containing a label identifier

**SYNOPSIS**

**C Syntax**

```
void
plabel ( label_id )
Pint            label_id;          label identifier
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

The LABEL subroutine puts a structure element containing a label identifier into the currently open structure. Labels are used to mark locations within a structure.

If the current edit mode is INSERT, the LABEL element is inserted into the currently open structure after the element currently pointed to by the element pointer. If the edit mode is REPLACE, the LABEL element replaces the element pointed to. In either case, the element pointer is updated to point to the new LABEL element.

**C Input Parameter**

*label_id* The label identifier is used by other subroutines to reference the label element's location in the structure.

**Execution**

The LABEL element is a marker within the structure. When the structure is traversed, LABEL elements are ignored.

LABEL elements are used by the SET ELEMENT POINTER AT LABEL and DELETE ELEMENTS BETWEEN LABELS subroutines to identify locations within a structure. The labels allow you to locate an element or group of elements even if structure editing changes the numbering of elements within the structure. For example, you may wish to bracket a sequence of related elements in a structure with labels.

Label identifiers need not be unique in a structure. The search for a label within a structure is always from the element after the current position of the element pointer toward the end of the structure. Subroutines which refer to labels reference the next occurrence of the specified label within the structure. If the specified label is not found before the end of the structure, an error is generated.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

SET ELEMENT POINTER AT LABEL (3P)
DELETE ELEMENTS BETWEEN LABELS (3P)
INQUIRE ELEMENT POINTER (3P)
ELEMENT SEARCH (3P)

**NAME**

MESSAGE – display a message on a workstation

**SYNOPSIS**

**C Syntax**

```
void
pmessage ( ws, msg )
Pint        ws;        workstation identifier
char        *msg;      message string
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

MESSAGE displays a message (character string) on the specified workstation. See *Execution* below for a description of the placement of the message on the workstation.

**C Input Parameters**

*ws* The workstation to send the message to.

*msg* The null terminated array of char (character string) to display on the workstation.

**Execution**

MESSAGE displays the specified character string on the specified workstation. The location of the message's display is dependent upon the workstation type as follows:

*x_tool* and *x_drawable* workstation types

If the workstation has a text subwindow associated with it, the message is written to that window. If a STRING input device is also active and using the text subwindow, the message will be interleaved (correctly) with the operator input.

If the workstation does not have a text subwindow associated with it, the message will be written to the lower left corner of the workstation's graphics window and will be removed upon the next clearing of that window. See UPDATE WORKSTATION and SET DISPLAY UPDATE STATE for a description of when the window is cleared. See WORKSTATION TYPE SET for a description of how to associate a text subwindow with a workstation.

The MESSAGE function only display a message when the PM is running. If the PM is not running then the function is silently ignored.

When the PM is running the message function causes a window to pop up on the display. The window contains the message and a button labelled ''Done''. Selecting the Done button causes the window to disappear.

The message window is implemented with Athena widgets. The resource database attributes recognized by the message window and the default values are:

| | |
|---|---|
| *message.box.orientation | vertical, |
| *message.box.allowResize | True, |
| *message*label.resize | True, |
| *message*button.label | Done |

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

**SEE ALSO**

WORKSTATION TYPE SET (3P)

**NAME**

NON-UNIFORM B-SPLINE CURVE – create a non-uniform B-spline curve structure element

**SYNOPSIS**

**C Syntax**

```
void
pnuni_bsp_curv (order, knots, rationality, cpoints, min, max)
Pint            order;          spline order
Pfloat_list     *knots;         list of knots
Prational       rationality;    rationality specifier
Ppoint_list34   *cpoints;       list of 3D or 4D control points
Pfloat          min, max;       parameter range
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

NON-UNIFORM B-SPLINE CURVE creates a structure element containing the definition of a non-uniform B-spline curve. The curve may be rational or non-rational.

**C Input Parameters**

*order* The order of the curve. The degree of the curve's basic functions is one less than *order*.

*knots* A pointer to a Pfloat_list structure containing a non-decreasing sequence of real numbers specifying the curve's knot vector. Pfloat_list is defined in phigs.h as follows:

```
typedef struct {
        Pint            num_floats;     /* number of Pfloats in list */
        Pfloat          *floats;        /* list of floats */
} Pfloat_list;
```

*rationality*

The rationality of the curve. Prational is defined in phigs.h as follows:

```
typedef enum {
        PNON_RATIONAL = 0,
        PRATIONAL = 1
} Prational;
```

*cpoints* A pointer to a Ppoint_list34 structure containing the list of the curve's control points. The points are specified in Modelling Coordinates. Ppoint_list34 is defined in phigs.h as follows:

```
typedef struct {                                /* list of 3D or 4D points */
        Pint                    num_points;     /* number of control points */
        union {
                Ppoint3         *point3d;       /* array of 3D points */
                Ppoint4         *point4d;       /* array of 4D points */
        } points;
} Ppoint_list34;
```

The point4d member of the points union is used if rationality is PRATIONAL; otherwise, the point3d member is used.

Ppoint3 is defined in phigs.h as follows:

```
typedef struct {
        Pfloat    x;    /* x coordinate */
        Pfloat    y;    /* y coordinate */
        Pfloat    z;    /* z coordinate */
} Ppoint3;
```

Ppoint4 is defined in phigs.h as follows:

```
typedef struct {
        Pfloat    x;    /* x coordinate */
        Pfloat    y;    /* y coordinate */
        Pfloat    z;    /* z coordinate */
        Pfloat    w;    /* w coordinate */
} Ppoint4;
```

*min, max*
The parameter limits specifying the range over which the curve is to be evaluated.

**Execution**

Depending on the edit mode, a NON-UNIFORM B-SPLINE CURVE element is either inserted into the open structure after the element pointer, or replaces the element pointed at by the element pointer. The element pointer is then updated to point to the NON-UNIFORM B-SPLINE CURVE structure element.

The spline order must be greater than zero. Curves of unsupported order will be displayed by drawing a polyline connecting the control points. Curves of order one are drawn as a polymarker using the current polymarker attributes.

The knots must form a non-decreasing sequence of numbers.

The rationality selector parameter may have the enumerated value *Rational* or *Non-rational*. When Rational is specified, the control points are specified as 4D homogeneous modelling coordinates. When Non-rational is specified, the control points are 3D PHIGS modelling coordinates.

The number of control points must be at least as large as the order. The number of control points plus the spline order must equal the number of knots.

At structure traversal time, a non-uniform B-spline curve primitive is drawn using the extended polyline attributes. See SET POLYLINE REPRESENTATION PLUS and SET CURVE APPROXIMATION CRITERIA for more information on display attributes.

**Attributes Applied**

The attributes listed below are used to display the NON-UNIFORM B-SPLINE CURVE primitive when the structure is traversed. The Aspect Source Flags (ASFs) tell where to access the output display attributes. These attributes can come directly from the traversal state list, or they can be accessed indirectly, using the appropriate index in the traversal state list and the corresponding bundled representation in the PHIGS workstation state list.

| | |
|---|---|
| polyline colour | polyline colour index ASF |
| linewidth scale factor | linewidth scale factor ASF |
| linetype | linetype ASF |
| polyline shading method | polyline shading method ASF |
| curve approximation criteria | curve approximation criteria ASF |
| polyline index | |
| depth cue index | |
| name set | |

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

600 Ignoring function, not enough control points for specified order

601 Ignoring function, knot sequence is not non-decreasing

602 Ignoring function, order is inconsistent with number of knots and control points specified

615 Ignoring function, parameter range is inconsistent with knots

**SEE ALSO**

ADD TRIMMING CURVE (3P+)
SET CURVE APPROXIMATION CRITERIA (3P+)
INQUIRE CURVE AND SURFACE FACILITIES (3P+)
SET POLYLINE REPRESENTATION PLUS (3P+)

**NAME**

NON-UNIFORM B-SPLINE SURFACE – create a non-uniform B-spline surface structure element

**SYNOPSIS**

**C Syntax**

**pnuni_bsp_surf ( uorder, vorder, uknots, vknots, rationality, grid, nloops, tloops )**

| | | |
|---|---|---|
| **Pint** | **uorder;** | *U spline order* |
| **Pint** | **vorder;** | *V spline order* |
| **Pfloat_list** | ***uknots;** | *U knots* |
| **Pfloat_list** | ***vknots;** | *V knots* |
| **Prational** | **rationality;** | *rationality selector* |
| **Ppoint_grid34** | ***grid;** | *grid of 3D or 4D control points* |
| **Pint** | **nloops;** | *number of trim curve loops* |
| **Ptrimcurve_list** | ***tloops;** | *trim curve loops* |

**Required PHIGS Operating States**

(PHOP, ∗, STOP, ∗)

**DESCRIPTION**

**Purpose**

NON-UNIFORM B-SPLINE SURFACE creates a structure element containing the definition of a non-uniform B-spline surface.

**C Input Parameters**

*uorder, vorder*

Order of the surface in the *u* and *v* dimensions, respectively.

*uknots, vknots*

Pointers to Pfloat_list structures listing the knots for the *u* and *v* dimensions of the surface. Pfloat_list is defined in phigs.h as follows:

```
typedef struct {
        Pint            num_floats;     /* number of Pfloats in list */
        Pfloat          *floats;        /* list of floats */
} Pfloat_list;
```

*rationality*

The rationality selector. Prational is defined in phigs.h as follows:

```
typedef enum {
        PNON_RATIONAL = 0,
        PRATIONAL = 1
} Prational;
```

*grid* Pointer to a Ppoint_grid34 structure containing the surface control points. Ppoint_grid34 is defined in phigs.h as follows:

```
typedef struct {                        /* grid of 3D or 4D points, [u_dim] [v_dim] */
   Ppcs_dims    num_points;             /* number of points in each dimension */
   union {
        Ppoint3 *point3d;               /* array of 3D points */
        Ppoint4 *point4d;               /* array of 4D points */
   } points;
} Ppoint_grid34;
```

Ppcs_dims is defined in phigs.h as:

```
typedef struct {
        Pint    u_dim;    /* dimension (number of divisions) along u */
        Pint    v_dim;    /* dimension (number of divisions) along v */
} Ppcs_dims;
```

Ppoint3 is defined in phigs.h as:

```
typedef struct {
        Pfloat    x;    /* x coordinate */
        Pfloat    y;    /* y coordinate */
        Pfloat    z;    /* z coordinate */
} Ppoint3;
```

Ppoint4 is defined in phigs.h as:

```
typedef struct {
        Pfloat    x;    /* x coordinate */
        Pfloat    y;    /* y coordinate */
        Pfloat    z;    /* z coordinate */
        Pfloat    w;    /* w coordinate */
} Ppoint4;
```

*nloops* Number of trimming curve loops.

*tloops* A pointer to an array of Ptrimcurve_list structures that specifies the trimming curve loops. Each of these structures contains the specification of an individual trimming curve loop (set of trimming curves). Ptrimcurve_list is defined in phigs.h as:

```
typedef struct {
        Pint            num_curves;    /* number of trimming curves in list */
        Ptrimcurve      *curves;       /* list of curves */
N
} Ptrimcurve_list;
```

Ptrimcurve is defined in phigs.h as:

```
typedef struct {
        Pedge_flag      visible;       /* curve visibility flag */
        Prational       rationality;   /* rationality */
        Pint            order;         /* curve order */
        Pint            approx_type    /* approximation type */
        Pfloat          approx_val     /* approximation value */
        Pfloat_list     knots;         /* curve knot vector */
        Pfloat          tmin, tmax;    /* curve parameter range */
        Ppoint_list23   cpts;          /* control points */
} Ptrimcurve;
```

Pedge_flag is defined in phigs.h as:

```
typedef enum {
        PEDGE_OFF,
```

```
        PEDGE_ON
} Pedge_flag;
```

Prational is defined in phigs.h as:

```
typedef enum {
        PNON_RATIONAL = 0,
        PRATIONAL = 1
} Prational;
```

Pfloat_list is defined in phigs.h as:

```
typedef struct {
        Pint    num_floats;     /* number of Pfloats in list */
        Pfloat  *floats;        /* list of floats */
} Pfloat_list;
```

Ppoint_list23 is defined in phigs.h as:

```
typedef struct {                                    /* list of 2D or 3D points */
        Pint                    num_points;         /* number of points */
        union {
                Ppoint          *point2d;           /* array of 2D points */
                Ppoint3         *point3d;           /* array of 3D points */
        } points;
} Ppoint_list23;
```

The *point3d* member of the points union is used if the rationality member of Ptrimcurve is PRATIONAL; otherwise, the *point2d* member is used. Ppoint is defined in phigs.h as:

```
typedef struct {
        Pfloat  x;      /* x coordinate */
        Pfloat  y;      /* y coordinate */
} Ppoint;
```

Ppoint3 is defined in phigs.h as:

```
typedef struct {
        Pfloat  x;      /* x coordinate */
        Pfloat  y;      /* y coordinate */
        Pfloat  z;      /* z coordinate */
} Ppoint3;
```

**Execution**

Depending on the edit mode, a NON-UNIFORM B-SPLINE SURFACE element is either inserted into the open structure after the element pointer, or replaces the element pointed at by the element pointer. The element pointer is then updated to point to this NON-UNIFORM B-SPLINE SURFACE structure element.

The surface *u* and *v* orders must be positive integers. Surfaces of unsupported orders in either dimension will be displayed by drawing the control grid; that is, the corresponding surface of order 2 in both dimensions.

The *u* and *v* knots must each form a non-decreasing sequence of numbers.

The *rationality selector* parameter may have the enumerated value *Rational* or *Non-rational*. When *Rational* is specified, the control points must be specified as 4D homogeneous modelling coordinates. When *Non-rational* is specified, the control points are 3D PHIGS modelling coordinates.

The number of control points in the *u* and *v* directions must be at least as large as the corresponding order. The number of control points in each direction plus the corresponding spline order must be equal to the corresponding number of knots.

At structure traversal time, a NON-UNIFORM B-SPLINE SURFACE primitive is drawn using the attributes that apply to FILL AREA SET 3 primitives, plus the extended interior attributes. See SET INTERIOR REPRESENTATION PLUS for a listing of these.

**Trimming Curves**

The surface parameter area to be rendered may be defined by trimming curves. The trimming definition of a surface consists of a list of loops. Each of these loops is a list of one or more trimming curves. Each trimming curve is a non-uniform rational B-spline curve defined in the 2D parameter space of the surface. The curves of a loop connect in a head-to-tail fashion. Each loop must be explicitly closed.

The restrictions on trimming curves are the same as those for non-uniform B-spline curves, and are specified in the function NON-UNIFORM B-SPLINE CURVE.

The trimming curves associated with a surface can touch each other only at their end points. No other pair of trimming curves can connect at that same point. A trimming curve cannot be self-touching or self-intersecting.

If trimming curves are specified, then they define the edges of the surface. Each trimming curve has a flag that controls its visibility. These visibility flags have lower precedence than the edge flag attribute and are ignored if the edge flag attribute is set to OFF.

PHIGS PLUS restricts trimming loops of a single surface primitive to a certain canonical form. This form must redundantly identify the interior parameter area in two different ways: according to the odd winding and the curve handedness rules.

**Attributes Applied**

The attributes listed below are used to display the NON-UNIFORM B-SPLINE SURFACE primitive when the structure is traversed. The Aspect Source Flags (ASFs) tell where to access the output display attributes. These attributes can come directly from the pipline context, or they can be accessed indirectly, using the appropriate index in the pipline context and the corresponding bundled representation in the PHIGS workstation state list.

| | |
|---|---|
| interior colour | interior colour index ASF |
| back interior colour | back interior colour ASF |
| interior style | interior style ASF |
| back interior style | back interior style ASF |
| interior style index | interior style index ASF |
| back interior style index | back interior style index ASF |
| interior shading method | interior shading method ASF |
| back interior shading method | back interior shading method ASF |
| interior reflectance equation | interior reflectance equation ASF |
| back interior reflectance equation | back interior reflectance equation ASF |
| area properties | area properties ASF |
| back area properties | back area properties ASF |
| interior index | |
| edge colour | edge colour index ASF |
| edge flag | edge flag ASF |
| edgetype | edgetype ASF |
| edgewidth scale factor | edgewidth scale factor ASF |

| | |
|---|---|
| surface approximation criteria | surface approximation criteria ASF |
| trimming curve approximation criteria | trimming curve approximation criteria ASF |
| edge index | |
| face distinguishing mode | |
| face culling mode | |
| depth cue index | |
| light source state | |
| name set | |

**ERRORS**

| | |
|---|---|
| 005 | Ignoring function, function requires state (PHOP, *, STOP, *) |
| 600 | Ignoring function, not enough control points for specified order |
| 601 | Ignoring function, knot sequence is not non-decreasing |
| 602 | Ignoring function, order is inconsistent with number of knots and control points specified |

**SEE ALSO**

SET SURFACE APPROXIMATION CRITERIA (3P+)
NON-UNIFORM B-SPLINE CURVE (3P+)
SET INTERIOR REPRESENTATION PLUS (3P+)
INQUIRE CURVE AND SURFACE FACILITIES (3P+)

**NAME**

OFFSET ELEMENT POINTER – move the element pointer in the open structure relative to its current position

**SYNOPSIS**

**C Syntax**

```
void
poffset_elem_ptr ( ep_offset )
Pint            ep_offset;          element pointer offset
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

OFFSET ELEMENT POINTER moves the element pointer in the currently open structure a specified number of elements from its current position.

**C Input Parameter**

*ep_offset*

Specifies the number of structure elements to move the element pointer from its current position.

**Execution**

OFFSET ELEMENT POINTER adds *element pointer offset* to the value of the element pointer in the open structure. The offset may be positive or negative. The pointer is set to point to the resulting element number.

If the resulting element number is less than 0, the element pointer is set to 0.

If the resulting element number is greater than the number of elements in the open structure, the element pointer is set to the last element in the structure.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

INQUIRE ELEMENT POINTER (3P)
INQUIRE CURRENT ELEMENT TYPE AND SIZE (3P)
ELEMENT SEARCH (3P)
SET ELEMENT POINTER (3P)
SET ELEMENT POINTER AT LABEL (3P)

**NAME**

OPEN ARCHIVE FILE – open specified archive file

**SYNOPSIS**

**C Syntax**

```
void
popen_ar_file ( archive_id, archive_file )
Pint            archive_id;          archive identifier
char            *archive_file;       archive file name
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use OPEN ARCHIVE FILE to open an archive file and associate a specified archive file identifier with the open archive file.

**C Input Parameters**

*archive_id*

An integer specifying the identifier to be associated with the open archive file.

*archive_file*

A character string giving the file name of the archive file. The application should not open this file; PHIGS will do this when it needs to access the file.

Neither the application nor the operator should attempt to interact with this file after passing it to OPEN ARCHIVE FILE.

**Execution**

If the specified archive file does not exist, then a new file is created and remains empty until CLOSE ARCHIVE FILE is called. At that time, any archived structures will be written to the file.

If the specified file exists and is a valid PEX-SI archive file, the contents are read and available for retrieval. If the named file exists but is empty, it is treated as if it did not exist (in other words, it becomes a valid PEX-SI archive file and remains empty until it is closed, at which time any archived structures are written to it).

The physical file is opened only by PHIGS when reading from or writing to it (in other words, during calls to OPEN ARCHIVE and CLOSE ARCHIVE).

The PHIGS archive state is set to *archive open* (AROP), and the specified archive file identifier is added to the *set of open archive files* maintained in the PHIGS state list.

**Note:** It is recommended that the application leave an archive file open until it is no longer needed, in order to avoid repeating the overhead associated with the OPEN ARCHIVE FILE operation.

One archive file format is supported: a private *binary* format. The binary format is supported for users who want compact archives over standard conformance.

**ERRORS**

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

402 Ignoring function, archive file identifier already in use

400 Ignoring function, the archive file cannot be opened

401 Ignoring function, opening this archive file would exceed the maximum number of simultaneously open archive files

403 Ignoring function, the archive file is not a PHIGS archive file

**SEE ALSO**

ARCHIVE STRUCTURES (3P)
CLOSE ARCHIVE FILE (3P)
INQUIRE ARCHIVE FILES (3P)
INQUIRE ARCHIVE STATE VALUE (3P)
RETRIEVE STRUCTURE IDENTIFIERS (3P)
RETRIEVE STRUCTURES (3P)
DELETE ALL STRUCTURES FROM ARCHIVE (3P)

**NAME**

OPEN PHIGS – open and initialize the PHIGS environment

**SYNOPSIS**

**C Syntax**

```
void
popen_phigs ( error_file, memory )
char            *error_file;        name of error file
size_t          memory;             NOT USED
```

**Required PHIGS Operating States**

(PHCL, WSCL, STCL, ARCL)

**DESCRIPTION**

**Purpose**

OPEN PHIGS initializes PHIGS and enables access to the PHIGS functions. OPEN PHIGS must be called prior to calling any other PHIGS functions.

**C Input Parameters**

*error_file*

A pointer to the *error file* to log PHIGS error messages to. The *error file* can be either a pointer to a valid UNIX file name or a null pointer, for example (char*)0. A null pointer implies that standard error is to be used as the error file. If a file name is specified, PHIGS will attempt to access the file for writing. If this attempt fails, OPEN PHIGS will fail and the appropriate error will be reported to standard error.

The error file argument passed to OPEN PHIGS will be passed to ERROR HANDLING. ERROR HANDLING will also pass this argument to ERROR LOGGING. If for some reason ERROR LOGGING cannot access the specified error file, the error message will be written to standard error. ERROR LOGGING appends messages to the error file; it does not truncate the file when OPEN PHIGS is called. If the specified file does not exist, it will only be created if ERROR LOGGING is called.

ERROR LOGGING writes the abstract PHIGS function name, the error number, and an error description to the error file. If for some reason the text for the function name and/or error description can't be determined, ERROR LOGGING will just write the function number and the error number.

*memory* This parameter is not used by the PEX-SI API.

**Execution**

OPEN PHIGS:

- Connects to a default or specified server. The default server is the one returned by XDisplayName (NULL). (OPEN XPHIGS must be used to specify a server.)
- Sets the PHIGS system state variable to 'PHOP'.
- Initializes the default workstation description tables and makes them available for use by the other PHIGS functions. These tables reflect the capabilities of the default or specified server.
- Creates the PHIGS state list and initializes it with default values taken from the PHIGS description table.
- Stores *error file* in the PHIGS error state list.

The default values for the PHIGS state list, PHIGS description table, and workstation description tables are in the Appendix.

When an error in PHIGS is detected, ERROR HANDLING is called and passes three items of informations: the *error file*, the function number of the PHIGS function that detected the error, and the error number. ERROR HANDLING calls ERROR LOGGING, which writes an error message to the error file. PHIGS users can replace ERROR HANDLING with a function of their own. This function may in turn call ERROR LOGGING. See ERROR HANDLING and ERROR LOGGING for more information.

PHIGS only writes to the error file if ERROR LOGGING is called, either by ERROR HANDLING or the application. If the *error file* does not exist when OPEN PHIGS is called, it will only be created if and when ERROR LOGGING is called.

**ERRORS**

001 Ignoring function, function requires state (PHCL, WSCL, STCL, ARCL)

450 Ignoring function, the specified error file is invalid

**SEE ALSO**

INQUIRE SYSTEM STATE VALUE (3P)
CLOSE PHIGS (3P)
OPEN ARCHIVE FILE (3P)
OPEN STRUCTURE (3P)
OPEN WORKSTATION (3P)
OPEN XPHIGS (3P)

**NAME**

OPEN STRUCTURE – create a new structure or begin editing an existing structure

**SYNOPSIS**

**C Syntax**

```
void
popen_struct ( struct_id )
Pint            struct_id;          structure identifier
```

**Required PHIGS Operating States**

(PHOP, *, STCL, *)

**DESCRIPTION**

**Purpose**

If the specified structure exists, OPEN STRUCTURE opens the structure for editing. If the specified structure does not exist, PHIGS creates a new structure and opens it for editing.

**C Input Parameter**

*struct_id*

The identifier of the structure to open or create.

**Execution**

If the specified structure does not exist, a new empty structure is created and opened, and the element pointer is set to 0. If the structure specified does exist, the structure is opened for editing and the element pointer is set to the last element in the structure.

A structure consists of a sequence of numbered structure elements. A structure element may contain:

- The specification for an output primitive
- Attribute selection
- View selection
- A modelling transformation
- A reference to another structure
- A name set or label
- A generalized structure element
- Application data

An element pointer in the open structure is used to reference specific structure elements. When a PHIGS function creates a new element, the current edit mode determines whether the new element is inserted after or replaces the element pointed to by the current element pointer. You can perform the following operations on the open structure:

- Add or delete structure elements
- Copy all elements from another structure
- Inquire about the current element type and size or the current element content
- Change the position of the element pointer
- Inquire the current position of the element pointer

The PHIGS state list stores the identifier for the currently open structure, the position of the current element pointer, the current edit mode, and a list of all structure identifiers in use.

**ERRORS**

006 Ignoring function, function requires state (PHOP, *, STCL, *)

**SEE ALSO**

OPEN PHIGS (3P)
INQUIRE OPEN STRUCTURE (3P)
CLOSE STRUCTURE (3P)

INQUIRE STRUCTURE STATE VALUE (3P)
INQUIRE STRUCTURE IDENTIFIERS (3P)

**NAME**

OPEN WORKSTATION – open a workstation

**SYNOPSIS**

**C Syntax**

```
void
popen_ws ( ws_id, conn_id, ws_type )
Pint            ws_id;          workstation identifier
Pconnid *conn_id;   connection identifier
Pint            ws_type;        workstation type
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

OPEN WORKSTATION opens a workstation of the specified workstation type. The workstation state list is created and initialized to conform as nearly as possible to the workstation description table associated with the specified workstation type. If the workstation is successfully opened, the PHIGS workstation state variable is set to WSOP.

PEX-SI supports two predefined workstation types, *x_tool* and *x_drawable*. Their characteristics are described below. In addition, the application can create and modify its own workstation types with the WORKSTATION TYPE CREATE and ORKSTATION TYPE SET functions.

If the workstation is opened successfully, a specific workstation type is created and associated with the open workstation. This specific workstation type contains the workstation description table that describes the capabilities of the opened workstation. The specific workstation type can be retrieved with INQUIRE WORKSTATION CONNECTION AND TYPE.

**C Input Parameters**

*ws_id* The workstation identifier to be associated with this workstation. This value is used to identify the workstation in subsequent PHIGS function calls.

*conn_id* A pointer to the connection identifier of the workstation. The type of value to use depends on the workstation type:

x_tool If the *conn_id* is NULL, a window is created on the default server. If the *conn_id* is not NULL, it is interpreted as a display name and a window for the workstation is created on the named server, provided that server supports the X extension. An example of using this would be:

```
Popen_ws (ws_id, (char*)"unix:0", phigs_ws_type_x_tool);
```

x_drawable

The connection identifier must be a pointer to a Pconnid_x_drawable structure, cast to a char*. Pconnid_x_drawable is defined in phigs.h as follows:

```
typdef struct {
        Display         *display;
        XID             drawable_id;
} Pconnid_x_drawable;
```

*ws_type* The type of workstation to open. Recognized types are described fully in the *Available Workstation Types* section below. They are declared in phigs.h. A short summary is listed here:

*phigs_ws_type_x_tool*

PHIGS creates an X window for the workstation on a specified or default server.

*phigs_ws_type_x_drawable*

PHIGS uses a specified X window for the workstation.

**Execution**

OPEN WORKSTATION opens a PHIGS workstation of the specified type and associates it with the specified workstation identifier.

When a workstation is opened, PHIGS creates a copy of the workstation type specified in the OPEN WORSTATION call and binds it to the opened workstation. This copy is called the *specific* workstation type. The workstation description table (WDT) of this specific type is checked against the capabilities of the server and window PHIGS is using for the workstation. If the capabilities specified by the WDT cannot be realized with that server and window, PHIGS modifies the WDT of the specific workstation type to reflect the available capabilities. The workstation type parameter to OPEN WORKSTATION is not modified; only the specific workstation type is (potentially) modified. The specific workstation type bound to a workstation can be retrieved with the PHIGS function INQUIRE WORKSTATION CONNECTION AND TYPE.

Some of a workstation type's workstation description table values can be changed by the application prior to opening a workstation of that type. See WORKSTATION TYPE CREATE and WORKSTATION TYPE SET for more information on this.

**Available Workstation Types**

*phigs_ws_type_x_tool*

PHIGS creates an X window on a specified or default server and uses it for the PHIGS workstation's display surface. If the PHIGS Monitor is running (see OPEN PHIGS), the default category of this workstation type is OUTIN, which indicates that both input and output are available; otherwise the default category is OUTPUT.

If the category is OUTIN, PHIGS creates an additional input-only window that it uses to detect pointer events for its Locator, Stroke, and Pick input devices. This window is transparent, overlies the output window completely, and duplicates any size and position changes made to the ouput window.

Many of the characteristics of an x_tool workstation type, such as its size and position, can be set prior to opening the workstation. See WORKSTATION TYPE CREATE and WORKSTATION TYPE SET for a complete list of the modifiable characteristics and their default settings.

*phigs_ws_type_x_drawable*

PHIGS uses an application-specified X window for the PHIGS workstation's display surface. The window to use is specified in the connection identifier parameter. The window must be open and associated with a server that supports the PEX extension.

If the PHIGS Monitor is running (see OPEN PHIGS), the default category of this workstation type is OUTIN, which indicates that both input and output are available; otherwise the default category is OUTPUT.

Some of the characteristics of an x_drawable workstation type, such as its size and position, can be set prior to opening the workstation. See WORKSTATION TYPE CREATE and WORKSTATION TYPE SET for a complete list of the modifiable characteristics and their default settings.

**Window System Interaction**

Workstation DC limits correspond to the window size used by the PHIGS workstation when the PHIGS workstation is opened. The units are drawable coordinates. When the API responds to a window resize event, more or less of the window will be exposed; the PHIGS output will not be scaled. Decreases in size cause portions of the PHIGS output to be clipped away if the new size is less than the PHIGS viewport limits. Size increase beyond the viewport limits will not reveal any additional PHIGS ouput.

PHIGS automatically redraws the PHIGS workstation when portions of it are exposed, such as when it is brought to the top of other windows or moved from an iconic state to an open state. This redrawing may potentially change portions of the workstation state list by making the ''state of visual representation'' correct and making all requested entries current.

For both tool and drawable workstations, a DestroyNotify event on the workstation's window will cause PHIGS to exit. A DestroyNotify event is generated either by the application or more likely by the operator when he destroys or ''Quit''s the window.

**ERRORS**

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

050 Ignoring function, connection identifier not recognized by the implementation

053 Ignoring function, workstation identifier already is in use

052 Ignoring function, workstation type not recognized by the implementation

055 Ignoring function, workstation cannot be opened for an implementation-dependent reason

063 Ignoring function, opening this workstation would exceed the maximum number of simultaneously open workstations

-168 Ignoring function, cannot open workstation on a read-only visual with no predefined colormap properties

-169 Ignoring function, cannot create an X colormap resource

-200 Ignoring function, cannot connect to the designated or default server

-201 Ignoring function, the specified or default X server does not support a compatible PEX extension

-202 Ignoring function, an X allocation error has occurred

-203 Ignoring function, can’t create an Xt shell

**SEE ALSO**

INQUIRE SET OF OPEN WORKSTATIONS (3P)
OPEN PHIGS (3P)
CLOSE WORKSTATION (3P)
INQUIRE WORKSTATION CONNECTION AND TYPE (3P)
INQUIRE WORKSTATION STATE VALUE (3P)
WORKSTATION TYPE CREATE (3P)
WORKSTATION TYPE SET (3P)

**NAME**

OPEN XPHIGS – open and initialize PHIGS in the X environment

**SYNOPSIS**

**C Syntax**

```
void
popen_xphigs ( error_file, memory, xinfo_mask, xinfo )
char            *error_file;        name of error file
Plong           memory;             /fInot used/fP
unsigned long   xinfo_mask;
Pxphigs_info    *xinfo;
```

**Required PHIGS Operating States**

(PHCL, WSCL, STCL, ARCL)

**DESCRIPTION**

**Purpose**

OPEN XPHIGS is similar to OPEN PHIGS but allows specification of additional run-time options. It initializes the API and enables access to the PHIGS functions. OPEN PHIGS or OPEN XPHIGS must be called prior to calling any other PHIGS functions.

**C Input Parameters**

*error_file*

A pointer to the *error file* to log PHIGS error messages to. The *error file* can be either a pointer to a valid UNIX file name or a null pointer, for example (char*)0. A null pointer implies that standard error is to be used as the error file. If a file name is specified, PHIGS will attempt to access the file for writing. If this attempt fails, OPEN XPHIGS will fail and the appropriate error will be reported to standard error.

The error file argument passed to OPEN XPHIGS will be passed to ERROR HANDLING. ERROR HANDLING will also pass this argument to ERROR LOGGING. If for some reason ERROR LOGGING cannot access the specified error file, the error message will be written to standard error. ERROR LOGGING appends messages to the error file; it does not truncate the file when OPEN XPHIGS is called. If the specified file does not exist, it will only be created if ERROR LOGGING is called.

ERROR LOGGING writes the abstract PHIGS function name, the error number, and an error description to the error file. If for some reason the text for the function name and/or error description can't be determined, ERROR LOGGING will just write the function number and the error number.

*memory* This parameter is not used by the PEX-SI API.

*xinfo_mask*

A bitmask indicating which of the options are being set. This mask is a bitwise OR of one or more of the valid option mask bits and indicates which fields of the Pxphigs_info structure are set.

| | |
|---|---|
| PXPHIGS_INFO_DISPLAY | The display pointer: *display* |
| PXPHIGS_INFO_RMDB | The resource manager: *rmdb* |
| PXPHIGS_INFO_APPL_ID | The application name and class: *appl_id* |
| PXPHIGS_INFO_ARGS | The command line arguments: *args* |
| PXPHIGS_INFO_FLAGS_NO_MON | The No Monitor flag: *flags.no_monitor* |
| PXPHIGS_INFO_FLAGS_CLIENT_SS | The Force Client SS flag: *flags.force_client_SS* |

*xinfo* A pointer to a Pxphigs_info structure. This structure is used to specify X-related options to PHIGS. *xinfo* is defined in phigs.h as follows:

```
typedef struct {
        Display                 *display;           /* valid display pointer */
        XrmDatabase             rmdb;               /* a valid database */
        struct {
                char            *name;
                char            *class;
        } appl_id;                                  /* for resolving database attributes */
        struct {
                int             *argc_p;
                char            **argv;
        } args;                                     /* for merging args into specified database */
        struct {
                unsigned        no_monitor: 1;      /* 1 ==> monitor will not be executed */
                unsigned        force_client_SS;    /* 1 ==> always use client-side CSS */
        } flags;
} Pxphigs_info;
```

Only the fields indicated by *xinfo_mask* are examined.

*display* specifies both the PHIGS default server and the connection PHIGS is to use when communicating with it. The PHIGS default server holds the master copy of the central structure store, if it supports PEX structure storage, and is where *tool* workstations are opened if their location is not specified to the OPEN WORKSTATION function. PHIGS uses the specified connection for all communication to the default server, even if a different connection is specified for a *drawable* workstation in a subsequent call to OPEN WORKSTATION. PHIGS uses the specified connection for the duration of the PHIGS session; it must not be closed before calling CLOSE PHIGS. If *display* is not specified, the default server will be the one returned by a call to XDisplayName() with an argument of NULL.

*rmdb* an X resource database handle. PHIGS uses this database to build the default workstation description tables. Database searches are on the resource names and classes listed below. Each search is qualified by the name and class specified in *appl_id*. If *args* are also specified PHIGS will merge them into the database prior to searching the database for resources PHIGS recognizes. Any merged arguments will be removed from the argument list. The resources PHIGS recognizes are:

| ***Argument String*** | ***Resource Name*** | ***Resource Class*** | ***Type*** | ***Valid Values*** |
|---|---|---|---|---|
| -display | display | Display | String | |
| -bufmode | bufMode | BufMode | String | single \| double |
| = | geometry | Geometry | String | |
| -label | label | Label | String | |
| -iconlabel | iconLabel | IconLabel | String | |

The *display* resource specifies the name of the default server. The *bufMode* resource specifies the default buffering mode, ‘‘single’’ or ‘‘double’’. The *geometry* resource specifies the default window location and size for *phig_ws_type_x_tool* workstation types. The geometry is specified in the standard X geometry format:

<width>x<height>{+-}<xoffset>{+-}<yoffset>.

The *label* resource specifies the window label for *phig_ws_type_x_tool* workstation types. The *iconLabel* resource specifies the icon label for *phig_ws_type_x_tool* workstation types.

*appl_id.name* and *appl_id.class* are the application name and class to use when resolving resource database attributes. If not specified, the name ‘‘phigs’’ and class ‘‘Phigs’’ are used.

*args.argc_p* is a pointer to the argument count, *args.argv* is the array of command line arguments. The arguments are searched for attributes recognized by PHIGS and are merged into the specified

database, if any.

*flags.no_monitor* indicates whether the PHIGS Monitor (PM) is executed during popenphigs(). The PM is a separate program started by PHIGS that handles window events and PHIGS input for PHIGS workstations. PHIGS has complete control over this program; no user action is required to deal with it other than to indicate if it should be used or not. If the PM is not executed (flags.no_monitor = 1), PHIGS will not monitor and respond to X window events nor will it provide any PHIGS workstations that support PHIGS input devices. All the predefined workstation types will be of category OUTPUT (output only). If the PM is executed (flags.no_monitor = 0), PHIGS will monitor relevant window events and will provide predefined workstation types of category OUTIN (input and output).

*flags.force_client_SS* indicates whether the API should use client side structure storage even if server side structure storage is available in X servers with the PEX extension. *flags.force_client_SS* = 1 indicates that client side structure storage should always be used. *flags.force_client_SS* = 0 indicates that server side structure storage should be used if available.

**Execution**

OPEN XPHIGS sets internal state information and then calls popen_phigs().

**ERRORS**

| | |
|---|---|
| 001 | Ignoring function, function requires state (PHCL, WSCL, STCL, ARCL) |
| 450 | Ignoring function, the specified error file is invalid |
| -200 | Ignoring function, cannot connect to the designated or default server |
| -201 | Ignoring function, the specified or default X server does not support a compatible PEX extension |

**SEE ALSO**

INQUIRE SYSTEM STATE VALUE (3P)
OPEN PHIGS (3P)
CLOSE PHIGS (3P)

**NAME**

POLYLINE – create structure element specifying 2D polyline

**SYNOPSIS**

**C Syntax**

```
void
ppolyline ( point_list )
Ppoint_list        *point_list;        list of points
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

The POLYLINE function places a structure element containing the full specification of a 2D polyline into the currently open structure, according to the current edit mode. A 2D polyline primitive is a set of connected straight lines in two dimensions, *x* and *y*. The *z* coordinate is assumed to be 0. The polyline is defined by a series of points in Modelling Coordinates (MC). A polyline may be closed or self-intersecting.

**C Input Parameters**

*point_list*

A pointer to a Ppoint_list structure containing a list of Ppoint structures, which contain the *x* and *y* coordinates for each point. At least two points must be specified; a polyline structure element that has fewer than two points will be treated in a workstation-dependent fashion.

The Ppoint_list structure is defined in phigs.h as follows:

```
typedef struct  {
        Pint    num_points;     /* number of Ppoints in the list */
        Ppoint  *points;        /* list of points */
} Ppoint_list;
```

The *num_points* component specifies the number of elements in the list. The *points* component is a pointer to a list of Ppoints *num_points* long. The Ppoint structure is defined in phigs.h as follows:

```
typedef struct  {
        Pfloat  x;   /* x coordinate */
        Pfloat  y;   /* y coordinate */
} Ppoint;
```

**Execution**

If the current edit mode is INSERT, the POLYLINE element is inserted into the open structure after the element pointed to by the element pointer. If the edit mode is REPLACE, the POLYLINE element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new POLYLINE element.

When the structure is traversed, the polyline element draws straight lines between the points specified, beginning with the first point and ending with the last. The current values of the attributes listed below will be applied to the polyline when it is drawn.

The coordinates used to specify the polyline primitive are MC. These may be any coordinate units that are convenient to the application. At traversal, these coordinate values are transformed by the current local and global modelling transformations, the view representation selected by the *current view index*, and the workstation transformation current on the workstation to which the structure is posted.

**Attributes Applied**

The attributes listed below are used to display the POLYLINE primitive when the structure is traversed. The Aspect Source Flags (ASFs) tell where to access the output display attributes. These attributes can come directly from the traversal state list, or they can be accessed indirectly, using the appropriate index in the

traversal state list and the corresponding bundled representation in the PHIGS workstation state list.

| | |
|---|---|
| polyline colour | polylinecolour index ASF |
| linewidth scale factor | linewidth scale factor ASF |
| linetype | linetype ASF |
| polyline shading method | polyline shading method ASF |
| polyline index | |
| depth cue index | |
| name set | |

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

INQUIRE POLYLINE FACILITIES (3P)
POLYLINE 3 (3P)
POLYLINE SET 3 WITH DATA (3P+)

**NAME**

POLYLINE 3 – create structure element specifying 3D polyline

**SYNOPSIS**

**C Syntax**

```
void
ppolyline3 ( point_list )
Ppoint_list3        *point_list;        list of points
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

The POLYLINE 3 function places a structure element containing the full specification of a 3D polyline into the currently open structure, according to the current edit mode. A 3D polyline primitive is a set of connected straight lines in 3D space defined by a series of Modelling Coordinate (MC) points. A polyline may be closed or self-intersecting.

**C Input Parameters**

*point_list*

A pointer to a Ppoint_list3 structure containing a list of Ppoint3 structures, which contain the *x*, *y*, and *z* coordinates for each point. At least two points must be specified; a polyline structure element that has fewer than two points will be treated in a workstation dependent fashion.

The Ppoint_list3 structure is defined in phigs.h as follows:

```
typedef struct  {
        Pint      num_points;    /* number of Ppoint3s in the list */
        Ppoint3   *points;       /* list of points */
} Ppoint_list3;
```

The *num_points* component specifies the number of elements in the list. The *points* component is a pointer to a list of Ppoint3s *num_points* long. The Ppoint3 structure is defined in phigs.h as follows:

```
typedef struct  {
        Pfloat    x;    /* x coordinate */
        Pfloat    y;    /* y coordinate */
        Pfloat    z;    /* z coordinate */
} Ppoint3;
```

**Execution**

If the current edit mode is INSERT, the POLYLINE 3 element is inserted into the open structure after the element pointed to by the element pointer. If the current edit mode is REPLACE, the POLYLINE 3 element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new POLYLINE 3 element.

When the structure is traversed, the polyline element draws straight lines between the points specified, beginning with the first point and ending with the last. The current values of the attributes listed below will be applied to the polyline when it is drawn.

The coordinates used to specify the polyline primitive are called Model Coordinates (MC). Model Coordinates can be any coordinate units that are convenient to the application. At traversal, these coordinate values are transformed by the current local and global modelling transformations, the view representation selected by the *current view index*, and the workstation transformation current on the workstation to which the structure is posted.

**Attributes Applied**

The attributes listed below are used to display the POLYLINE 3 primitive when the structure is traversed. The Aspect Source Flags (ASFs) tell where to access the output display attributes. These attributes can come directly from the traversal state list, or they can be accessed indirectly, using the appropriate index in the traversal state list and the corresponding bundled representation in the PHIGS workstation state list.

| | |
|---|---|
| polyline colour | polylinecolour index ASF |
| linewidth scale factor | linewidth scale factor ASF |
| linetype | linetype ASF |
| polyline shading method | polyline shading method ASF |
| polyline index | |
| depth cue index | |
| name set | |

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

INQUIRE POLYLINE FACILITIES (3P)
POLYLINE (3P)
POLYLINE SET 3 WITH DATA (3P+)

**NAME**

POLYLINE SET 3 WITH DATA – create a 3D polyline set structure element that includes colour and shading data

**SYNOPSIS**

**C Syntax**

**void**
**ppolyline_set3_data ( vflag, colour_type, npl, vdata )**

| | | |
|---|---|---|
| **Pint** | **vflag;** | *data per vertex flag* |
| **Pint** | **colour_type;** | *colour type* |
| **Pint** | **npl;** | *number of polylines in the set* |
| **Pline_vdata_list3** | ***vdata;** | *per line vertex data list* |

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

POLYLINE SET 3 WITH DATA creates a 3D polyline set primitive with colour and shading data.

A 3D polyline with data primitive is a set of connected lines in a 3D space defined by a series of Modelling Coordinate points. The element can optionally specify colour information for each vertex of the primitive.

**C Input Parameters**

*vflag* The data per vertex flag specifies the available data for each vertex of the primitive.

| | | |
|---|---|---|
| 0 | PVERT_COORD | *Coordinates Specified* |
| 1 | PVERT_COORD_COLOUR | *Coordinates and Colours Specified* |

*colour_type*

The colour type for specified vertex colours.

| | | |
|---|---|---|
| 0 | PINDIRECT | *Colour Index Specified* |
| 1 | PMODEL_RGB | *Red, Green, and Blue* |
| 2 | PMODEL_CIELUV | *CIE Colour Model* |
| 3 | PMODEL_HSV | *Hue, Saturation, and Value* |
| 4 | PMODEL_HLS | *Hue, Lightness, and Saturation* |

*npl* The number of polylines in the set.

*vdata* A pointer to an array of *npl* Pline_vdata_list3 structures that specifies the list of vertices and optionally associated colour information. Pline_vdata_list3 is defined in phigs.h as follows:

```
typedef struct {
        Pint            num_vertices;   /* number of vertices */
        Pline_vdata_arr3 vertex_data;   /* line vertex data */
} Pline_vdata_list3;
```

Pline_vdata_arr3 is defined in phigs.h as follows:

```
typedef union {
        Ppoint3   *points;      /* array of points */
        Pptco3    *ptcolrs;     /* array of points with colours */
        /* implementation-dependent data */
}Pline_vdata_arr3;
```

Ppoint3 is defined in phigs.h as follows:

```
typedef struct {
        Pfloat    x;    /* x coordinate */
        Pfloat    y;    /* y coordinate */
        Pfloat    z;    /* z coordinate */
} Ppoint3;
```

Pptco3 is defined in phigs.h as follows:

```
typedef struct {
        Ppoint3   point;    /* point coordinates */
        Pcoval    colr;     /* colour */
} Pptco3;
```

Ppoint3 is defined above. Pcoval is defined in phigs.h as follows:

```
typedef union {
        Pint          ind;      /* index in workstation colour bundle table */
        Pcolr_rep     direct;   /* direct colour components */
} Pcoval;
```

When *colour_model* is PINDIRECT, *ind* is used; otherwise, *direct* is used.

Pcolr_rep is defined in phigs.h as follows:

```
typedef union {
        Prgb          rgb;          /* Red Green Blue colour specification */
        Pcieluv       cieluv;       /* CIE L*U*V* colour specification */
        Phls          hls;          /* Hue Lightness Saturation colour specification */
        Phsv          hsv;          /* Hue Saturation Value colour specification */
        Pdata         unsupp;       /* Colour in unsupported colour model */
} Pcolr_rep;
```

Prgb is defined in phigs.h as follows:

```
typedef struct {
        Pfloat        red;          /* red, hue, etc */
        Pfloat        green;        /* green, saturation, lightness, etc */
        Pfloat        blue;         /* blue, value, saturation, etc */
} Prgb;
```

Pcieluv is defined in phigs.h as follows:

```
typedef struct {
        Pfloat        cieluv_x;     /* x coefficient */
        Pfloat        cieluv_y;     /* y coefficient */
        Pfloat        cieluv_y_lum; /* y luminance */
} Pcieluv;
```

Phls is defined in phigs.h as follows:

```
typedef struct {
        Pfloat        hue;          /* hue */
        Pfloat        lightness;    /* lightness */
        Pfloat        satur;        /* saturation */
} Phls;
```

Phsv is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          hue;            /* hue */
        Pfloat          satur;          /* saturation */
        Pfloat          value;          /* value */
} Phsv;
```

Pdata is defined in phigs.h as follows:

```
typedef struct {
        size_t          size;           /* size of data */
        char            *data           /* pointer to data */
} Pdata;
```

**Execution**

Depending on the edit mode, a POLYLINE SET 3 WITH DATA element is inserted into the open structure after the element pointer, or replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the newly created POLYLINE SET 3 WITH DATA element.

When the structure is traversed, POLYLINE SET 3 WITH DATA generates a set of unconnected polylines defined by a set of points. Each point sequence in the set generates connected line segments. Colour information for shading the line segments can be specified with each point as part of the primitive definition.

POLYLINE SET 3 WITH DATA elements containing less than two points can be created, but are treated in a workstation-dependent fashion.

**Attributes Applied**

The attributes listed below are used to display the POLYLINE SET 3 WITH DATA primitive when the structure is traversed. The Aspect Source Flags (ASFs) tell where to access the output display attributes. These attributes can come directly from the traversal state list, or they can be accessed indirectly, using the appropriate index in the traversal state list and the corresponding bundled representation in the PHIGS workstation state list.

| | |
|---|---|
| polyline colour | polyline colour index ASF |
| linewidth scale factor | linewidth scale factor ASF |
| linetype | linetype ASF |
| polyline shading method | polyline shading method ASF |
| polyline index | |
| depth cue index | |
| name set | |

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

INQUIRE POLYLINE FACILITIES PLUS (3P+)
POLYLINE 3 (3P)

**NAME**

POLYMARKER – create structure element specifying 2D polymarker primitive

**SYNOPSIS**

**C Syntax**

```
void
ppolymarker ( point_list )
Ppoint_list        *point_list;        list of points
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

The POLYMARKER function puts the full specification of a POLYMARKER primitive into the currently open structure. A POLYMARKER primitive is a series of markers, such as dots, plus signs, asterisks, circles, or *x*'s, drawn in the display at specified Modelling Coordinate (MC) points on the $z = 0$ plane.

**C Input Parameters**

*point_list*

A pointer to a Ppoint_list structure containing a list of Ppoint structures, which contain the *x* and *y* coordinates for each point at which markers will be placed.

The Ppoint_list structure is defined in phigs.h as follows:

```
typedef struct  {
        Pint      num_points;    /* number of Ppoint elements in the list */
        Ppoint    *points;       /* list of points */
} Ppoint_list;
```

The *num_points* component specifies the number of elements in the list. The *points* component is a pointer to a list of Ppoints *num_points* long. The Ppoint structure is defined in phigs.h as follows:

```
typedef struct  {
        Pfloat    x;    /* x coordinate */
        Pfloat    y;    /* y coordinate */
} Ppoint;
```

**Execution**

If the current edit mode is INSERT, the POLYMARKER element is inserted in the open structure after the element pointed at by element pointer. If the current edit mode is REPLACE, the POLYMARKER element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new POLYMARKER element.

When the structure is traversed, the POLYMARKER element places a marker at each of the coordinate locations specified. The current values of the attributes listed below will be applied to the marker.

The coordinates used to specify the polymarker primitive are MCs. These may be any coordinate units that are convenient to the application. At traversal, these coordinate values are transformed by the current local and global modelling transformations, the view representation selected by the current view index, and the workstation transformation current on the workstation to which the structure is posted.

**Attributes Applied**

The attributes listed below are used to display the POLYMARKER primitive when the structure is traversed. The Aspect Source Flags (ASFs) tell where to access the output display attributes. These attributes can come directly from the traversal state list, or they can be accessed indirectly, using the appropriate index in the traversal state list and the corresponding bundled representation in the PHIGS workstation state list.

| | |
|---|---|
| marker type | marker type ASF |

| | |
|---|---|
| marker size scale factor | marker size scale factor ASF |
| polymarker colour | polymarker colour index ASF |
| polymarker index | |
| depth cue index | |
| name set | |

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

INQUIRE POLYMARKER FACILITIES (3P)
POLYMARKER 3 (3P)

**NAME**

POLYMARKER 3 – create structure element specifying 3D polymarker primitive

**SYNOPSIS**

**C Syntax**

```
void
ppolymarker3 ( point_list )
Ppoint_list3        *point_list;        list of points
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

The POLYMARKER 3 function puts the full specification of a POLYMARKER 3 primitive into the currently open structure. A POLYMARKER 3 primitive is a series of markers, such as dots, plus signs, asterisks, circles, or *x*'s drawn in the display at the specified Modelling Coordinate (MC) points.

**C Input Parameters**

*point_list*

A pointer to a Ppoint_list3 structure containing a list Ppoint3 structures, which contain the *x* and *y* coordinates for each point at which markers will be placed. The Ppoint_list3 structure is defined in phigs.h as follows:

```
typedef struct  {
    Pint    num_points;     /* number of Ppoint3 elements in the list */
    Ppoint3  *points;       /* list of points */
} Ppoint_list3;
```

The *num_points* component specifies the number of elements in the list. The *points* component is a pointer to a list of Ppoint3s *num_points* long. The Ppoint3 structure is defined in phigs.h as follows:

```
typedef struct  {
        Pfloat    x;    /* x coordinate */
        Pfloat    y;    /* y coordinate */
        Pfloat    z;    /* z coordinate */
} Ppoint3;
```

**Execution**

If the current edit mode is INSERT, the POLYMARKER 3 element is inserted in the open structure after the element pointed at by the element pointer. If the current edit mode is REPLACE, the POLYMARKER element replaces the element pointed at by the element pointer. In either case, the element pointer is updated to point to the new POLYMARKER 3 element.

When the structure is traversed, the POLYLINE 3 element places a marker at each of the coordinate locations specified. The current values of the attributes listed below will be applied to the marker.

The marker positions are specified in Model Coordinates (MC). These coordinates can be any coordinate units that are convenient to the application. At traversal, these coordinate values are transformed by the current local and global modelling transformations, the view representation selected by the current view index, and the workstation transformation current on the workstation to which the structure is posted.

**Attributes Applied**

The attributes listed below are used to display the POLYMARKER 3 primitive when the structure is traversed. The Aspect Source Flags (ASFs) tell where to access the output display attributes. These attributes can come directly from the traversal state list, or they can be accessed indirectly, using the appropriate index in the traversal state list and the corresponding bundled representation in the PHIGS workstation state list.

| | |
|---|---|
| marker type | marker type ASF |
| marker scale factor | marker size factor ASF |
| polymarker colour | polymarker colour index ASF |
| polymarker index | |
| depth cue index | |
| name set | |

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

INQUIRE POLYMARKER FACILITIES (3P)
POLYMARKER (3P)

**NAME**

POST STRUCTURE – assign a structure network to a workstation for display

**SYNOPSIS**

**C Syntax**

```
void
ppost_struct ( ws_id, struct_id, priority )
Pint        ws_id;          workstation identifier
Pint        struct_id;      structure identifier
Pfloat      priority;       priority
```

**Required PHIGS Operating States**

(PHOP, WSOP, ∗, ∗)

**DESCRIPTION**

**Purpose**

POST STRUCTURE assigns a structure, and the structure network of which it is the root, to a workstation for display. The workstation's *display update state* determines when the structure network is traversed, updating the display.

**C Input Parameters**

*ws_id* The identifier of the workstation to which the structure should be posted.

*struct_id*
The identifier of the structure to be posted.

*priority* The display priority of this structure in relation to other structures posted to the same workstation, in the range of 0.0 to 1.0, inclusive.

**Execution**

POST STRUCTURE adds *structure identifier* to the *table of posted structures* on the workstation with *workstation identifier*. If the structure does not exist, it will be created as an empty structure and posted to *ws_id*.

If more than one structure network is posted to the workstation, display *priority* determines the display order of the structure networks. Structures with higher priority values take precedence over (i.e., are drawn on top of) structures with lower priority values. If the structure identifier is already posted on the workstation, the structure is removed from the table of posted structures, and reposted with the priority value specified. If two structures are posted with the same priority, the structure posted (or reposted) last assumes a higher priority.

If the workstation's display update state permits, POST STRUCTURE initiates traversal of all the structure networks posted to the workstation, ordered by their priority. (See SET DISPLAY UPDATE STATE for more information on the workstation display update state.) The structure elements are traversed in sequence, beginning with the first element of the posted structure. Traversal of subordinate structures in the network is prompted by EXECUTE STRUCTURE elements.

The structure network and the workstation state together determine the image displayed on the workstation's display surface. While a structure remains posted to a workstation, any changes in the structure network or the workstation's tables can result in immediate changes to the display, depending on the structure's priority and the workstation's display update state.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, ∗, ∗)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (i.e., the workstation category is neither OUTPUT, OUTIN, nor MO)

208 Ignoring function, the display priority is out of range

**SEE ALSO**

UNPOST STRUCTURE (3P)
UNPOST ALL STRUCTURES (3P)
INQUIRE POSTED STRUCTURES (3P)
INQUIRE SET OF WORKSTATIONS TO WHICH POSTED (3P)
INQUIRE NUMBER OF DISPLAY PRIORITIES SUPPORTED (3P)

**NAME**

QUADRILATERAL MESH 3 WITH DATA – creates a 3D quadrilateral mesh structure element that includes colour and shading data

**SYNOPSIS**

**C Syntax**

**void**
**pquad_mesh3_data ( fflag, vflag, colour_model, dim, fdata, vdata )**

| | | |
|---|---|---|
| **Pint** | **fflag;** | *what data per fill facet is available* |
| **Pint** | **vflag;** | *what data per vertex is available* |
| **Pint** | **colour_model;** | *colour type* |
| **Pint_size** | ∗**dim;** | *dimension of cells (in vertices)* |
| **Pfacet_data_arr3** | ∗**fdata;** | *facet data* |
| **Pfacet_vdata_arr3** | ∗**vdata;** | *vertex data* |

**Required PHIGS Operating States**

(PHOP, ∗, STOP, ∗)

**DESCRIPTION**

**Purpose**

QUADRILATERAL MESH 3 WITH DATA creates a 3D quadrilateral mesh primitive with colour and shading data.

**C Input Parameters**

*fflag* The data per facet flag specifies the available data for each quadrilateral of the quadrilateral mesh.

| | | |
|---|---|---|
| 0 | PFACET_NONE | *No Facet Data Specified* |
| 1 | PFACET_COLOUR | *Facet Colours Specified* |
| 2 | PFACET_NORMAL | *Facet Normal Specified* |
| 3 | PFACET_COLOUR_NORMAL | *Facet Normal and Colours Specified* |

*vflag* The data per vertex flag specifies the available data for each vertex of the primitive.

| | | |
|---|---|---|
| 0 | PVERT_COORD | *Coordinates Specified* |
| 1 | PVERT_COORD_COLOUR | *Coordinates and Colours Specified* |
| 2 | PVERT_COORD_NORMAL | *Coordinates and Vertex Normal Specified* |
| 3 | PVERT_COORD_COLOUR_NORMAL | *Coordinates, Vertex Colour, and Vertex Normal Specified* |

*colour_model*

The colour type for specified vertex colours.

| | | |
|---|---|---|
| 0 | PINDIRECT | *Colour Index Specified* |
| 1 | PMODEL_RGB | *Red, Green, and Blue* |
| 2 | PMODEL_CIELUV | *CIE Colour Model* |
| 3 | PMODEL_HSV | *Hue, Saturation, and Value* |
| 4 | PMODEL_HLS | *Hue, Lightness, and Saturation* |

*dim* A pointer to a Pint_size structure that specifies the number of vertices along the *x* and *y* coordinates. Pint_size is defined in phigs.h as follows:

```
typedef struct {
        Pint    size_x;    /* number of vertices along the x-axis */
        Pint    size_y;    /* number of vertices along the y-axis */
} Pint_size;
```

*fdata* A pointer to a Pfacet_data_arr3 structure that specifies the colour and/or normal information for each facet. Note that number of facets = (size_x - 1) * (size_y - 1). Pfacet_data_arr3 is defined in phigs.h as follows:

```
typedef union {
        Pcoval      *colrs;     /* array (size_x - 1) * (size_y - 1) of colours */
        Pvec3       *norms;     /* array (size_x - 1) * (size_y - 1) of unit normals */
        Pconorm3    *conorms;   /* array (size_x - 1) * (size_y - 1) of colours and normals */
} Pfacet_data_arr3;
```

Pcoval is defined in phigs.h as follows:

```
typedef union {
        Pint        index;  /* index in workstation colour bundle table */
        Pcolr_rep   direct; /* direct colour components */
} Pcoval;
```

When *colour_model* is PINDIRECT, index is used; otherwise, direct is used.

Pcolr_rep is defined in phigs.h as follows:

```
typedef union {
        Prgb        rgb;        /* Red Green Blue colour specification */
        Pcieluv     cieluv;     /* CIE L*U*V* colour specification */
        Phls        hls;        /* Hue Lightness Saturation colour specification */
        Phsv        hsv;        /* Hue Saturation Value colour specification */
        Pdata       unsupp;     /* Colour in unsupported colour model */
} Pcolr_rep;
```

Prgb is defined in phigs.h as follows:

```
typedef struct {
        Pfloat      red;        /* red, hue, etc */
        Pfloat      green;      /* green, saturation, lightness, etc */
        Pfloat      blue;       /* blue, value, saturation, etc */
} Prgb;
```

Pcieluv is defined in phigs.h as follows:

```
typedef struct {
        Pfloat      cieluv_x;       /* x coefficient */
        Pfloat      cieluv_y;       /* y coefficient */
        Pfloat      cieluv_y_lum;   /* y luminance */
} Pcieluv;
```

Phls is defined in phigs.h as follows:

```
typedef struct {
        Pfloat      hue;        /* hue */
        Pfloat      lightness;  /* lightness */
        Pfloat      satur;      /* saturation */
} Phls;
```

Phsv is defined in phigs.h as follows:

```
typedef struct {
        Pfloat      hue;        /* hue */
        Pfloat      satur;      /* saturation */
        Pfloat      value;      /* value */
```

```
} Phsv;
```

Pdata is defined in phigs.h as follows:

```
typedef struct {
        size_t          size;           /* size of data */
        char            *data           /* pointer to data */
} Pdata;
```

Pvec3 is defined as:

```
typedef struct {
        Pfloat  delta_x;    /* x magnitude */
        Pfloat  delta_y;    /* y magnitude */
        Pfloat  delta_z;    /* z magnitude */
} Pvec3;
```

Pconorm3 is defined as:

```
typedef union {
        Pcoval  colr;       /* colour */
        Pvec3   norm;       /* unit normal */
} Pconorm3;
```

Pcoval and Pvec3 are defined above.

*vdata* A pointer to a Pfacet_vdata_arr3 structure that specifies the quadrilateral mesh vertices and optionally associated colour and normal information. Pfacet_vdata_arr3 is defined as:

```
typedef union {
        Ppoint3         *points;        /* array of (size_x * size_y) of points */
        Pptco3          *ptcolrs;       /* array of (size_x * size_y) of points and colours */
        Pptnorm3        *ptnorms;       /* array of (size_x * size_y) of points and normals */
        Pptconorm3      *ptconorms;     /* array of (size_x * size_y) of points, colours, and normals */
        /* implementation-dependent types can go here */
} Pfacet_vdata_arr3;
```

Ppoint3 is defined as:

```
typedef struct {
        Pfloat  x;    /* x coordinate */
        Pfloat  y;    /* y coordinate */
        Pfloat  z;    /* z coordinate */
} Ppoint3;
```

Pptco3 is defined as:

```
typedef struct {
        Ppoint3     point;    /* point coordinates */
        Pcoval      colr;     /* colour */
} Pptco3;
```

Ppoint3 is defined above. Pcoval is defined as:

```
typedef union {
        Pint        ind;      /* index in workstation colour bundle table */
        Pcolr_rep   direct;   /* direct colour components */
```

```
} Pcoval;
```

Pcolr_rep is defined above. Pptnorm3 is defined as:

```
typedef struct {
        Ppoint3  point;    /* point coordinates */
        Pvec3    norm;     /* unit normal */
} Pptnorm3;
```

Ppoint3 and Pvec3 are defined above. Pptconorm3 is defined as:

```
typedef struct {
        Ppoint3    point;    /* point coordinates */
        Pcoval     colr;     /* colour */
        Pvec3      norm;     /* unit normal */
} Pptconorm3;
```

Ppoint3, Pcoval, and Pvec3 are defined above.

**Execution**

Depending on the edit mode, a QUADRILATERAL MESH 3 WITH DATA element is inserted into the open structure after the element pointer, or it replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the newly created QUADRILATERAL MESH 3 WITH DATA element.

When the structure is traversed, QUADRILATERAL MESH 3 WITH DATA generates a three-dimensional mesh of $(m\text{-}1) \times (n\text{-}1)$ quadrilaterals from a two-dimensional array of $m \times n$ vertices. Line segments form the boundary of each quadrilateral in the mesh. Each quadrilateral can be rendered empty, hollow, shaded, or filled with a colour, pattern, or hatch style. A mesh with $n < 2$ or $m < 2$ vertices can be created, but it is treated in a workstation-dependent fashion.

QUADRILATERAL MESH 3 WITH DATA uses the same attributes as the FILL AREA SET 3 WITH DATA primitive.

The QUADRILATERAL MESH 3 WITH DATA element can specify the facet colour and normal vector of each quadrilateral in the mesh, as well as colours and normal vectors for each vertex. These colours and normal vectors are used in conjunction with the current lighting and depth-cueing attributes to colour and shade the primitive. Note that unexpected lighting effects will result if the supplied normals are not of unit length.

**Attributes Applied**

The attributes listed below are used to display the QUADRILATERAL MESH 3 WITH DATA primitive when the structure is traversed. The Aspect Source Flags (ASFs) tell where to access the output display attributes. These attributes can come directly from the traversal state list, or they can be accessed indirectly, using the appropriate index in the traversal state list and the corresponding bundled representation in the PHIGS workstation state list.

| | |
|---|---|
| interior colour | interior colour index ASF |
| back interior colour | back interior colour ASF |
| interior style | interior style ASF |
| back interior style | back interior style ASF |
| interior style index | interior style index ASF |
| back interior style index | back interior style index ASF |
| interior shading method | interior shading method ASF |
| back interior shading method | back interior shading method ASF |
| interior reflectance equation | interior reflectance equation ASF |
| back interior reflectance equation | back interior reflectance equation ASF |
| area properties | area properties ASF |
| back area properties | back area properties ASF |

| | |
|---|---|
| interior index | |
| edge colour | edge colour index ASF |
| edge flag | edge flag ASF |
| edgetype | edgetype ASF |
| edgewidth scale factor | edgewidth scale factor ASF |
| edge index | |
| face distinguishing mode | |
| face culling mode | |
| depth cue index | |
| light source state | |
| name set | |

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

FILL AREA SET 3 WITH DATA (3P+)

**NAME**

REDRAW ALL STRUCTURES – redisplay all structures posted to a workstation

**SYNOPSIS**

**C Syntax**

```
void
predraw_all_structs ( ws, control_flag )
Pint            ws;             workstation identifier
Pctrl_flag      control_flag;   controls the redraw of the structures
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

REDRAW ALL STRUCTURES updates the specified workstation's display by:

- Completing any actions in progress
- Clearing the display surface if the surface is NOT EMPTY, or if the *control flag* is ALWAYS
- Updating any pending viewing transformation and pending HLHSR mode
- Redisplaying all structures posted to the workstation
- Setting the workstation's *state of visual representation* to CORRECT

**C Input Parameters**

*ws* The identifier of the workstation whose structures are to be redrawn.

*control_flag*

Controls whether the display surface is automatically cleared. Pctrl_flag is defined in phigs.h. Valid values are:

```
typedef enum {
        PFLAG_COND,
        PFLAG_ALWAYS
}Pctrl_flag;
```

PFLAG_COND clears the display surface before redrawing only if the surface is not empty.

PFLAG_ALWAYS automatically clears the surface before redrawing.

**Execution**

REDRAW ALL STRUCTURES performs the following actions in sequence:

1. Any buffered communications for the specified workstation are completed without first clearing the display surface.
2. If the *control flag* is set to CONDITIONALLY, then the workstation's display surface is only cleared if it is NOT EMPTY. If the control flag is set to ALWAYS, the display surface is cleared regardless of whether the display is EMPTY. In any case, the entry is then set to EMPTY.

   **Note:** Using a control flag of CONDITIONALLY avoids unnecessarily clearing an empty display surface. A value of ALWAYS is useful when generating multiple copies of the current image on hardcopy devices. However, for PEX-SI where an output device is always a CRT, the value ALWAYS is not meaningful.
3. For every view representation in the workstation's state list, if the *view transformation update state* is PENDING, the current view representation is loaded from the *requested* view representation, and the update state is set to NOTPENDING.
4. If the workstation transformation update state is PENDING, the *current workstation window* and *current workstation viewport* are loaded with the *requested* values for each, and the update state is

set to NOTPENDING.

5. If the workstation's *HLHSR update state* is PENDING, the *current HLHSR mode* is updated to the value of the *requested HLHSR mode*, and the update state is set to NOTPENDING.
6. All structure networks posted for this workstation are redisplayed in the order of their priority.
7. The workstation's *state of visual representation* is set to CORRECT.

The workstation's state of visual representation indicates if the display is CORRECT, SIMULATED, or DEFERRED. INQUIRE DISPLAY UPDATE STATE returns the workstation's *display surface empty* and state of visual representation state list entries. Normally, the traversal in Step 6 causes the display surface empty workstation state to become NOTEMPTY. (If all the posted structures are empty, or if they contain elements but no output primitive elements, the display surface empty may be EMPTY or NOTEMPTY.)

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (i.e., the workstation category is neither OUTPUT, OUTIN, nor MO)

**SEE ALSO**

UPDATE WORKSTATION (3P)
INQUIRE DISPLAY UPDATE STATE (3P)
SET DISPLAY UPDATE STATE (3P)

**NAME**

REMOVE NAMES FROM SET – create a structure element to remove names from the current name set

**SYNOPSIS**

**C Syntax**

```
void
premove_names_set ( set )
Pint_list          *set;          set of names to be removed
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

REMOVE NAMES FROM SET creates a structure element containing names to be removed from the traversal-time *current name set*, and puts the element into the currently open structure according to the current edit mode. The current name set is compared during traversal to the workstation's *name set filters* to determine if primitives which follow in the structure network are invisible, highlighted, and/or selectable by PICK input devices. Each name in the *name set* is a small non-negative integer.

**C Input Parameter**

*set* A pointer to a Pint_list structure containing the set of names to be removed. The Pint_list data structure is defined in phigs.h as:

```
typedef struct {
        Pint    num_ints;   /* number of Pints in list */
        Pint    *ints;      /* list of integers */
} Pint_list;
```

**Execution**

If the current edit mode is INSERT, the REMOVE NAMES FROM SET element is inserted into the open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, the REMOVE NAMES FROM SET element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

When traversal of a posted structure network starts, the *current name set* is empty. During traversal, the member names specified by ADD NAMES TO SET elements are added to the current name set. The member names specified by REMOVE NAMES FROM SET elements are removed from the current name set.

The updated current name set applies to primitives that follow in the structure network by set-intersection with the workstation's *name set filters*, which are set by SET INVISIBILITY FILTER, SET HIGHLIGHTING FILTER, and SET PICK FILTER. Each workstation has a single *invisibility filter*, a single *highlighting filter*, and a *pick filter* for each PICK input device. The actual appearance of highlighting is workstation-dependent.

Each filter contains an *inclusion set* and an *exclusion set* of names, both empty by default. A primitive is *eligible* if at least one name in the current name set is in the inclusion set and no name in the current name set is in the exclusion set. If the current name set is empty, subsequent primitives are not eligible. If the workstation's inclusion set is empty (the default), no primitives are eligible. That is, no primitives are invisible, highlighted, or selectable by PICK input devices.

Each name is a small non-negative integer. PHIGS conformance requires support for at least 64 names; PEXSI supports the range 0 to 1023. The same names can be added and removed any number of times during traversal. Since the presence or absence of each name in the current nameset and the workstation's filter affects the eligibility of subsequent primitives, 1024 names provide up to 1024 different simultaneous groupings of primitives.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

ADD NAMES TO SET (3P)
SET INVISIBILITY FILTER (3P)
SET HIGHLIGHTING FILTER (3P)
SET PICK FILTER (3P)
INCREMENTAL SPATIAL SEARCH (3P)
INCREMENTAL SPATIAL SEARCH 3 (3P)

**NAME**

REQUEST CHOICE – request an operator interaction with a choice device

**SYNOPSIS**

**C Syntax**

```
void
preq_choice ( ws, dev, in_status, choice )
Pint            ws;             workstation identifier
Pint            dev;            choice device number
Pin_status      *in_status;     OUT input status
Pint            *choice;        OUT choice
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use REQUEST CHOICE to request an operator interaction with a specified choice device. The device must be in request mode. See SET CHOICE MODE and INITIALIZE CHOICE for more information. See the *Available Devices* section of INITIALIZE CHOICE for a description of the available choice devices.

The request suspends PHIGS until the specified device is triggered, or the operator performs a BREAK. A BREAK is performed by typing the CONTROL and D keys simultaneously (^D).

**C Input Parameters**

*ws* Workstation identifier. An integer specifying the workstation with which the requested choice device is associated.

*dev* The device number of the requested choice device.

**C Output Parameters**

*in_status*

A pointer to the location to store the status of the request. Pin_status is defined in phigs.h as follows:

```
typedef enum {
        PIN_STATUS_NONE,
        PIN_STATUS_OK,
        PIN_STATUS_NO_IN
} Pin_status;
```

*choice* A pointer to the location to store the selected choice value, if any.

**Execution**

REQUEST CHOICE requests a logical input value from the specified choice device. The specified device must be in request mode. See SET CHOICE MODE for more information.

REQUEST CHOICE creates a measure process for the specified device and suspends PHIGS until the device is triggered or the operator issues a BREAK. A BREAK is generated by the operator by depressing the CONTROL and D keys simultaneously (^D).

If the choice device is triggered and a choice has not been made, the status is returned as PIN_STATUS_NONE, and no choice value is returned.

If the choice device is triggered and a choice has been made, the status is returned as PIN_STATUS_OK, and the choice value contains the choice number selected by the operator.

If a BREAK occurs, the status is returned as PIN_STATUS_NO_IN, and no choice value is returned.

In either case, when the device is triggered or a BREAK is issued, the current measure process is terminated and PHIGS processing resumes.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

251 Ignoring function, the function requires the input device to be in REQUEST mode

**SEE ALSO**

INITIALIZE CHOICE 3 (3P)
SET CHOICE MODE (3P)

**NAME**

REQUEST LOCATOR – request an operator interaction with a locator device

**SYNOPSIS**

**C Syntax**

```
void
preq_loc ( ws, dev, in_status, view_ind, loc_pos )
Pint        ws;          workstation identifier
Pint        dev;         locator device number
Pin_status  *in_status;  OUT input status
Pint        *view_ind;   OUT view index
Ppoint      *loc_pos;    OUT locator position
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use REQUEST LOCATOR to request an operator interaction with a specified locator device. The device must be in request mode. See SET LOCATOR MODE and INITIALIZE LOCATOR for more information. See the *Available Devices* section of INITIALIZE LOCATOR for a description of the available locator devices.

The request suspends PHIGS until the specified device is triggered, or the operator performs a BREAK. A BREAK is performed by typing the CONTROL and D keys simultaneously (^D).

**C Input Parameters**

*ws* Workstation identifier. An integer specifying the workstation with which the requested locator device is associated.

*dev* The device number of the requested locator device. See the *Available Devices* section of INITIALIZE LOCATOR for a description of the available locator devices.

**C Output Parameters**

*in_status*

A pointer to the location to store the status of the request. Pin_status

```
typedef enum {
        PIN_STATUS_NONE,
        PIN_STATUS_OK,
        PIN_STATUS_NO_IN
} Pin_status;
```

*view_ind*

A pointer to the location to store the view representation used to map from Device Coordinates (DC) to World Coordinates (WC).

*loc_pos* A pointer to a Ppoint structure in which the system returns the locator position coordinates. Ppoint is defined in phigs.h as follows:

```
typedef enum {
        Pfloat  x;      /* x coordinate */
        Pfloat  y;      /* y coordinate */
} Ppoint;
```

**Execution**

REQUEST LOCATOR requests a logical input value from the specified locator device. The specified device must be in request mode. See SET LOCATOR MODE for more information.

REQUEST LOCATOR creates a measure process for the specified device and suspends PHIGS until the device is triggered or the operator issues a BREAK. A BREAK is generated by the operator by depressing the Control

and D keys simultaneously (^D).

If a BREAK occurs, a status of PIN_STATUS_NO_IN is returned and the locator data is undefined. If a status of PIN_STATUS_OK is returned, the locator data is available. In either case, the measure process is then terminated and PHIGS processing resumes.

A locator measure consists of a *view_ind* and *loc_pos*. The *view_ind* specifies the view representation from the workstation's view table that was used to map the locator position from Normalized Projections Coordinates (NPC) to WC. The *loc_pos* is the position of the locator in WC. **Note:** The 2D and 3D locator measure processes are the same except that the 2D process discards the *z* coordinate. The 3D version of this function, REQUEST LOCATOR 3, can be used if the value of the *z* coordinate is needed.

See INITIALIZE LOCATOR for a description of the available locator devices.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

251 Ignoring function, the function requires the input device to be in REQUEST mode

**SEE ALSO**

INITIALIZE LOCATOR (3P)
SET LOCATOR MODE (3P)
REQUEST LOCATOR 3 (3P)
SET VIEW TRANSFORMATION INPUT PRIORITY (3P)

**NAME**

REQUEST LOCATOR 3 – request an operator interaction with a locator device

**SYNOPSIS**

**C Syntax**

**void**
**preq_loc3 ( ws, dev, in_status, view_ind, loc_pos )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **dev;** | *locator device number* |
| **Pin_status** | ***in_status;** | *OUT input status* |
| **Pint** | ***view_ind;** | *OUT view index* |
| **Ppoint3** | ***loc_pos;** | *OUT locator position* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use REQUEST LOCATOR 3 to request an operator interaction with a specified locator device. The device must be in request mode. See SET LOCATOR MODE and INITIALIZE LOCATOR 3 for more information. See the *Available Devices* section of INITIALIZE LOCATOR 3 for a description of the available locator devices.

The request suspends PHIGS until the specified device is triggered, or the operator performs a BREAK. A BREAK is performed by typing the CONTROL and D keys simultaneously (^D).

**C Input Parameters**

*ws* Workstation identifier. An integer specifying the workstation with which the requested locator device is associated.

*dev* The device number of the requested locator device. See the *Available Devices* section of INITIALIZE LOCATOR 3 for a description of the available locator devices.

**C Output Parameters**

*in_status*

A pointer to the location to store the status of the request. Pin_status is defined in phigs.h as follows:

```
typedef enum {
        PIN_STATUS_NONE,
        PIN_STATUS_OK,
        PIN_STATUS_NO_IN
} Pin_status;
```

*view_ind*

A pointer to the location to store the view representation used to map from Device Coordinates (DC) to World Coordinates (WC).

*loc_pos* A pointer to a Ppoint3 structure in which to the system stores the locator position coordinates. Ppoint3 is defined in phigs.h as follows:

```
typedef struct {
        Pfloat    x;    /* x coordinate */
        Pfloat    y;    /* y coordinate */
        Pfloat    z;    /* z coordinate */
} Ppoint3;
```

**Execution**

REQUEST LOCATOR 3 requests a logical input value from the specified locator device. The specified device must be in request mode. See SET LOCATOR MODE for more information.

REQUEST LOCATOR 3 creates a measure process for the specified device and suspends PHIGS until the device is triggered or the operator issues a BREAK. A BREAK is generated by the operator by depressing the Control and D keys simultaneously (^D).

If a BREAK occurs, a status of PIN_STATUS_NO_IN is returned, and the locator data is undefined. If a status of PIN_STATUS_OK is returned, the locator data is available. In either case, the measure process is then terminated and PHIGS processing resumes.

A locator measure consists of a *view_ind* and WC *loc_pos*. The *view_ind* specifies the view representation from the workstation's view table that was used to map the locator position from Normalized Projections Coordinates (NPC) to WC. The *loc_pos* is the position of the locator in WC. **Note:** The 2D and 3D locator measure processes are the same except that the 2D process discards the *z* coordinate. The 2D version of this function, REQUEST LOCATOR, can be used if the value of the *z* coordinate is not needed.

See INITIALIZE LOCATOR 3 for a description of the available locator devices.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

251 Ignoring function, the function requires the input device to be in REQUEST mode

**SEE ALSO**

INITIALIZE LOCATOR 3 (3P)
SET LOCATOR MODE (3P)
REQUEST LOCATOR (3P)
SET VIEW TRANSFORMATION INPUT PRIORITY (3P)

**NAME**

REQUEST PICK – request an operator interaction with a pick device

**SYNOPSIS**

**C Syntax**

**void**
**preq_pick ( ws, dev, depth, in_status, rpick )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **dev;** | *pick device number* |
| **Pint** | **depth;** | *max. depth of returned path* |
| **Pin_status** | ***in_status;** | *OUT input status* |
| **Ppick_path** | ***rpick;** | *OUT pick* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use REQUEST PICK to request an operator interaction with a specified pick device. The device must be in request mode. See SET PICK MODE and INITIALIZE PICK for more information. See the *Available Devices* section of INITIALIZE PICK for a description of the available pick devices.

The request suspends PHIGS until the specified device is triggered, or the operator performs a BREAK. A BREAK is performed by typing the CONTROL and D keys simultaneously (^D).

**C Input Parameters**

*ws* Workstation identifier. An integer specifying the workstation with which the requested pick device is associated.

*dev* The device number of the requested pick device.

*depth* The maximum depth of the pick path to return.

**C Output Parameters**

*in_status*

A pointer to the location to store the status of the request. Pin_status is defined in phigs.h as follows:

```
typedef enum {
        PIN_STATUS_NONE,
        PIN_STATUS_OK,
        PIN_STATUS_NO_IN
} Pin_status;
```

*rpick* A pointer to a Ppick_path structure in which the system stores the pick path. Ppick_path is defined in phigs.h as:

```
typedef struct {
        Pint                depth;      /* pick path depth */
        Ppick_path_elem     *path_list; /* pick path list */
} Ppick_path;
```

*depth* indicates the number of elements in the measure's path. This is the depth value contained in the device's current measure. It is not affected by the *maximum depth to return* parameter. Thus, the number of elements returned in *path_list* may be less than *depth*.

*path_list* is the array of path elements defining the location of the picked primitive in the Central Structure Store (CSS). **This array must be allocated by the calling program and the array pointer assigned to this field before calling this function**. The array must be at least of length *maximum depth to return*. Ppick_path_elem is defined in phigs.h as:

```
typedef struct {
        Pint            struct_id;      /* structure identifier */
        Pint            pick_id;        /* pick identifier */
        Pint            elem_pos;       /* element number */
} Ppick_path_elem;
```

*struct_id*, *pick_id*, and *elem_pos* are the structure identifier, pick identifier, and element number, respectively, of each element in the path.

**Execution**

REQUEST PICK requests a logical input value from the specified pick device. The specified device must be in request mode. See SET PICK MODE for more information.

REQUEST PICK creates a measure process for the specified device and suspends PHIGS until the device is triggered or the operator issues a BREAK. A BREAK is generated by the operator by depressing the CONTROL and D keys simultaneously (^D).

If the pick device is triggered and a pick has not been made, the status is returned as PIN_STATUS_NONE, and no pick data is returned. If the pick device is triggered and a pick has been made, the status is returned as PIN_STATUS_OK, and the pick data is available in the output parameters. If a BREAK occurs, the status is returned as PIN_STATUS_NO_IN, and no pick value is returned.

In all cases, when the device is triggered or a BREAK is issued, the current measure process is terminated and PHIGS processing resumes.

See INITIALIZE PICK 3 for a description of the available PICK devices and how their measure values are determined.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

060 Ignoring function, specified workstation is not of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

251 Ignoring function, the function requires the input device to be in REQUEST mode

**SEE ALSO**

INITIALIZE PICK 3 (3P)
SET PICK FILTER (3P)
SET PICK IDENTIFIER (3P)
SET PICK MODE (3P)

**NAME**

REQUEST STRING – request an operator interaction with a string device

**SYNOPSIS**

**C Syntax**

```
void
preq_string ( ws, dev, in_status, string )
Pint          ws;          workstation identifier
Pint          dev;         string device number
Pin_status    *in_status;  OUT input string
char          *string;     OUT string
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use REQUEST STRING to request an operator interaction with a specified string device. The device must be in request mode. See SET STRING MODE and INITIALIZE STRING for more information. See the *Available Devices* section of INITIALIZE STRING for a description of the available string devices.

The request suspends PHIGS until the specified device is triggered or the operator performs a BREAK. A BREAK is performed by typing the CONTROL and D keys simultaneously (^D).

**C Input Parameters**

*ws* Workstation identifier. An integer specifying the workstation with which the requested string device is associated.

*dev* The device number of the requested string device. See the *Available Devices* section of INITIALIZE STRING for a description of the available string devices.

**C Output Parameter**

*in_status*

A pointer to the location to store the status of the request. Pin_status is defined in phigs.h as follows:

```
typedef enum {
        PIN_STATUS_NONE,
        PIN_STATUS_OK,
        PIN_STATUS_NO_IN
} Pin_status;
```

*string* A pointer to an array in which to store the string. The array must be allocated by the caller and should be at least as large as the buffer size of the device.

**Execution**

REQUEST STRING requests a logical input value from the specified string device. The specified device must be in request mode. See SET STRING MODE for more information.

REQUEST STRING creates a measure process for the specified device and suspends PHIGS until the device is triggered or the operator issues a BREAK. A BREAK is generated by the operator by depressing the CONTROL and D keys simultaneously (^D).

If a BREAK occurs, a status of PIN_STATUS_NO_IN is returned and the string data is undefined. If a status of PIN_STATUS_OK is returned, the string data is available. In either case, the measure process is terminated and PHIGS processing resumes.

For a description of the available string devices, see INITIALIZE STRING.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

| | |
|---|---|
| 054 | Ignoring function, the specified workstation is not open |
| 061 | Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN |
| 250 | Ignoring function, the specified device is not available on the specified workstation |
| 251 | Ignoring function, the function requires the input device to be in REQUEST mode |

**SEE ALSO**

INITIALIZE STRING (3P)
SET STRING MODE (3P)

**NAME**

REQUEST STROKE – request an operator interaction with a stroke device

**SYNOPSIS**

**C Syntax**

**void**
**preq_stroke ( ws, dev, in_status, view_ind, stroke )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **dev;** | *stroke device number* |
| **Pin_status** | ***in_status;** | *OUT input status* |
| **Pint** | ***view_ind;** | *OUT view index* |
| **Ppoint_list** | ***stroke;** | *OUT stroke* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use REQUEST STROKE to request an operator interaction with a specified stroke device. The device must be in request mode. See SET STROKE MODE and INITIALIZE STROKE for more information. See the *Available Devices* section of INITIALIZE STROKE for a description of the available stroke devices.

The request suspends PHIGS until the specified device is triggered or the operator performs a BREAK. A BREAK is performed by typing the CONTROL and D keys simultaneously (^D).

**C Input Parameters**

*ws* Workstation identifier. An integer specifying the workstation with which the requested stroke device is associated.

*dev* The device number of the requested stroke device. See the *Available Devices* section of INITIALIZE STROKE for a description of the available stroke devices.

**C Output Parameter**

*in_status*

A pointer to the location to store the status of the request. Pin_status is defined in phigs.h as follows:

```
typedef enum {
        PIN_STATUS_NONE,
        PIN_STATUS_OK,
        PIN_STATUS_NO_IN
} Pin_status;
```

*view_ind*

A pointer to the location to store the view index used to transform the Device Coordinate (DC) positions to World Coordinate (WC) points.

*stroke* A pointer to the location to store the stroke points. Ppoint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint    num_points;     /* number of points in the list */
        Ppoint  *points;        /* list of points */
} Ppoint_list;
```

*num_points* is the number of points in *points*.

*points* is the array of Ppoint structures specifying the points in WC. **This array must be allocated by the calling program and the array pointer assigned to this field before calling this function**. The array must be at least as large as the buffer of the STROKE device. This buffer size is set when the device is initialized. Ppoint is defined in phigs.h as follows:

```
typedef struct {
        Pfloat    x;        /* x coordinate */
        Pfloat    y;        /* y coordinate */
} Ppoint;
```

**Execution**

REQUEST STROKE requests a logical input value from the specified stroke device. The specified device must be in request mode. See SET STROKE MODE for more information.

REQUEST STROKE creates a measure process for the specified device and suspends PHIGS until the device is triggered or the operator issues a BREAK. A BREAK is generated by the operator by depressing the CONTROL and D keys simultaneously (^D).

If a BREAK occurs, a status of PIN_STATUS_NO_IN is returned and the stroke data is undefined. If a status of PIN_STATUS_OK is returned, the stroke data is available. In either case, the measure process is then terminated and PHIGS processing resumes.

A stroke measure consists of a *view index* and WC *positions*. The *view index* specifies the view representation from the workstation's view table that was used to map the stroke positions from Normalized Projections Coordinates (NPC) to WC. The *positions* are the positions of the stroke in WC. **Note:** The 2D and 3D stroke measure processes are the same except that the 2D process discards the *z* coordinates. The 3D version of this function, REQUEST STROKE 3, can be used if the *z* coordinates are needed.

See INITIALIZE STROKE for a description of the available stroke devices.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

060 Ignoring function, the specified workstation is not of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

251 Ignoring function, the function requires the input device to be in REQUEST mode

**SEE ALSO**

INITIALIZE STROKE (3P)
SET STROKE MODE (3P)
SET VIEW TRANSFORMATION INPUT PRIORITY (3P)
REQUEST STROKE 3 (3P)

**NAME**

REQUEST STROKE 3 – request an operator interaction with a stroke device

**SYNOPSIS**

**C Syntax**

**void**
**preq_stroke3 ( ws, dev, in_status, view_ind, stroke )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **dev;** | *stroke device number* |
| **Pin_status** | ***in_status** | *OUT input status* |
| **Pint** | ***view_ind;** | *OUT view index* |
| **Ppoint_list3** | ***stroke;** | *OUT stroke* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use REQUEST STROKE 3 to request an operator interaction with a specified stroke device. The device must be in request mode. See SET STROKE MODE and INITIALIZE STROKE 3 for more information. See the *Available Devices* section of INITIALIZE STROKE 3 for a description of the available stroke devices.

The request suspends PHIGS until the specified device is triggered or the operator performs a BREAK. A BREAK is performed by typing the CONTROL and D keys simultaneously (^D).

**C Input Parameters**

*ws* Workstation identifier. An integer specifying the workstation with which the requested stroke device is associated.

*dev* The device number of the requested stroke device. See the *Available Devices* section of INITIALIZE STROKE 3 for a description of the available stroke devices.

**C Output Parameter**

*in_status*

A pointer to the location to store the request status. Pin_status is defined in phigs.h as follows:

```
typedef enum {
        PIN_STATUS_NONE,
        PIN_STATUS_OK,
        PIN_STATUS_NO_IN
} Pin_status;
```

*view_ind*

A pointer to the location to store the view index used to transform the Device Coordinate (DC) positions to World Coordinate (WC) points.

*stroke* A pointer to the location to store the stroke points. Ppoint_list3 is defined in phig.h as follows:

```
typedef struct {
        Pint      num_points;     /* number of points in list */
        Ppoint3  *points;         /* list of points  */
} Ppoint_list3;
```

*num_points* is the number of points in *points*.

*points* is the array of Ppoint3 structures specifying the points in WC. **This array must be allocated by the calling program and the array pointer assigned to this field before calling this function**. The array must be at least as large as the buffer of the STROKE device. This buffer size is set when the device is initialized. Ppoint3 is defined in phigs.h as follows:

```
typedef struct {
        Pfloat    x;        /* x coordinate */
        Pfloat    y;        /* y coordinate */
        Pfloat    z;        /* z coordinate */
} Ppoint3;
```

**Execution**

REQUEST STROKE 3 requests a logical input value from the specified stroke device. The specified device must be in request mode. See SET STROKE MODE for more information.

REQUEST STROKE 3 creates a measure process for the specified device and suspends PHIGS until the device is triggered or the operator issues a BREAK. A BREAK is generated by the operator by depressing the CONTROL and D keys simultaneously (^D).

If a BREAK occurs, a status of PIN_STATUS_NO_IN is returned and the stroke data is undefined. If a status of PIN_STATUS_OK is returned, the stroke data is available. In either case, the measure process is then terminated and PHIGS processing resumes.

A stroke measure consists of a *view index* and WC *positions*. The *view index* specifies the view representation from the workstation's view table that was used to map the stroke positions from Normalized Projections Coordinates (NPC) to WC. The *positions* are the positions of the stroke in WC. **Note:** The 2D and 3D stroke measure processes are the same except that the 2D process discards the *z* coordinates. The 2D version of this function, REQUEST STROKE, can be used if the *z* coordinates are not needed.

See INITIALIZE STROKE 3 for a description of the available stroke devices.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

251 Ignoring function, the function requires the input device to be in REQUEST mode

**SEE ALSO**

INITIALIZE STROKE 3 (3P)
SET STROKE MODE (3P)
SET VIEW TRANSFORMATION INPUT PRIORITY (3P)
REQUEST STROKE (3P)

**NAME**

REQUEST VALUATOR – request an operator interaction with a valuator device

**SYNOPSIS**

**C Syntax**

```
void
preq_val ( ws, dev, in_status, value )
Pint          ws;          workstation identifier
Pint          dev;         valuator device number
Pin_status    *in_status;  OUT input status
Pfloat        *value;      OUT value
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use REQUEST VALUATOR to request an operator interaction with a specified valuator device. The device must be in request mode. See SET VALUATOR MODE and INITIALIZE VALUATOR for more information. See the *Available Devices* section of INITIALIZE VALUATOR for a description of the available valuator devices.

The request suspends PHIGS until the specified device is triggered or the operator performs a BREAK. A BREAK is performed by typing the CONTROL and D keys simultaneously (^D).

**C Input Parameters**

*ws* Workstation identifier. An integer specifying the workstation with which the requested valuator device is associated.

*dev* The device number of the requested valuator device. See the *Available Devices* section of INITIALIZE VALUATOR for a description of the available valuator devices.

**C Output Parameter**

*in_status*

A pointer to the location to store the request status. Pin_status is defined in phigs.h as follows:

```
typedef enum {
        PIN_STATUS_NONE,
        PIN_STATUS_OK,
        PIN_STATUS_NO_IN
} Pin_status;
```

*value* A pointer to the location to store the valuator value.

**Execution**

REQUEST VALUATOR requests a logical input value from the specified valuator device. The specified device must be in request mode. See SET VALUATOR MODE for more information.

REQUEST VALUATOR creates a measure process for the specified device and suspends PHIGS until the device is triggered or the operator issues a BREAK. A BREAK is generated by the operator by depressing the CONTROL and D keys simultaneously (^D).

If a BREAK occurs, a status of PIN_STATUS_NO_IN is returned and the valuator data is undefined. If a status of PIN_STATUS_OK is returned, the valuator data is available. In either case, the measure process is then terminated and PHIGS processing resumes.

See INITIALIZE VALUATOR for a description of the available valuator devices.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

251 Ignoring function, the function requires the input device to be in REQUEST mode

**SEE ALSO**

INITIALIZE VALUATOR (3P)
SET VALUATOR MODE (3P)

**NAME**

RESTORE MODELLING CLIPPING VOLUME – create structure element to restore the modelling clipping volume

**SYNOPSIS**

**C Syntax**

**void**
**prestore_model_clip_vol ()**

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

RESTORE MODELLING CLIPPING VOLUME creates a structure element to restore the modelling clipping volume, undoing the effect of SET MODELLING CLIPPING VOLUME and SET MODELLING CLIPPING VOLUME 3 elements in this structure.

**Execution**

If the current edit mode is INSERT, a RESTORE MODELLING CLIPPING VOLUME element is inserted into the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, the new element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new RESTORE MODELLING CLIPPING VOLUME element.

During traversal, a RESTORE MODELLING CLIPPING VOLUME element resets the current modelling clipping volume to the value of that attribute inherited by the structure being traversed. The attribute may have been modified by traversal of SET MODELLING CLIPPING VOLUME and/or SET MODELLING CLIPPING VOLUME 3 elements.

Modelling clipping to the volume controlled by these three elements is enabled and disabled by SET MODELLING CLIPPING INDICATOR.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

INQUIRE MODELLING CLIPPING FACILITIES (3P)
SET MODELLING CLIPPING INDICATOR (3P)
SET MODELLING CLIPPING VOLUME (3P)
SET MODELLING CLIPPING VOLUME 3 (3P)

**NAME**

RETRIEVE ALL STRUCTURES – retrieve all structures from an archive file

**SYNOPSIS**

**C Syntax**

```
void
pret_all_structs ( archive_id )
Pint            archive_id;          archive identifier
```

**Required PHIGS Operating States**

(PHOP, *, *, AROP)

**DESCRIPTION**

**Purpose**

Use RETRIEVE ALL STRUCTURES to retrieve all structures from the specified open archive file into the Central Structure Store (CSS).

**C Input Parameter**

*archive_id*

The archive identifier specifying the open archive file from which to read.

**Execution**

All structures in the specified open archive file are copied into the CSS. If any of the structures to be retrieved currently exist in the CSS, the conflict is resolved as follows:

- If the retrieval conflict resolution flag is MAINTAIN, then the conflicting structure will not be copied into the CSS (its contents are maintained).
- If the retrieval conflict resolution flag is UPDATE, then the conflicting structure in the CSS will be overwritten (its contents are updated).
- If the retrieval conflict resolution flag is ABANDON, then no structures will be copied to the CSS.

The retrieval conflict resolution flag is set by the SET CONFLICT RESOLUTION subroutine.

If the currently open structure must be overwritten because of a conflict while the retrieval conflict resolution flag is UPDATE, then it is closed, its contents are overwritten, and it is re-opened. The element pointer will be set to point to the last element.

**ERRORS**

007 Ignoring function, function requires state (PHOP, *, *, AROP)

404 Ignoring function, the specified archive file is not open

405 Ignoring function, name conflict occurred while conflict resolution flag has value ABANDON

**SEE ALSO**

OPEN ARCHIVE FILE (3P)
ARCHIVE ALL STRUCTURES (3P)
SET CONFLICT RESOLUTION (3P)
INQUIRE ALL CONFLICTING STRUCTURES (3P)
RETRIEVE STRUCTURE IDENTIFIERS (3P)
RETRIEVE STRUCTURE NETWORKS (3P)
RETRIEVE STRUCTURES (3P)
DELETE ALL STRUCTURES FROM ARCHIVE (3P)

**NAME**

RETRIEVE PATHS TO ANCESTORS – retrieve the ancestors of the specified structure from an archive file

**SYNOPSIS**

**C Syntax**

**void**
**pret_paths_ances ( ar_id, struct_id, po, pd, store, paths, status )**

| | | |
|---|---|---|
| **Pint** | **ar_id;** | *archive identifier* |
| **Pint** | **struct_id;** | *structure identifier* |
| **Ppath_order** | **po;** | *path order* |
| **Pint** | **pd;** | *path depth* |
| **Pstore** | **store;** | *handle to store object* |
| **Pelem_ref_list_list** | ****paths;** | *OUT structure path list* |
| **Pint** | ***status;** | *OUT status of retrieval* |

**Required PHIGS Operating States**

(PHOP, *, *, AROP)

**DESCRIPTION**

**Purpose**

Use RETRIEVE PATHS TO ANCESTORS to determine the path or paths in the specified archive file which reference the specified structure. It performs the same function in PEX-SI as RETRIEVE PATHS TO ANCESTORS does in standard PHIGS.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing data associated with the device state. This buffer is passed as the *store* argument.

The store buffer is a data area managed by PHIGS. While the application is responsible for creating the inital buffer through a call to STORE, PHIGS manages this area such that there is sufficient memory for the specific inquiry. The data record within the store buffer is accessed by the pointer pointed to by *paths*.

*ar_id* Identifier of the archive file from which to retrieve structure path data.

*struct_id*
Identifier of the structure whose ancestors are to be retrieved.

*po* Order in which the path lists are to be returned. Valid values defined in phigs.h are:

PORDER_TOP_FIRST
PORDER_BOTTOM_FIRST

*pd* Depth (maximum number of references) of path lists to return.

*store* The memory buffer PHIGS is to use for storing the information returned. This buffer must exist prior to calling this function (see CREATE STORE (3P)).

**C Output Parameters**

*paths* A pointer to a structure Pelem_ref_list_list, containing a list of the specified structure's structure path lists. Pelem_ref_list_list is defined in phigs.h as follows:

```
typedef struct{
        Pint            num_elem_ref_lists;   /* number of execute reference lists */
        Pelem_ref_list  *elem_ref_lists;      /* list of execute reference lists */
} Pelem_ref_list_list;
```

The *num_elem_ref_lists* component specifies the number of structure path lists, or execute reference lists.

The *elem_ref_lists* component is a pointer to a list, *num_elem_ref_lists* long, of Pelem_ref_list

structures, containing the specified structure's execute reference lists. Pelem_ref_list is defined in phigs.h as follows:

```
typedef struct{
        Pint        num_elem_refs;   /* number of execute references */
        Pelem_ref   *elem_refs;      /* list of execute references */
} Pelem_ref_list;
```

*elem_refs* is a pointer to a list *num_elem_refs* long of Pelem_ref structures containing the structure identifier and element number of each execute reference structure element in the execute reference list. Pelem_ref is defined in phigs.h as follows:

```
typedef struct {
        Pint    struct_id;    /* structure identifier */
        Pint    elem_pos;     /* element number */
} Pelem_ref;
```

*status* An integer used for returning the success or failure of the routine.

**Execution**

When RETRIEVE PATHS TO ANCESTORS is called, *structure path list* is filled with list(s) identifying the EXECUTE STRUCTURE structure elements that refer to *structure identifier*, in the order of traversal. These EXECUTE STRUCTURE structure elements are represented as (*structure identifier*, element position) pairs, giving the parent structure identifier and the position of the EXECUTE STRUCTURE structure element. Whenever *structure identifier* itself is included at the bottom of a returned path of ancestors, it is represented by a (*structure identifier*, element position) pair with an element position of 0. *path order* and *path depth* are used to determine the portion of each path returned. The number of references returned in each path is specified by *path depth*; a *path depth* of 0 returns all the references in the path. In case of truncation, *path order* determines whether the head (*top first*) or the tail (*bottom first*) portion of a path is returned. If a path truncation results in two or more partial paths with the same set of element references, only one of the identical path portions is returned.

For example, specifying *top first* and a depth of 0 returns all paths to *structure identifier*. Specifying *top first* and a depth of 1 returns the root of each structure network that references *structure identifier*. Specifying *bottom first* and a depth of 2 returns all the parents of *structure identifier*.

**ERRORS**

007 Ignoring function, function requires state (PHOP, *, *, AROP)

201 Ignoring function, the specified structure does not exist

207 Ignoring function, the specified path depth is less than zero

**SEE ALSO**

OPEN ARCHIVE FILE (3P)
ARCHIVE STRUCTURE NETWORKS (3P)
SET CONFLICT RESOLUTION (3P)
INQUIRE ALL CONFLICTING STRUCTURES (3P)
INQUIRE CONFLICTING STRUCTURES IN NETWORK (3P)
RETRIEVE PATHS TO DESCENDANTS (3P)
RETRIEVE STRUCTURE NETWORKS (3P)
DELETE STRUCTURE NETWORKS FROM ARCHIVE (3P)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

RETRIEVE PATHS TO DESCENDANTS – retrieve the descendants of the specified structure from an archive file

**SYNOPSIS**

**C Syntax**

**void**
**pret_paths_descs ( ar_id, struct_id, po, pd, store, paths, status )**

| | | |
|---|---|---|
| **Pint** | **ar_id;** | *archive identifier* |
| **Pint** | **struct_id;** | *structure identifier* |
| **Ppath_order** | **po;** | *path order* |
| **Pint** | **pd;** | *path depth* |
| **Pstore** | **store;** | *handle to Store object* |
| **Pelem_ref_list_list** | ****paths;** | *OUT path list* |
| **Pint** | ***status;** | *OUT status of retrieval* |

**Required PHIGS Operating States**

(PHOP, *, *, AROP)

**DESCRIPTION**

**Purpose**

Use RETRIEVE PATHS TO DESCENDANTS to determine the path or paths in the specified archive file that are referenced by the specified structure. It performs the same function in PEX-SI as RETRIEVE PATHS TO DESCENDANTS does in standard PHIGS.

**C Input Parameters**

Applications using the C binding must create a buffer to be used by this function as memory space for storing data associated with the device state. This buffer is passed as the *store* argument.

The store buffer is a data area managed by PHIGS. While the application is responsible for creating the inital buffer through a call to STORE, PHIGS manages this area such that there is sufficient memory for the specific inquiry. The data record within the store buffer is accessed by the pointer pointed to by *paths*.

*ar_id* Identifier of the archive file from which to retrieve structure path data.

*struct_id*
Identifier of the structure whose descendants are to be retrieved.

*po* Order in which the path lists are to be returned. Valid values defined in phigs.h are:

PORDER_TOP_FIRST
PORDER_BOTTOM_FIRST

*pd* Depth (maximum number of references) of path lists to return.

*store* The memory buffer PHIGS is to use for storing the information returned. This buffer must exist prior to calling this function (see CREATE STORE (3P)).

**C Output Parameters**

*paths* A pointer to a structure, Pelem_ref_list_list, containing a list of the specified structure's structure path lists. Pelem_ref_list_list is defined in phigs.h as follows:

```
typedef struct{
        Pint            num_elem_ref_lists;     /* number of execute reference lists */
        Pelem_ref_list  *elem_ref_lists;        /* list of execute reference lists */
} Pelem_ref_list_list;
```

The *num_elem_ref_lists* component specifies the number of structure path lists, or execute reference lists.

The *elem_ref_lists* component is a pointer to a list, *num_elem_ref_lists* long, of Pelem_ref_list structures, containing the specified structure's execute reference lists. Pelem_ref_list is defined in

phigs.h as follows:

```
typedef struct{
        Pint        num_elem_refs;   /* number of execute references */
        Pelem_ref   *elem_refs;      /* list of execute references */
} Pelem_ref_list;
```

*elem_refs* is a pointer to a list *num_elem_refs* long of Pelem_ref structures containing the structure identifier and element number of each execute reference structure element in the execute reference list. Pelem_ref is defined in phigs.h as follows:

```
typedef struct {
        Pint    struct_id;    /* structure identifier */
        Pint    elem_pos;     /* element number */
} Pelem_ref;
```

*status* An integer used for returning the success or failure of the routine.

**Execution**

When RETRIEVE PATHS TO DESCENDANTSS is called, *structure path list* is filled with list(s) identifying the EXECUTE STRUCTURE structure elements that are referenced by *structure identifier*, in the order of traversal. These EXECUTE STRUCTURE structure elements are represented as (structure identifier, element position) pairs, giving the parent structure identifier and the position of the EXECUTE STRUCTURE structure element. The bottom-most element of a structure network, if included in a returned path, is indicated by a (structure identifier, element position) pair containing the identifier of the bottom-most structure and an element position of 0. *path order* and *path depth* are used to determine the portion of each path returned. The number of references returned in each path is specified by *path depth*; a *path depth* of 0 returns all the references in the path. In case of truncation, *path order* determines whether the head (*top first*) or the tail (*bottom first*) portion of a path is returned. If a path truncation results in two or more partial paths with the same set of element references, only one of the identical path portions is returned.

For example, specifying *top first* and a depth of 0 returns all paths from *structure identifier*. Specifying *top first* and a depth of 1 returns each EXECUTE STRUCTURE structure element in *structure identifier* as a separate path list. Specifying *bottom first* and a depth of 1 returns all the bottom-most structures of the network.

**ERRORS**

007 Ignoring function, function requires state (PHOP, *, *, AROP)

201 Ignoring function, the specified structure does not exist

207 Ignoring function, the specified path depth is less than zero

**SEE ALSO**

OPEN ARCHIVE FILE (3P)
ARCHIVE STRUCTURE NETWORKS (3P)
SET CONFLICT RESOLUTION (3P)
INQUIRE ALL CONFLICTING STRUCTURES (3P)
INQUIRE CONFLICTING STRUCTURES IN NETWORK (3P)
RETRIEVE PATHS TO ANCESTORS (3P)
RETRIEVE STRUCTURE NETWORKS (3P)
DELETE STRUCTURE NETWORKS FROM ARCHIVE (3P)
CREATE STORE (3P)
DELETE STORE (3P)

**NAME**

RETRIEVE STRUCTURE IDENTIFIERS – retrieve a list of structure identifiers from an archive file

**SYNOPSIS**

**C Syntax**

**void**
**pret_struct_ids ( archive_id, max_ids, start, ids, actual_ids )**

| | | |
|---|---|---|
| **Pint** | **archive_id;** | *archive identifier* |
| **Pint** | **max_ids;** | *size of appl. archive id list* |
| **Pint** | **start;** | *start position of ids* |
| **Pint_list** | ***ids;** | *OUT list of structure ids* |
| **Pint** | ***actual_ids;** | *OUT actual number of ids in PHIGS* |

**Required PHIGS Operating States**

(PHOP, *, *, AROP)

**DESCRIPTION**

**Purpose**

Use RETRIEVE STRUCTURE IDENTIFIERS to retrieve a list of the identifiers of all structures archived in the specified open archive file.

**C Input Parameters**

*archive_id*

The archive identifier specifying the open archive file to read from.

*max_ids*

The number of *ints* items in the *ids* output parameter for which the application has allocated memory. *max_ids* is the number of list elements (archive structure identifiers) that the system can return in *integers*. If a value of 0 is used here, no data will be returned in the *ints* list, but the total number of structure identifiers in the archive file will be returned in *actual_ids*.

*start* Starting position in the list of archive structure identifiers at which to begin the inquiry. The elements of the list of structure identifiers, beginning with the item number specified by *start*, are copied sequentially into *ints* until *ints* is full or all the structure identifiers have been copied.

**C Output Parameters**

*ids* A pointer to a Pint_list structure in which the list of structure identifiers in the specified open archive file is returned. The Pint_list structure is defined in phigs.h as follows:

```
typedef struct {
        Pint    num_list;   /* number of Pints in list */
        Pint    *ints;      /* list of integers */
} Pint_list;
```

The *num_ints* component specifies the number of structure identifiers in the list.

The *ints* component is a pointer to a list, *num_ints* long, of the structure identifiers.

The application must allocate memory for *max_ids* elements in the *ints* list.

*actual_ids*

A pointer to an integer in which the total number of structure identifiers in the specified archive file is returned. This is the value required for *max_ids* if all structure identifiers are to be returned.

**ERRORS**

007 Ignoring function, function requires state (PHOP, *, *, AROP)

404 Ignoring function, the specified archive file is not open

**SEE ALSO**

OPEN ARCHIVE FILE (3P)
ARCHIVE STRUCTURES (3P)

INQUIRE ALL CONFLICTING STRUCTURES (3P)
SET CONFLICT RESOLUTION (3P)
RETRIEVE STRUCTURES (3P)
DELETE STRUCTURES FROM ARCHIVE (3P)

**NAME**

RETRIEVE STRUCTURE NETWORKS – retrieve specified structure networks from an archive file

**SYNOPSIS**

**C Syntax**

```
void
pret_struct_nets ( archive_id, struct_ids )
Pint        archive_id;     archive identifier
Pint_list   *struct_ids;    list of structure identifiers
```

**Required PHIGS Operating States**

(PHOP, *, *, AROP)

**DESCRIPTION**

**Purpose**

Use RETRIEVE STRUCTURE NETWORKS to retrieve a list of structure networks from the specified open archive file into the Central Structure Store (CSS).

**C Input Parameters**

*archive_id*

The archive identifier specifying the open archive file from which to read.

*struct_ids*

A pointer to a Pint_list structure containing the list of the root structure identifiers of the networks to be retrieved. The Pint_list structure is defined in phigs.h as follows:

```
typedef struct {
        Pint    num_ints;       /* number of Pints in list */
        Pint    *ints;          /* list of integers */
} Pint_list;
```

The *num_ints* component specifies the number of structure identifiers in the list.

The *ints* component is a pointer to a list, *num_ints* long, of the structure identifiers.

**Execution**

The structures belonging to the specified networks are copied from the indicated open archive file into the Central Structure Store (CSS). If any of the structures to be retrieved currently exist in the CSS, the conflict is resolved as follows:

- If the retrieval conflict resolution flag is MAINTAIN, then the conflicting structure will not be copied into the CSS (its contents are maintained).
- If the retrieval conflict resolution flag is UPDATE, then the conflicting structure in the CSS will be overwritten (its contents are updated).
- If the retrieval conflict resolution flag is ABANDON, then no structures will be copied to the CSS.

The retrieval conflict resolution flag is set by the SET CONFLICT RESOLUTION subroutine.

If the currently open structure must be overwritten because of a conflict while the retrieval conflict resolution flag is UPDATE, then it is closed, its contents are overwritten, and it is re-opened. The element pointer will be set to point to the last element.

If any of the structures to be retrieved do not exist in the archive file, an empty structure is created in the CSS, a warning is generated, and the retrieval operation continues for the remaining structures.

**ERRORS**

007 Ignoring function, function requires state (PHOP, *, *, AROP)

404 Ignoring function, the specified archive file is not open

405 Ignoring function, name conflict occurred while conflict resolution flag has value ABANDON

408 Warning, some of the specified structures do not exist on the archive file. PHIGS will create empty structures in their places

**SEE ALSO**

OPEN ARCHIVE FILE (3P)
ARCHIVE STRUCTURE NETWORKS (3P)
SET CONFLICT RESOLUTION (3P)
INQUIRE CONFLICTING STRUCTURES IN NETWORK (3P)
RETRIEVE ALL STRUCTURES (3P)
RETRIEVE PATHS TO ANCESTORS (3P)
RETRIEVE PATHS TO DESCENDANTS (3P)
DELETE STRUCTURE NETWORKS FROM ARCHIVE (3P)

**NAME**

RETRIEVE STRUCTURES – retrieve specified structures from an archive file

**SYNOPSIS**

**C Syntax**

```
void
pret_structs ( archive_id, struct_ids )
Pint        archive_id;      archive identifier
Pint_list   *struct_ids;     list of structure identifiers
```

**Required PHIGS Operating States**

(PHOP, *, *, AROP)

**DESCRIPTION**

**Purpose**

Use RETRIEVE STRUCTURES to retrieve a list of structures from the specified open archive file into the CSS.

**C Input Parameters**

*archive_id*

The archive identifier specifying the open archive file to read from.

*struct_ids*

A pointer to a Pint_list structure containing the list of structure identifiers to be retrieved. The Pint_list structure is defined in phigs.h as follows:

```
typedef struct {
        Pint    num_ints;    /* number of Pints in list */
        Pint    *ints;       /* list of integers */
} Pint_list;
```

The *num_ints* component specifies the number of structure identifiers in the list.

The *ints* component is a pointer to a list, *num_ints* long, of the structure identifiers.

**Execution**

The specified structures are copied from the indicated open archive file into the Central Structure Store (CSS). If any of the structures to be retrieved currently exist in the CSS, the conflict is resolved as follows:

- If the retrieval conflict resolution flag is MAINTAIN, then the conflicting structure will not be copied into the CSS (its contents are maintained).
- If the retrieval conflict resolution flag is UPDATE, then the conflicting structure in the CSS will be overwritten (its contents are updated).
- If the retrieval conflict resolution flag is ABANDON, then no structures will be copied to the CSS.

The retrieval conflict resolution flag is set by the SET CONFLICT RESOLUTION subroutine.

If the currently open structure must be overwritten because of a conflict while the retrieval conflict resolution flag is UPDATE, then it is closed, its contents are overwritten, and it is re-opened. The element pointer will be set to point to the last element.

If any of the structures to be retrieved do not exist in the archive file, then an empty structure is created in the CSS, a warning is generated, and the retrieval operation continues for the remaining structures.

**ERRORS**

007 Ignoring function, function requires state (PHOP, *, *, AROP)

404 Ignoring function, the specified archive file is not open

405 Ignoring function, name conflict occurred while conflict resolution flag has value ABANDON

408 Warning, some of the specified structures do not exist on the archive file. PHIGS will create empty structures in their places

**SEE ALSO**

OPEN ARCHIVE FILE (3P)
ARCHIVE STRUCTURES (3P)
SET CONFLICT RESOLUTION (3P)
INQUIRE ALL CONFLICTING STRUCTURES (3P)
RETRIEVE ALL STRUCTURES (3P)
RETRIEVE STRUCTURE IDENTIFIERS (3P)
RETRIEVE STRUCTURE NETWORKS (3P)
DELETE STRUCTURES FROM ARCHIVE (3P)

**NAME**

ROTATE – calculate a 2D transformation matrix to perform a specified 2D rotation

**SYNOPSIS**

**C Syntax**

```
void
protate ( angle, error_ind, m)
Pfloat      angle;          rotation angle
Pint        *error_ind;     OUT error indicator
Pmatrix     m;              OUT transformation matrix
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use ROTATE to generate a 2D homogeneous $(3 \times 3)$ transformation matrix that performs a 2D rotation.

The returned matrix may be passed as an argument to SET LOCAL TRANSFORMATION or SET GLOBAL TRANSFORMATION to modify the modelling transformation that is applied to output primitives during traversal.

**C Input Parameter**

*angle* The angle of the rotation in radians. Positive angles indicate counterclockwise rotation, and negative angles indicate clockwise rotation.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*m* The $3 \times 3$ transformation matrix that performs the specified rotation. Pmatrix is defined in phigs.h as follows:

```
typedef Pfloat Pmatrix[3][3];
```

**Execution**

ROTATE returns a 2D homogeneous $(3 \times 3)$ transformation matrix that performs the rotation specified by *angle*. The rotation is specified in radians. The rotation is relative to the origin of the current modelling coordinate system.

**ERRORS**

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

**SEE ALSO**

ROTATE X (3P)
ROTATE Y (3P)
ROTATE Z (3P)
SET LOCAL TRANSFORMATION 3 (3P)
BUILD TRANSFORMATION MATRIX 3 (3P)
COMPOSE MATRIX (3P)
TRANSFORM POINT (3P)

**NAME**

ROTATE X – calculate a 3D transformation matrix to perform a specified rotation about the *x* axis

**SYNOPSIS**

**C Syntax**

```
void
protate_x ( angle, error_ind, m)
Pfloat      angle;          /* rotation angle */
Pint        *error_ind;     /* OUT error indicator */
Pmatrix3    m;              /* OUT transformation matrix */
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use ROTATE X to generate a 3D homogeneous $(4 \times 4)$ transformation matrix that performs a 3D rotation about the *x* axis.

The returned matrix may be passed as an argument to SET LOCAL TRANSFORMATION 3 or SET GLOBAL TRANSFORMATION 3 to modify the modelling transformation that is applied to output primitives during traversal.

**C Input Parameter**

*angle* The angle of the rotation in radians. Positive angles indicate counterclockwise rotation, and negative angles indicate clockwise rotation.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*m* The $4 \times 4$ homogeneous transformation matrix that performs the specified rotation. Pmatrix3 is defined in phigs.h as follows:

```
typedef Pfloat Pmatrix3[4][4];
```

**Execution**

ROTATE X returns the 3D homogeneous $(4 \times 4)$ transformation matrix that performs the rotation specified by *angle* about the *x* axis. The rotation is specified in radians and is relative to the origin of the current modelling coordinate system.

**ERRORS**

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

**SEE ALSO**

ROTATE (3P)
SET LOCAL TRANSFORMATION 3 (3P)
BUILD TRANSFORMATION MATRIX 3 (3P)
COMPOSE MATRIX 3 (3P)
TRANSFORM POINT 3 (3P)

**NAME**

ROTATE Y – calculate a 3D transformation matrix to perform a specified rotation about the *y* axis

**SYNOPSIS**

**C Syntax**

```
void
protate_y ( angle, error_ind, m)
Pfloat      angle;          rotation angle
Pint        *error_ind;     OUT error indicator
Pmatrix3    m;              OUT transformation matrix
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use ROTATE Y to generate a 3D homogeneous $(4 \times 4)$ transformation matrix that performs a 3D rotation about the *y* axis.

The returned matrix may be passed as an argument to SET LOCAL TRANSFORMATION 3 or SET GLOBAL TRANSFORMATION 3 to modify the modelling transformation that is applied to output primitives during traversal.

**C Input Parameter**

*angle* The angle of the rotation in radians. Positive angles indicate counterclockwise rotation, and negative angles indicate clockwise rotation.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*m* The $4 \times 4$ homogeneous transformation matrix that performs the specified rotation. Pmatrix3 is defined in phigs.h as follows:

typedef Pfloat Pmatrix3[4][4];

**Execution**

ROTATE Y returns the 3D homogeneous $(4 \times 4)$ transformation matrix that performs the rotation specified by *angle* about the *y* axis. The rotation is specified in radians and is relative to the origin of the current modelling coordinate system.

**ERRORS**

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

**SEE ALSO**

ROTATE (3P)
SET LOCAL TRANSFORMATION 3 (3P)
BUILD TRANSFORMATION MATRIX 3 (3P)
COMPOSE MATRIX 3 (3P)
TRANSFORM POINT 3 (3P)

**NAME**

ROTATE Z – calculate a 3D transformation matrix to perform a specified rotation about the *z* axis

**SYNOPSIS**

**C Syntax**

```
void
protate_z ( angle, error_ind, m)
Pfloat      angle;          rotation angle
Pint        *error_ind;     OUT error indicator
Pmatrix3    m;              OUT transformation matrix
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use ROTATE Z to generate a 3D homogeneous $(4 \times 4)$ transformation matrix that performs a 3D rotation about the *z* axis.

The returned matrix can be passed as an argument to SET LOCAL TRANSFORMATION 3 or SET GLOBAL TRANSFORMATION 3 to modify the modelling transformation that is applied to output primitives during traversal.

**C Input Parameter**

*angle* The angle of the rotation in radians. Positive angles indicate counterclockwise rotation, and negative angles indicate clockwise rotation.

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*m* The $4 \times 4$ homogeneous transformation matrix that performs the specified rotation. Pmatrix3 is defined in phigs.h as follows:

```
typedef Pfloat Pmatrix3[4][4];
```

**Execution**

ROTATE Z returns the 3D homogeneous $(4 \times 4)$ transformation matrix that performs the rotation specified by *angle* about the *z* axis. The rotation is specified in radians and is relative to the origin of the current modelling coordinate system.

**ERRORS**

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

**SEE ALSO**

ROTATE (3P)
SET LOCAL TRANSFORMATION 3 (3P)
BUILD TRANSFORMATION MATRIX 3 (3P)
COMPOSE MATRIX 3 (3P)
TRANSFORM POINT 3 (3P)

**NAME**

SAMPLE CHOICE – sample the current measure of a choice device

**SYNOPSIS**

**C Syntax**

**void**
**psample_choice ( ws, dev, choice_in_status, choice )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **dev;** | *choice device number* |
| **Pin_status** | ***choice_in_status;** | *OUT choice input status* |
| **Pint** | ***choice;** | *OUT choice* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use SAMPLE CHOICE to sample the current measure of the specified choice device without waiting for the device trigger to fire. The device must be in SAMPLE mode. See SET CHOICE MODE and INITIALIZE CHOICE for more information.

**C Input Parameters**

*ws* Workstation identifier. An integer specifying the workstation with which the specified choice device is associated.

*dev* The device number of the choice device to be sampled. See the *Available Devices* section of INITIALIZE CHOICE for a description of the available devices.

**C Output Parameter**

*choice_in_status*

A pointer to a location to store the status of the sample. Pin_status is defined in phigs.h as:

```
typedef enum {
        PIN_STATUS_NONE,
        PIN_STATUS_OK,
        PIN_STATUS_NO_IN
} Pin_status;
```

*choice* A pointer to a location to store the results of the choice.

**Execution**

SAMPLE CHOICE samples the specified choice device. The device must be in SAMPLE mode. See SET CHOICE MODE and INITIALIZE CHOICE for more information.

When a device is set to SAMPLE mode, a measure process is created for the device and the device is activated. When SAMPLE CHOICE is called, the current value of the measure process for the specified device is retrieved without waiting for the device's trigger to fire.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

252 Ignoring function, the function requires the input device to be in SAMPLE mode

**SEE ALSO**

SET CHOICE MODE (3P)
INITIALIZE CHOICE 3 (3P)
REQUEST CHOICE (3P)

GET CHOICE (3P)
INQUIRE CHOICE DEVICE STATE (3P)

**NAME**

SAMPLE LOCATOR – sample the current measure of a locator device

**SYNOPSIS**

**C Syntax**

**void**
**psample_loc ( ws, dev, view_ind, loc_pos )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **dev;** | *locator device number* |
| **Pint** | ***view_ind;** | *OUT view index* |
| **Ppoint** | ***loc_pos;** | *OUT locator data* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use SAMPLE LOCATOR to sample the current measure of the specified locator device without waiting for operator input. The device must be in SAMPLE mode. See SET LOCATOR MODE and INITIALIZE LOCATOR for more information.

**C Input Parameters**

*ws* Workstation identifier. An integer specifying the workstation with which the specified locator device is associated.

*dev* The device number of the locator device to be sampled. See the *Available Devices* section of INITIALIZE LOCATOR for a description of the available devices.

**C Output Parameters**

*view_ind*

The index of the view representation used to transform the Normalized Projection Coordinate (NPC) position to a World Coordinates (WC) position.

*loc_pos* Ppoint is defined in phigs.h as follows:

```
typedef struct {
        Pfloat    x;    /* x coordinate */
        Pfloat    y;    /* y coordinate */
} Ppoint;
```

**Execution**

SAMPLE LOCATOR samples the specified locator device. The device must be in SAMPLE mode. See SET LOCATOR MODE and INITIALIZE LOCATOR for more information.

When a device is set to SAMPLE mode, a measure process is created for the device and the device is activated. When SAMPLE LOCATOR is called, the current value of the measure process for the specified device is retrieved without waiting for further input from the operator.

A LOCATOR device measure consists of a *position* and a *view index*. The *position* is the WC point corresponding to the position on the workstation selected by the operator. The *view index* is the index of the view used to transform *position* from Device Coordinates (DC) to WC. See INITIALIZE LOCATOR and SET VIEW TRANSFORMATION INPUT PRIORITY for a description of how this view representation is determined. The workstation transform is used to transform the DC position to a NPC position.

**Note:** The 2D and 3D locator measure processes are the same except that the 2D process discards the *z* coordinate. The 3D version of this function, SAMPLE LOCATOR 3, can be used if the value of the *z* coordinate is needed.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

252 Ignoring function, the function requires the input device to be in SAMPLE mode

**SEE ALSO**

INITIALIZE LOCATOR (3P)
SET LOCATOR MODE (3P)
REQUEST LOCATOR 3 (3P)
GET LOCATOR (3P)
INQUIRE LOCATOR DEVICE STATE (3P)
SET VIEW TRANSFORMATION INPUT PRIORITY (3P)
SAMPLE LOCATOR 3 (3P)

**NAME**

SAMPLE LOCATOR 3 – sample the current measure of a locator device

**SYNOPSIS**

**C Syntax**

**void**
**psample_loc3 ( ws, dev, view_ind, loc_pos )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **dev;** | *locator device number* |
| **Pint** | ***view_ind;** | *OUT view index* |
| **Ppoint3** | ***loc_pos;** | *OUT locator position* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use SAMPLE LOCATOR 3 to sample the current measure of the specified locator device without waiting for input from the operator. The device must be in SAMPLE mode. See SET LOCATOR MODE and INITIALIZE LOCATOR 3 for more information.

**C Input Parameters**

*ws* Workstation identifier. An integer specifying the workstation with which the specified locator device is associated.

*dev* The device number of the locator device to be sampled. See the *Available Devices* section of INITIALIZE LOCATOR 3 for a description of the available devices.

**C Output Parameters**

*view_ind*

The index of the view representation used to transform the Normalized Projection Coordinate (NPC) position to a World Coordinates (WC) position.

*loc_pos* Ppoint3 is defined in phigs.h as follows:

```
typedef struct {
        Pfloat    x;    /* x coordinate */
        Pfloat    y;    /* y coordinate */
        Pfloat    z;    /* z coordinate */
} Ppoint3;
```

**Execution**

SAMPLE LOCATOR 3 samples the specified locator device. The device must be in SAMPLE mode. See SET LOCATOR MODE and INITIALIZE LOCATOR 3 for more information.

When a device is set to SAMPLE mode, a measure process is created for the device and the device is activated. When SAMPLE LOCATOR 3 is called, the current value of the measure process for the specified device is retrieved without waiting for further input from the operator.

A LOCATOR device measure consists of a *position* and a *view index*. The *position* is the WC point corresponding to the position on the workstation selected by the operator. The *view index* is the index of the view used to transform *position* from Device Coordinates (DC) to WC. See INITIALIZE LOCATOR 3 and SET VIEW TRANSFORMATION INPUT PRIORITY for a description of how this view representation is determined. The workstation transform is used to transform the DC position to a NPC position.

**Note:** The 2D and 3D locator measure processes are the same except that the 2D process discards the *z* coordinate. The 2D version of this function, SAMPLE LOCATOR, can be used if the value of the *z* coordinate is not needed.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

252 Ignoring function, the function requires the input device to be in SAMPLE mode

**SEE ALSO**

INITIALIZE LOCATOR 3 (3P)
SET LOCATOR MODE (3P)
REQUEST LOCATOR 3 (3P)
GET LOCATOR 3 (3P)
INQUIRE LOCATOR DEVICE STATE 3 (3P)
SET VIEW TRANSFORMATION INPUT PRIORITY (3P)
SAMPLE LOCATOR (3P)

**NAME**

SAMPLE PICK – sample the current measure of a pick device

**SYNOPSIS**

**C Syntax**

```
void
psample_pick ( ws, dev, depth, pick_in_status, rpick )
Pint            ws;             workstation identifier
Pint            dev;            pick device number
Pint            depth;          max. depth of returned path
Pin_status      *pick_in_status; OUT pick input status
Ppick_path      *rpick;         OUT pick path
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use SAMPLE PICK to sample the current measure of the specified pick device without waiting for input from the operator. The device must be in SAMPLE mode. See SET PICK MODE and INITIALIZE PICK for more information.

**C Input Parameters**

*ws* Workstation identifier. An integer specifying the workstation with which the specified pick device is associated.

*dev* The device number of the pick device to be sampled. See the *Available Devices* section of INITIALIZE PICK for a description of the available devices.

*depth* The maximum number of path elements to return.

**C Output Parameter**

*pick_in_status*

Pin_status is defined in phigs.h as:

```
typedef enum {
                PIN_STATUS_NONE,
                PIN_STATUS_OK,
                PIN_STATUS_NO_IN
} Pin_status;
```

*rpick* Ppick_path is defined in phigs.h as:

```
typedef struct {
        Pint            depth;      /* pick path depth */
        Ppick_path_elem *path_list; /* pick path list */
} Ppick_path;
```

*depth* indicates the number of elements in the measure's path. This is the depth value contained in the device's current measure. It is not affected by the *maximum depth to return* parameter. Thus the number of elements returned in *path_list* may be less than *depth*.

*path_list* is the array of path elements defining the location of the picked primitive in the Central Structure Store (CSS). **This array must be allocated by the calling program and the array pointer assigned to this field before calling this function.** The array must be at least of length *maximum depth to return*. Ppick_path_elem is defined in phigs.h as:

```
typedef struct {
        Pint    struct_id;    /* structure identifier */
        Pint    pick_id;      /* pick identifier */
        Pint    elem_pos;     /* element number */
} Ppick_path_elem;
```

*struct_id*, *pick_id*, and *elem_pos* are the structure identifier, pick identifier, and element number, respectively, of each element in the path.

**Execution**

SAMPLE PICK samples the specified pick device. The device must be in SAMPLE mode. See SET PICK MODE and INITIALIZE PICK for more information.

When a device is set to SAMPLE mode, a measure process is created for the device and the device is activated. When SAMPLE PICK is called, the current value of the measure process for the specified device is retrieved without waiting for further input from the operator.

A PICK device measure consists of a *status* and a *pick path*. The *status* indicates whether or not a pick by the operator was successfully resolved. The *path* describes the location of the picked primitive, if any, in the *Central Structure Store* (CSS). The *pick filter* of a PICK device controls which output primitives on the device's workstation are pickable. By default, no output primitives are pickable. See SET PICK FILTER for more information on the *pick filter*.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

060 Ignoring function, specified workstation is not of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

252 Ignoring function, the function requires the input device to be in SAMPLE mode

**SEE ALSO**

SET PICK FILTER (3P)
SET PICK IDENTIFIER (3P)
SET PICK MODE (3P)
INITIALIZE PICK 3 (3P)
REQUEST PICK (3P)
GET PICK (3P)
INQUIRE PICK DEVICE STATE (3P)

**NAME**

SAMPLE STRING – sample current measure of a string device

**SYNOPSIS**

**C Syntax**

```
void
psample_string ( ws, dev, string )
Pint        ws;         workstation identifier
Pint        dev;        string device number
char        *string;    OUT string
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use SAMPLE STRING to sample the current measure of the specified string device without waiting for input from the operator. The device must be in SAMPLE mode. See SET STRING MODE and INITIALIZE STRING for more information.

**C Input Parameters**

*ws* Workstation identifier. An integer specifying the workstation with which the specified string device is associated.

*dev* The device number of the string device to be sampled. See the *Available Devices* section of INITIALIZE STRING for a description of the available devices.

**C Output Parameter**

*string* A pointer to a character array in which to store the STRING measure. The string is null terminated. The array should be at least as large as the buffer of the STRING device. This buffer size is set when the device is initialized.

**Execution**

SAMPLE STRING samples the specified string device. The device must be in SAMPLE mode. See SET STRING MODE and INITIALIZE STRING for more information.

When a device is set to SAMPLE mode, a measure process is created for the device and the device is activated. When SAMPLE STRING is called, the current value of the measure process for the specified device is retrieved without waiting for further input from the operator.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

252 Ignoring function, the function requires the input device to be in SAMPLE mode

**SEE ALSO**

SET STRING MODE (3P)
INITIALIZE STRING 3 (3P)
REQUEST STRING (3P)
GET STRING (3P)
INQUIRE STRING DEVICE STATE (3P)

**NAME**

SAMPLE STROKE – sample the current measure of a stroke device

**SYNOPSIS**

**C Syntax**

**void**
**psample_stroke ( ws, dev, view_ind, stroke )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **dev;** | *stroke device number* |
| **Pint** | ***view_ind** | *OUT view index* |
| **Ppoint_list** | ***stroke;** | *OUT locator postion* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use SAMPLE STROKE to sample the current measure of the specified stroke device without waiting for input from the operator. The device must be in SAMPLE mode. See SET STROKE MODE and INITIALIZE STROKE for more information.

**C Input Parameters**

*ws* Workstation identifier. An integer specifying the workstation with which the specified stroke device is associated.

*dev* The device number of the stroke device to be sampled. See the *Available Devices* section of INITIALIZE STROKE for a description of the available devices.

**C Output Parameters**

*view_ind*

The view index used to transform the Device Coordinate (DC) positions to World Coordinate (WC) points.

*stroke* Ppoint_list is defined in phigs.h as follows:

```
typedef struct {
        Pint        num_points;
        Ppoint      *points;
} Ppoint_list;
```

*num_points* is the number of points in *points*.

*points* is the array of Ppoint structures specifying the points in WC. **This array must be allocated by the calling program and the array pointer assigned to this field before calling this function.** The array must be at least as large as the buffer of the STROKE device. This buffer size is set when the device is initialized. Ppoint is defined in phigs.h as follows:

```
typedef struct {
        Pfloat    x;    /* x coordinate */
        Pfloat    y;    /* y coordinate */
} Ppoint;
```

**Execution**

SAMPLE STROKE samples the specified stroke device. The device must be in SAMPLE mode. See SET STROKE MODE and INITIALIZE STROKE for more information.

When a device is set to SAMPLE mode, a measure process is created for the device and the device is activated. When SAMPLE STROKE is called, the current value of the measure process for the specified device is retrieved without waiting for further input from the operator.

A STROKE device measure consists of a list of WC *points* and a *view index*. The *points* correspond to positions on the workstation selected by the operator. The *view index* is the index of the view used to transform these positions from DC to WC. See INITIALIZE STROKE and SET VIEW TRANSFORMATION INPUT PRIORITY for a description of how this view representation is determined. The workstation transformation is used to map the DC position to a Normalized Projection Coordinate (NPC) position.

**Note:** The 2D and 3D stroke measure processes are the same except that the 2D process discards the *z* coordinate. The 3D version of this function, SAMPLE STROKE 3, can be used if the value of the *z* coordinate is needed.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

252 Ignoring function, the function requires the input device to be in SAMPLE mode

**SEE ALSO**

INITIALIZE STROKE (3P)
SET STROKE MODE (3P)
REQUEST STROKE 3 (3P)
GET STROKE (3P)
INQUIRE STROKE DEVICE STATE (3P)
SET VIEW TRANSFORMATION INPUT PRIORITY (3P)
SAMPLE STROKE 3 (3P)

**NAME**

SAMPLE STROKE 3 – sample the current measure of a stroke device

**SYNOPSIS**

**C Syntax**

**void**
**psample_stroke3 ( ws, dev, view_ind, stroke )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **dev;** | *stroke device number* |
| **Pint** | ***view_ind;** | *OUT view index* |
| **Ppoint_list3** | ***stroke;** | *OUT stroke data* |

**Required PHIGS Operating States**

(PHOP, WSOP, ∗, ∗)

**DESCRIPTION**

**Purpose**

Use SAMPLE STROKE 3 to sample the current measure of the specified stroke device without waiting for input from the operator. The device must be in SAMPLE mode. See SET STROKE MODE and INITIALIZE STROKE 3 for more information.

**C Input Parameters**

*ws* Workstation identifier. An integer specifying the workstation with which the specified stroke device is associated.

*dev* The device number of the stroke device to be sampled. See the *Available Devices* section of INITIALIZE STROKE 3 for a description of the available devices.

**C Output Parameters**

*view_ind*

The view index used to transform the Device Coordinate (DC) positions to World Coordinate (WC) points.

*stroke* Ppoint_list3 is defined in phigs.h as follows:

```
typedef struct {
        Pint      num_points;
        Ppoint3  *points;
} Ppoint_list3;
```

*num_points* is the number of points in *points*.

*points* is the array of Ppoint3 structures specifying the points in WC. **This array must be allocated by the calling program and the array pointer assigned to this field before calling this function.** The array must be at least as large as the buffer of the STROKE device. This buffer size is set when the device is initialized. Ppoint3 is defined in phigs.h as follows:

```
typedef struct {
        Pfloat    x;    /* x coordinate */
        Pfloat    y;    /* y coordinate */
        Pfloat    z;    /* z coordinate */
} Ppoint3;
```

**Execution**

SAMPLE STROKE 3 samples the specified stroke device. The device must be in SAMPLE mode. See SET STROKE MODE and INITIALIZE STROKE 3 for more information.

When a device is set to SAMPLE mode, a measure process is created for the device and the device is activated. When SAMPLE STROKE 3 is called, the current value of the measure process for the specified device is retrieved without waiting for further input from the operator.

A STROKE device measure consists of a list of WC *points* and a *view index*. The *points* correspond to positions on the workstation selected by the operator. The *view index* is the index of the view used to transform these positions from DC to WC. See INITIALIZE STROKE 3 and SET VIEW TRANSFORMATION INPUT PRIORITY for a description of how this view representation is determined. The workstation transformation is used to map the DC position to a Normalized Projection Coordinate (NPC) position.

**Note:** The 2D and 3D stroke measure processes are the same except that the 2D process discards the *z* coordinate. The 2D version of this function, SAMPLE STROKE, can be used if the value of the *z* coordinate is not needed.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, ∗, ∗)

054 Ignoring function, the specified workstation is not open

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

252 Ignoring function, the function requires the input device to be in SAMPLE mode

**SEE ALSO**

INITIALIZE STROKE 3 (3P)
SET STROKE MODE (3P)
REQUEST STROKE 3 (3P)
GET STROKE 3 (3P)
INQUIRE STROKE DEVICE STATE 3 (3P)
SET VIEW TRANSFORMATION INPUT PRIORITY (3P)
SAMPLE STROKE (3P)

**NAME**

SAMPLE VALUATOR – sample current measure of a valuator device

**SYNOPSIS**

**C Syntax**

```
void
psample_val ( ws, dev, value )
Pint        ws;        workstation identifier
Pint        dev;       valuator device number
Pfloat      *value;    OUT value
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

Use SAMPLE VALUATOR to sample the current measure of the specified valuator device without waiting for input from the operator. The device must be in SAMPLE mode. See SET VALUATOR MODE and INITIALIZE VALUATOR for more information.

**C Input Parameters**

*ws* Workstation identifier. An integer specifying the workstation with which the specified valuator device is associated.

*dev* The device number of the valuator device to be sampled. See the *Available Devices* section of INITIALIZE VALUATOR for a description of the available devices.

**C Output Parameter**

*value* A pointer to a Pfloat variable in which to store the VALUATOR measure.

**Execution**

SAMPLE VALUATOR samples the specified valuator device. The device must be in SAMPLE mode. See SET VALUATOR MODE and INITIALIZE VALUATOR for more information.

When a device is set to SAMPLE mode, a measure process is created for the device and the device is activated. When SAMPLE VALUATOR is called, the current value of the measure process for the specified device is retrieved without waiting for further input from the operator.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

252 Ignoring function, the function requires the input device to be in SAMPLE mode

**SEE ALSO**

SET VALUATOR MODE (3P)
INITIALIZE VALUATOR 3 (3P)
REQUEST VALUATOR (3P)
GET VALUATOR (3P)
INQUIRE VALUATOR DEVICE STATE (3P)

**NAME**

SCALE – calculate a 2D transformation matrix to perform a specified scaling

**SYNOPSIS**

**C Syntax**

```
void
pscale ( scale_vector, error_ind, m )
Pvec        *scale_vector;      scale factor vector
Pint        *error_ind;         OUT error indicator
Pmatrix     m;                  OUT transformation matrix
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use SCALE to generate a 2D homogeneous $(3 \times 3)$ transformation matrix that performs a 2D scaling.

The returned matrix can be passed as an argument to SET LOCAL TRANSFORMATION or SET GLOBAL TRANSFORMATION to modify the modelling transformation that is applied to output primitives during traversal.

**C Input Parameter**

*scale_vector*

A pointer to a Pvec data structure containing the scale factors to be applied to the *x* and *y* dimensions. Pvec is defined in phigs.h as follows:

```
typedef struct  {
        Pfloat    delta_x;        /* x coordinate */
        Pfloat    delta_y;        /* y coordinate */
} Pvec;
```

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*m* The $3 \times 3$ homogeneous transformation matrix that performs the specified scaling. Pmatrix is defined in phigs.h as follows:

```
typedef  Pfloat  Pmatrix[3][3];
```

**Execution**

SCALE returns a 2D homogeneous $(3 \times 3)$ transformation matrix that performs the scaling specified by *scale factor vector*. The scaling is relative to the origin of the current modelling coordinate system.

The *scale factor vector* specifies *sx* and *sy* scaling factors that control scaling in the *x* and *y* directions.

**ERRORS**

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

**SEE ALSO**

SET LOCAL TRANSFORMATION (3P)
BUILD TRANSFORMATION MATRIX (3P)
COMPOSE MATRIX (3P)
SCALE 3 (3P)
TRANSFORM POINT (3P)

**NAME**

SCALE 3 – calculate a 3D transformation matrix to perform a specified scaling

**SYNOPSIS**

**C Syntax**

```
void
pscale3 ( scale_vector, error_ind, m)
Pvec3          *scale_vector;      scale factor vector
Pint           *error_ind;         OUT error indicator
Pmatrix3       m;                  OUT transformation matrix
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use SCALE 3 to generate a 3D homogeneous $(4 \times 4)$ transformation matrix that performs a 3D scaling.

The returned matrix can be passed as an argument to SET LOCAL TRANSFORMATION 3 or SET GLOBAL TRANSFORMATION 3 to modify the modelling transformation that is applied to output primitives during traversal.

**C Input Parameter**

*scale_vector*

A pointer to a Pvec3 data structure containing the scale factors to be applied to the *x*, *y*, and *z* dimensions. Pvec3 is defined in phigs.h as follows:

```
typedef struct  {
        Pfloat    delta_x;        /* x coordinate */
        Pfloat    delta_y;        /* y coordinate */
        Pfloat    delta_z;        /* z coordinate */
} Pvec3;
```

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*m* The $4 \times 4$ homogeneous transformation matrix that performs the specified scaling. Pmatrix3 is defined in phigs.h as follows:

```
typedef Pfloat  Pmatrix3[4][4];
```

**Execution**

SCALE 3 returns a 3D homogeneous $(4 \times 4)$ transformation matrix that performs the scaling specified by *scale factor vector*. The scaling is relative to the origin of the current modelling coordinate system.

The *scale factor vector* specifies *sx*, *sy*, and *sz* scaling factors that control scaling in the *x*, *y*, and *z* directions.

**ERRORS**

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

**SEE ALSO**

SET LOCAL TRANSFORMATION 3 (3P)
BUILD TRANSFORMATION MATRIX 3 (3P)
COMPOSE MATRIX 3 (3P)
SCALE (3P)
TRANSFORM POINT 3 (3P)

**NAME**

SET ANNOTATION STYLE – create structure element to set the *current annotation style* attribute

**SYNOPSIS**

**C Syntax**

```
void
pset_anno_style ( style )
Pint            style;          annotation style
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET ANNOTATION STYLE creates a structure element containing a value for the current annotation style attribute, which defines the height of a capital letter in Text Local Coordinates. This attribute applies to the output primitives

- ANNOTATION TEXT RELATIVE
- ANNOTATION TEXT RELATIVE 3

If the current edit mode is INSERT, a SET ANNOTATION STYLE element is inserted into the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, the new SET ANNOTATION STYLE element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

**C Input Parameter**

*style* An integer value specifying one of the following annotation styles:

| | | |
|---|---|---|
| 1 | PANNO_STYLE_UNCONNECTED | *Unconnected* |
| 2 | PANNO_STYLE_LEAD_LINE | *Lead Line* |

**Execution**

When the SET ANNOTATION STYLE element is traversed, the current annotation style entry in the traversal state list is set to *annotation style*. Characters in text output primitives that follow in the structure network are drawn with the current annotation style.

The *Lead Line* value means that the annotation text is connected to its reference point by a lead line drawn using the current polyline attributes.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

PHIGS WORKSTATION DESCRIPTION TABLE (7P)
INQUIRE ANNOTATION FACILITIES (3P)
ANNOTATION TEXT RELATIVE 3 (3P)

**NAME**

SET ANNOTATION TEXT ALIGNMENT – create a structure element to set the *current text alignment* attribute

**SYNOPSIS**

**C Syntax**

```
void
pset_anno_align ( text_align )
Ptext_align          *text_align;          text alignment
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET ANNOTATION TEXT ALIGNMENT creates a structure element containing a value for the current annotation text alignment attribute, which positions the annotation text string in relation to the *annotation text position*. This attribute applies to the output primitives:

- ANNOTATION TEXT RELATIVE
- ANNOTATION TEXT RELATIVE 3

If the current edit mode is INSERT, a SET ANNOTATION TEXT ALIGNMENT element is inserted into the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, then the new SET ANNOTATION TEXT ALIGNMENT element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

**C Input Parameter**

*text_align*

A pointer to the *text alignment* structure, defined as follows:

```
typedef struct {
        Phor_text_align     hor; horizontal component
        Pvert_text_align    vert; vertical component
} Ptext_align;
```

*text_align->hor* is the horizontal component. This is an enumerated value, and may be one of:

| | |
|---|---|
| PHOR_NORM | *NORMAL* |
| PHOR_LEFT | *LEFT* |
| PHOR_CTR | *CENTER* |
| PHOR_RIGHT | *RIGHT* |

*text_align->vert* is the vertical component. This is an enumerated value, and may be one of:

| | |
|---|---|
| PVERT_NORM | *NORMAL* |
| PVERT_TOP | *TOP* |
| PVERT_CAP | *CAP* |
| PVERT_HALF | *HAFL* |
| PVERT_BASE | *BASE* |
| PVERT_BOTTOM | *BOTTOM* |

**Execution**

When the SET ANNOTATION TEXT ALIGNMENT element is traversed, the current annotation text alignment entry in the traversal state list is set to annotation text alignment. This attribute is used to position text strings from annotation text output primitives that follow in the structure network, in relation to the text position provided with each text output primitive. The two components of the alignment can be considered individually.

Imagine first rendering the text string using all other text attributes, and then moving the entire text string to place the *text extent parallelogram* that surrounds the character bodies into the correct position in relation to the text position. (The size and shape of the text is entirely specified by the other attributes.) This movement is oriented by the *annotation text character up vector* and the *annotation text character base vector*. (Consider the direction of the character up vector to be vertical and that of the character base vector to be horizontal.)

The horizontal alignment of LEFT or RIGHT requires the corresponding side of the parallelogram to pass through the text position. The horizontal alignment of CENTER causes the text position to lie midway between the left and right sides of the parallelogram.

The vertical alignment corresponds to one of the five horizontal lines through the definition of a character. (These lines are in the same location for every character in a single font.) The vertical alignment of TOP or BOTTOM requires the corresponding side of the parallelogram to pass through the text position. The vertical alignment of CAP causes the text position to lie on the capline of the string (when the annotation text path is LEFT or RIGHT) or on the cap line of the topmost character in the string (when the annotation text path is UP or DOWN). The vertical alignment of BASE causes the text position to lie on the base line of the entire string (when the annotation text path is LEFT or RIGHT) or on the base line of the bottom character in the string (when annotation text path is UP or DOWN). The vertical alignment of HALF causes the text position to lie on the half line of the entire string (when the annotation text path is LEFT or RIGHT) or on a line midway between the half lines of the top and bottom characters (when annotation text path is UP or DOWN).

The Normal value of either *annotation text alignment* component causes an effect equivalent to one of the other values of the same component. PHIGS defines which other value is used as the natural alignment for the annotation text path value used:

| *Annotation Text Path Value* | *Equivalent (Horizontal, Vertical) Alignment* |
|---|---|
| RIGHT | *Left, Base* |
| LEFT | *Right, Base* |
| UP | *Center, Base* |
| DOWN | *Center, Top* |

The default annotation text alignment is (NORMAL, NORMAL); the default annotation text path is RIGHT.

**Example**

(CENTER, TOP) annotation text alignment might be used with annotation text path RIGHT to center a chart's *x-axis* label under the *x-axis*, without calculating the combined size of the characters in the string. (RIGHT, CENTER) might be used with annotation text path DOWN to center the right edge of a chart's *y-axis* label along the *y-axis*. Each character faces normally, but the characters in the string would proceed down the display, to the left of the *y-axis*.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

ANNOTATION TEXT RELATIVE 3 (3P)
SET TEXT ALIGNMENT (3P)

**NAME**

SET ANNOTATION TEXT CHARACTER HEIGHT – create structure element to set the *current annotation text character height* attribute

**SYNOPSIS**

**C Syntax**

```
void
pset_anno_char_ht ( height )
Pfloat          height;          character height
```

**Required PHIGS Operating States**

(PHOP, ∗, STOP, ∗)

**DESCRIPTION**

**Purpose**

SET ANNOTATION TEXT CHARACTER HEIGHT creates a structure element containing a value for the current annotation text character height attribute, which defines the height of a capital letter, in Text Local Coordinates. This attribute applies to the output primitives:

- ANNOTATION TEXT RELATIVE
- ANNOTATION TEXT RELATIVE 3

If the current edit mode is INSERT, a SET ANNOTATION TEXT CHARACTER HEIGHT element is inserted in the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, then the new SET ANNOTATION TEXT CHARACTER HEIGHT element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

**C Input Parameter**

*height* A real value specifying the annotation text character height.

**Execution**

When the SET ANNOTATION TEXT CHARACTER HEIGHT element is traversed, the current annotation text character height entry in the traversal state list is set to SET ANNOTATION TEXT CHARACTER HEIGHT. Characters in text output primitives that follow in the structure network are drawn with the current annotation text character height.

The annotation text character height is specified in Text Local Coordinates (TLC) and determines the height of a capital letter, measured from the character base line parallel to the *character up vector*. The character base line and character up vector are relative to the TLC system created by the reference point and annotation offset specified in the annotation text relative output primitive.

The default annotation text character height is 0.01.

**ERRORS**

005 Ignoring function, function requires state (PHOP, ∗, STOP, ∗)

**SEE ALSO**

ANNOTATION TEXT RELATIVE 3 (3P)
SET CHARACTER HEIGHT (3P)

**NAME**

SET ANNOTATION TEXT CHARACTER UP VECTOR – create structure element to set the *current annotation text character up vector* attribute

**SYNOPSIS**

**C Syntax**

```
void
pset_anno_char_up_vec ( up_vect )
Pvec            *up_vect;          character up vector
```

**Required PHIGS Operating States**

(PHOP, ∗, STOP, ∗)

**DESCRIPTION**

**Purpose**

SET ANNOTATION TEXT CHARACTER UP VECTOR creates a structure element containing a value for the current annotation text character up vector attribute that determines the orientation of individual characters. This attribute applies to the output primitives:

- ANNOTATION TEXT RELATIVE
- ANNOTATION TEXT RELATIVE 3

If the current edit mode is INSERT, then a SET ANNOTATION TEXT CHARACTER UP VECTOR element is inserted in the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, then the new SET ANNOTATION TEXT CHARACTER UP VECTOR element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

**C Input Parameter**

*up_vect* A pointer to the annotation text character up vector containing TLC values, defined as follows:

```
typedef struct {
    Pfloat      delta_x;    x coordinate
    Pfloat      delta_y;    y coordinate
} Pvec;
```

**Execution**

When the SET ANNOTATION TEXT CHARACTER UP VECTOR element is traversed, the current annotation text character up vector entry in the traversal state list is set to annotation text character up vector. Characters in text output primitives that follow in the structure network are oriented by the annotation text character up vector. The *annotation text character base vector* is defined as a vector of arbitrary length set at a right angle to the character up vector in the clockwise direction.

The default annotation text character up vector is (0,1); the default annotation text character base vector is (1,0). If the annotation text character up vector is determined to be degenerate (in other words, has length 0), then the default is used.

**ERRORS**

005 Ignoring function, function requires state (PHOP, ∗, STOP, ∗)

**SEE ALSO**

ANNOTATION TEXT RELATIVE 3 (3P)
SET CHARACTER UP VECTOR (3P)

**NAME**

SET ANNOTATION TEXT PATH – create a structure element to set the *current text path* attribute

**SYNOPSIS**

**C Syntax**

```
void
pset_anno_path ( path )
Ptext_path      path;          text path
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET ANNOTATION TEXT PATH creates a structure element containing a value for the text path attribute that controls the direction in which the string is written relative to the character up and base vectors. This attribute applies to the following output primitives:

- ANNOTATION TEXT RELATIVE
- ANNOTATION TEXT RELATIVE 3

If the current edit mode is INSERT, then a SET ANNOTATION TEXT PATH element is inserted into the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, then the new SET ANNOTATION TEXT PATH element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

**C Input Parameter**

*path* The annotation text path is one of the following enumerated values:

| | |
|---|---|
| PPATH_RIGHT | *Right* |
| PPATH_LEFT | *Left* |
| PPATH_UP | *UP* |
| PPATH_DOWN | *Down* |

**Execution**

When the SET ANNOTATION TEXT PATH element is traversed, the current annotation text path entry in the traversal state list is set to annotation text path. The annotation text path determines the direction of displacement between one character and the next in a string, defined in relation to the annotation text character base and up vectors.

If annotation text path is set to Right, then the text string is written along a base line in the direction of the ANNOTATION TEXT CHARACTER BASE VECTOR. If it is set to Left, then the text string is written along a base line in the direction opposite to the ANNOTATION TEXT CHARACTER BASE VECTOR. If it is set to Up, then the text string is written in the direction of the ANNOTATION TEXT CHARACTER UP VECTOR. If it is set to Down, then the text string is written in the direction opposite to the ANNOTATION TEXT CHARACTER UP VECTOR.

The ANNOTATION TEXT CHARACTER BASE VECTOR attribute is an implicit attribute derived from the ANNOTATION TEXT CHARACTER UP VECTOR.

**Note:** Annotation text path controls only the direction in which the string is written. The position of the string in relation to the text position point is controlled by the *annotation text alignment*.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

ANNOTATION TEXT RELATIVE 3 (3P)
SET TEXT PATH (3P)

**NAME**

SET BACK INTERIOR COLOUR – create a structure element to set the back interior colour

**SYNOPSIS**

**C Syntax**

```
void
pset_back_int_colr ( colour )
Pgcolr          *colour;          back interior colour
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET BACK INTERIOR COLOUR creates a structure element to set the back interior colour.

When the current back interior colour Aspect Source Flag (ASF) is set to INDIVIDUAL, the current value of this attribute defines the back interior colour to be applied to the following output primitives:

- FILL AREA
- FILL AREA 3
- FILL AREA SET
- FILL AREA SET 3
- FILL AREA SET 3 WITH DATA
- GENERALIZED DRAWING PRIMITIVE *(Fill Area Primitives)*
- GENERALIZED DRAWING PRIMITIVE 3 *(Fill Area Primitives)*
- NON-UNIFORM B-SPLINE SURFACE
- SET OF FILL AREA SET 3 WITH DATA
- QUADRILATERAL MESH 3 WITH DATA
- TRIANGLE STRIP 3 WITH DATA

**C Input Parameters**

*colour* The colour model. Pgcolr is defined as:

```
typedef struct {
        Pint            type;           /* indirect, RGB, CIE, HSV, HLS */
        union {
          Pint          ind;            /* colour table index */
          struct {
                Pfloat          x;      /* red, hue, etc. */
                Pfloat          y;      /* green, saturation, lightness, etc. */
                Pfloat          z;      /* blue, value, saturation, etc. */
          } general;
        } val;
} Pgcolr;
```

**Execution**

Depending on the edit mode, a SET BACK INTERIOR COLOUR element with the value back interior colour in the specified colour model is inserted into the open structure after the element pointer or replaces the element pointed at by the element pointer. The element pointer is then updated to point to this SET BACK INTERIOR COLOUR element.

Back face attributes (such as back area properties, back interior colour, back reflectance characteristics, back interior shading method, back interior style, and back interior style index) are used only when the face distinguishing mode has the value *Yes*, and are applied only to the back facing portion of area defining

primitives. See SET FACE DISTINGUISHING MODE (3P+) for details on front and back face determination.

When the current back interior colour ASF is set to INDIVIDUAL, area-defining output primitives, which follow in the structure network and are subject to back face attributes, are filled using the current back interior colour.

When the current back interior colour ASF is set to BUNDLED, the back interior colour is taken from the workstation extended representation indicated by the current interior index. In this case, the back interior colour set with SET BACK INTERIOR COLOUR has no effect.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

SET INTERIOR COLOUR (3P+)
SET FACE DISTINGUISHING MODE (3P+)
SET INDIVIDUAL ASF (3P)
SET INTERIOR REPRESENTATION PLUS (3P+)

**NAME**

SET BACK INTERIOR REFLECTANCE EQUATION – create a structure element to set the current back interior reflectance equation

**SYNOPSIS**

**C Syntax**

```
void
pset_back_refl_eqn ( equation )
Pint            equation;          back reflectance equation
```

**Required PHIGS Operating States**

(PHOP, ∗, STOP, ∗)

**DESCRIPTION**

**Purpose**

SET BACK INTERIOR REFLECTANCE EQUATION creates a structure element to set the current back interior reflectance equation.

When the current back interior reflectance equation ASF is set to INDIVIDUAL, the current value of this attribute applies to the following output primitives:

- FILL AREA
- FILL AREA 3
- FILL AREA SET
- FILL AREA SET 3
- FILL AREA SET 3 WITH DATA
- SET OF FILL AREA SET 3 WITH DATA
- NON-UNIFORM B-SPLINE SURFACE
- QUADRILATERAL MESH 3 WITH DATA
- TRIANGLE STRIP 3 WITH DATA

**C Input Parameters**

*equation*

The reflectance equation. The possible values defined in phigs.h are:

| | | |
|---|---|---|
| 1 | PREFL_NONE | *No Reflectance Calculation Performed* |
| 2 | PREFL_AMBIENT | *Use Ambient Term* |
| 3 | PREFL_AMB_DIFF | *Use Ambient and Diffuse Terms* |
| 4 | PREFL_AMB_DIFF_SPEC | *Use Ambient, Diffuse, and Specular Terms* |

**Execution**

Depending on the edit mode, a SET BACK INTERIOR REFLECTANCE EQUATION element with the value reflectance equation is inserted into the open structure after the element pointer or replaces the element pointed at by the element pointer. The element pointer is then updated to point to this SET BACK INTERIOR REFLECTANCE EQUATION element.

At traversal time, if the specified reflectance equation is not defined, then the default is used.

Back face attributes (such as back area properties, back interior colour, back interior reflectance equation, back interior shading method, back interior style, and back interior style index) are used only when the face distinguishing mode has the value *Yes* and are applied to only the back facing portion of area defining primitives. See SET FACE DISTINGUISHING MODE (3P+) for details on front and back face determination.

When the current back interior reflectance equation ASF is set to INDIVIDUAL, area-defining output primitives which follow in the structure network are filled using the current back interior reflectance equation.

When the current back interior reflectance equation ASF is set to BUNDLED, the back interior reflectance equation is taken from the workstation extended representation indicated by the current interior index. In this case, the back interior reflectance equation set with SET BACK INTERIOR REFLECTANCE EQUATION has no effect.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

SET FACE DISTINGUISHING MODE (3P+)
SET INDIVIDUAL ASF (3P)
SET INTERIOR REPRESENTATION PLUS (3P+)
SET REFLECTANCE EQUATION (3P+)
FILL AREA (3P)
FILL AREA 3 (3P)
FILL AREA SET (3P)
FILL AREA SET 3 (3P)
FILL AREA SET 3 WITH DATA (3P+)
SET OF FILL AREA SET 3 WITH DATA (3P+)
NON-UNIFORM B-SPLINE SURFACE (3P+)
QUADRILATERAL MESH 3 WITH DATA (3P+)
TRIANGLE STRIP 3 WITH DATA (3P+)

**NAME**

SET BACK INTERIOR SHADING METHOD – create a structure element to set the back interior shading method

**SYNOPSIS**

**C Syntax**

```
void
pset_back_int_shade_meth ( method )
Pint            method;          back shading method
```

**Required PHIGS Operating States**

(PHOP, ∗, STOP, ∗)

**DESCRIPTION**

**Purpose**

SET BACK INTERIOR SHADING METHOD creates a structure element to set the back interior shading method.

When the current back interior shading method ASF is set to INDIVIDUAL, the current value of this attribute applies to the following output primitives:

- FILL AREA
- FILL AREA 3
- FILL AREA SET
- FILL AREA SET 3
- FILL AREA SET 3 WITH DATA
- NON-UNIFORM B-SPLINE SURFACE
- SET OF FILL AREA SET 3 WITH DATA
- QUADRILATERAL MESH 3 WITH DATA
- TRIANGLE STRIP 3 WITH DATA

**C Input Parameters**

*method* The back interior shading method. The predefined values are:

| | | |
|---|---|---|
| 1 | PSD_NONE | *No Shading* |
| 2 | PSD_COLOUR | *Colour Interpolation Shading* |
| 3 | PSD_DOT_PRODUCT | *Dot Product Interpolation Shading* |
| 4 | PSD_NORMAL | *Normal Interpolation Shading* |

**Execution**

Depending on the edit mode, a SET BACK INTERIOR SHADING METHOD element with the value shading method is inserted into the open structure after the element pointer or replaces the element pointed at by the element pointer. The element pointer is then updated to point to this SET BACK INTERIOR SHADING METHOD element.

When the structure is traversed, the current interior shading method entry in the traversal state list is set to the specified value. This value is used with subsequent PHIGS Extension output primitives when the current interior shading method ASF entry in the traversal state list is INDIVIDUAL. During traversal, if the specified shading method is not available on a workstation, method one (*None*) will be used on that workstation.

The interior shading method supported by a workstation type can be determined with the function INQUIRE INTERIOR FACILITIES PLUS.

Back face attributes (such as back area properties, back interior colour, back interior reflectance equation, back interior shading method, back interior style, and back interior style index) are used only when the face distinguishing mode has the value *Yes*, and are applied only to the back facing portion of area defining primitives. See SET FACE DISTINGUISHING MODE (3P+) for details on front and back face determination.

When the current back interior shading method ASF is set to INDIVIDUAL, area-defining output primitives, which follow in the structure network and are subject to back face attributes, are filled using the current back interior shading method.

When the current back interior shading method ASF is set to BUNDLED, the back interior shading method is taken from the workstation extended representation indicated by the current interior index. In this case, the back interior shading meth set with SET BACK INTERIOR SHADING METHOD has no effect.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

SET FACE DISTINGUISHING MODE (3P+)
SET INDIVIDUAL ASF (3P)
SET INTERIOR REPRESENTATION PLUS (3P+)
SET INTERIOR SHADING METHOD (3P+)

**NAME**

SET BACK INTERIOR STYLE – create a structure element to set the back interior style

**SYNOPSIS**

**C Syntax**

```
void
pset_back_int_style ( style )
Pint_style          style;          back interior style
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET BACK INTERIOR STYLE creates a structure element to set the back interior style. SET BACK INTERIOR STYLE determines in what style the interior of the back facing area defining primitives should be filled: *Hollow, Solid, Pattern, Hatch, Empty,* or *General.*

When the current back interior style Aspect Source Flag (ASF) is set to INDIVIDUAL, the current value of this attribute defines the back interior style to be applied to the following output primitives:

- FILL AREA
- FILL AREA 3
- FILL AREA SET
- FILL AREA SET 3
- FILL AREA SET 3 WITH DATA
- GENERALIZED DRAWING PRIMITIVE (*Fill Area Primitives*)
- GENERALIZED DRAWING PRIMITIVE 3 (*Fill Area Primitives*)
- NON-UNIFORM B-SPLINE SURFACE
- SET OF FILL AREA SET 3 WITH DATA
- QUADRILATERAL MESH 3 WITH DATA
- TRIANGLE STRIP 3 WITH DATA

**C Input Parameters**

*style* A pointer to a Pint_style data structure. A Pint_style structure is defined in phigs.h as:

```
typedef enum {
        PSTYLE_HOLLOW,    Hollow
        PSTYLE_SOLID,     Solid
        PSTYLE_PAT,       Patterned
        PSTYLE_HATCH,     Hatched
        PSTYLE_EMPTY,     Empty
} Pint_style;
```

**Execution**

Depending on the edit mode, a SET BACK INTERIOR STYLE element with the value interior style is inserted into the open structure after the element pointer, or it replaces the element pointed at by the element pointer. The element pointer is then updated to point to this SET BACK INTERIOR STYLE element.

The back interior style is used to determine in what style the interior of the back facing area-defining primitives should be filled:

*Hollow*

No filling, but draw the boundary using the colour, or colours, resulting from the rendering pipeline. The linetype and linewidth of this bounding polyline are implementation-dependent. PEX-SI

presently uses a solid line of default width. Clipping of fill areas yields visible edges at clip boundaries.

*Solid*

Fill the interior using the colour, or colours, resulting from the rendering pipeline.

*Pattern*

Fill the interior using the colours resulting from the rendering pipeline. The current BACK INTERIOR STYLE INDEX will be used to select a pattern from the pattern table. In this context, the BACK INTERIOR STYLE INDEX is sometimes referred to as the back pattern index and must have a value greater than zero.

*Hatch*

Fill the interior using the colours resulting from the rendering pipeline. The current BACK INTERIOR STYLE INDEX will be used to select a hatch style from the hatch table. In this context, the BACK INTERIOR STYLE INDEX is sometimes referred to as the back hatch index.

*Empty*

No filling is performed. Area defining primitives drawn using this value will be visible only if the EDGE FLAG is set to ON.

If the requested interior style is not available on a workstation, *Hollow* will be used on that workstation.

Back face attributes (such as back area properties, back interior colour, back reflectance characteristics, back interior shading method, back interior style, and back interior style index) are used only when the face distinguishing mode has the value *Yes*, and are applied only to the back-facing portion of area-defining primitives. See SET FACE DISTINGUISHING MODE (3P+) for details on front and back face determination.

When the current back interior style ASF is set to INDIVIDUAL, area-defining output primitives, which follow in the structure network and are subject to back face attributes, are filled using the current back interior style.

When the current back interior style ASF is set to BUNDLED, the back interior style is taken from the workstation extended representation indicated by the current interior index. In this case, the back interior style set with SET BACK INTERIOR STYLE has no effect.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

SET FACE DISTINGUISHING MODE (3P+)
SET BACK INTERIOR STYLE INDEX (3P+)
SET INDIVIDUAL ASF (3P)
SET INTERIOR REPRESENTATION PLUS (3P+)
SET INTERIOR STYLE (3P)

**NAME**

SET BACK INTERIOR STYLE INDEX – create a structure element to set the back interior style index

**SYNOPSIS**

**C Syntax**

```
void
pset_back_int_style_ind ( index )
Pint            index;          back interior style index
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET BACK INTERIOR STYLE INDEX creates a structure element to set the back interior style index.

When the current back interior style index Aspect Source Flag (ASF) is set to INDIVIDUAL, the current value of this attribute defines the back interior style index to be applied to the following output primitives:

- FILL AREA
- FILL AREA 3
- FILL AREA SET
- FILL AREA SET 3
- FILL AREA SET 3 WITH DATA
- GENERALIZED DRAWING PRIMITIVE (*Fill Area Primitives*)
- GENERALIZED DRAWING PRIMITIVE 3 (*Fill Area Primitives*)
- NON-UNIFORM B-SPLINE SURFACE
- SET OF FILL AREA SET 3 WITH DATA
- QUADRILATERAL MESH 3 WITH DATA
- TRIANGLE STRIP 3 WITH DATA

**C Input Parameters**

*index* The back interior style index.

**Execution**

Depending on the edit mode, a SET BACK INTERIOR STYLE INDEX element with the value interior style index is inserted into the open structure after the element pointer or replaces the element pointed at by the element pointer. The element pointer is then updated to point to this SET BACK INTERIOR STYLE INDEX element.

During traversal, the back interior style index is used only for area-defining facets that are back facing. For interior styles *Hollow*, *Solid*, and *Empty*, the back interior style index value is not used. For interior style *Pattern*, this value is greater than zero and is a pointer into the pattern table of the workstation. For interior style *Hatch*, this value determines which of a number of hatch styles is used. Hatch styles greater than zero are reserved for registration. Hatch styles less than one are workstation-dependent.

At traversal time, if the entry associated with the specified interior style index is not defined, then interior style index one is substituted.

Back face attributes (such as back area properties, back interior colour, back reflectance characteristics, back interior shading method, back interior style, and back interior style index) are used only when the face distinguishing mode has the value *Yes* and are applied only to the back facing portion of area defining primitives. See SET FACE DISTINGUISHING MODE (3P+) for details on front and back face determination.

When the current back interior style index ASF is set to INDIVIDUAL, area-defining output primitives, which follow in the structure network and are subject to back face attributes, are filled using the current back interior style index.

When the current back interior style index ASF is set to BUNDLED, the back interior style index is taken from the workstation extended representation indicated by the current interior index. In this case, the back interior style index set with SET BACK INTERIOR STYLE INDEX has no effect.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

SET FACE DISTINGUISHING MODE (3P+)
SET BACK INTERIOR STYLE (3P+)
SET INDIVIDUAL ASF (3P)
SET INTERIOR REPRESENTATION PLUS (3P+)
SET INTERIOR STYLE INDEX (3P)

**NAME**

SET BACK REFLECTANCE PROPERTIES – create a structure element to set the back area properties

**SYNOPSIS**

**C Syntax**

```
void
pset_back_refl_props ( properties )
Prefl_props          *properties;          back surface area properties
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET BACK REFLECTANCE PROPERTIES creates a structure element to set the back area properties.

When the current back area properties ASF is set to INDIVIDUAL, the current value of this attribute applies to the following output primitives:

- FILL AREA
- FILL AREA 3
- FILL AREA SET
- FILL AREA SET 3
- FILL AREA SET 3 WITH DATA
- GENERALIZED DRAWING PRIMITIVE (*Fill Area Primitives*)
- GENERALIZED DRAWING PRIMITIVE 3 (*Fill Area Primitives*)
- SET OF FILL AREA SET 3 WITH DATA
- NON-UNIFORM B-SPLINE SURFACE
- QUADRILATERAL MESH 3 WITH DATA
- TRIANGLE STRIP 3 WITH DATA

**C Input Parameters**

*properties*

A pointer to a Prefl_props structure that specifies the back area properties. Prefl_props is defined in phigs.h as follows:

```
typedef struct {
        Pfloat    ambient_coef;    /* ambient reflectance coefficient */
        Pfloat    diffuse_coef;    /* diffuse reflectance coefficient */
        Pfloat    specular_coef;   /* specular reflectance coefficient */
        Pgcolr    specular_colr;   /* specular colour */
        Pfloat    specular_exp;    /* specular exponent */
        Pfloat    transpar_coef;   /* transparency coefficient */
} Prefl_props;
```

Pgcolr is defined as:

```
typedef struct {
        Pint            type;           /* indirect, RGB, CIE, HSV, HLS */
        union {
          Pint          ind;            /* colour table index */
          struct {
                Pfloat          x;      /* red, hue, etc. */
```

```
            Pfloat      y;      /* green, saturation, lightness, etc. */
            Pfloat      z;      /* blue, value, saturation, etc. */
        } general;
    } val;
} Pgcolr;
```

The color type is one of the following:

| | |
|---|---|
| 0 | PINDIRECT |
| 1 | PMODEL_RGB |
| 2 | PMODEL_CIELUV |
| 3 | PMODEL_HSV |
| 4 | PMODEL_HLS |

**Execution**

Depending on the edit mode, a SET BACK REFLECTANCE PROPERTIES element is inserted into the open structure after the element pointer or replaces the element pointed at by the element pointer. The element pointer is then updated to point to this SET BACK REFLECTANCE PROPERTIES element.

Back face attributes (such as back area properties, back interior colour, back reflectance characteristics, back interior shading method, back interior style, and back interior style index) are used only when the face distinguishing mode has the value *Yes* and are applied to only the back facing portion of area defining primitives. See SET FACE DISTINGUISHING MODE (3P+) for details on front and back face determination.

When the current back area properties ASF is set to INDIVIDUAL, area-defining output primitives, which follow in the structure network and are subject to back face attributes, are filled using the current back area properties.

When the current back area properties ASF is set to BUNDLED, the back area properties are taken from the workstation extended representation indicated by the current interior index. In this case, the back area properties set with SET BACK REFLECTANCE PROPERTIES has no effect.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

SET REFLECTANCE PROPERTIES (3P+)
SET FACE DISTINGUISHING MODE (3P+)
SET INDIVIDUAL ASF (3P)
SET INTERIOR REPRESENTATION PLUS (3P+)
SET FACE CULLING MODE (3P+)
FILL AREA (3P)
FILL AREA 3 (3P)
FILL AREA SET (3P)
FILL AREA SET 3 (3P)
FILL AREA SET 3 WITH DATA (3P+)
SET OF FILL AREA SET 3 WITH DATA (3P+)
QUADRILATERAL MESH 3 WITH DATA (3P+)
TRIANGLE STRIP 3 WITH DATA (3P+)

**NAME**

SET CHARACTER EXPANSION FACTOR – create structure element to set the current *character expansion factor* attribute

**SYNOPSIS**

**C Syntax**

```
void
pset_char_expan ( exp_factor )
Pfloat          exp_factor;          character expansion factor
```

**Required PHIGS Operating States**

(PHOP, ∗, STOP, ∗)

**DESCRIPTION**

**Purpose**

SET CHARACTER EXPANSION FACTOR creates a structure element containing a value for the current character expansion attribute, which changes the characters' width-to-height ratio from the ratio with which the font was designed.

When the character expansion factor Aspect Source Flag (ASF) is set to INDIVIDUAL, this attribute applies to the following output primitives:

- TEXT
- TEXT 3
- ANNOTATION TEXT RELATIVE
- ANNOTATION TEXT RELATIVE 3

**C Input Parameter**

*exp_factor*

A real value specifying the character expansion factor.

**Execution**

If the current edit mode is INSERT, a SET CHARACTER EXPANSION FACTOR element is inserted into the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, the new SET CHARACTER EXPANSION FACTOR element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element. When the SET CHARACTER EXPANSION FACTOR element is traversed, the current character expansion factor entry in the traversal state list is set to exp_factor.

When the current character expansion factor ASF is set to INDIVIDUAL, characters in text output primitives that follow in the structure network are modified by the absolute value of the expansion factor specified.

The default character expansion factor is 1.0, which results in no change in the width-to-height ratio of characters from their definition by the font designer. A character expansion factor of less than 1.0 produces narrower characters, and a character expansion factor greater than 1.0 produces wider characters.

When the character expansion factor ASF is set to BUNDLED, the character expansion factor is taken from the workstation's representation indicated by the current text index. In this case, the expansion factor set with SET CHARACTER EXPANSION FACTOR has no effect.

**ERRORS**

005 Ignoring function, function requires state (PHOP, ∗, STOP, ∗)

**SEE ALSO**

SET INDIVIDUAL ASF (3P)
SET TEXT REPRESENTATION (3P)
SET CHARACTER HEIGHT (3P)
SET CHARACTER SPACING (3P)

**NAME**

SET CHARACTER HEIGHT – create structure element to set the *current character height* attribute

**SYNOPSIS**

**C Syntax**

```
void
pset_char_ht ( height )
Pfloat          height;          character height
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET CHARACTER HEIGHT creates a structure element containing a value for the current character height attribute, which defines the height of a capital letter, in Text Local Coordinates. This attribute applies to the output primitives:

- TEXT
- TEXT 3

If the current edit mode is INSERT, then a SET CHARACTER HEIGHT element is inserted into the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, then the new SET CHARACTER HEIGHT element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

**C Input Parameter**

*height* A real value specifying the character height.

**Execution**

When the SET CHARACTER HEIGHT element is traversed, the current character height entry in the traversal state list is set to character height. Characters in text output primitives that follow in the structure network are drawn with the current character height.

The character height is specified in Text Local Coordinates (TLC) and determines the height of a capital letter, measured from the character base line parallel to the character up vector. The character base line and character up vector are relative to the Text Local Coordinate system created by the text position and text vectors specified in the text output primitive.

The default character height is 0.01.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

SET CHARACTER EXPANSION FACTOR (3P)
SET CHARACTER SPACING (3P)
SET TEXT REPRESENTATION (3P)
SET ANNOTATION TEXT CHARACTER HEIGHT (3P)

**NAME**

SET CHARACTER SPACING – create structure element to set the *current character spacing* attribute

**SYNOPSIS**

**C Syntax**

```
void
pset_char_space ( spacing )
Pfloat          spacing;          character spacing
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET CHARACTER SPACING creates a structure element containing a value for the current character spacing attribute, which controls how much space is inserted between characters in a text string.

When the character spacing Aspect Source Flag (ASF) is set to INDIVIDUAL, this attribute applies to the following output primitives:

- TEXT
- TEXT 3
- ANNOTATION TEXT RELATIVE
- ANNOTATION TEXT RELATIVE 3

**C Input Parameter**

*spacing* The character spacing, specified as a real fraction of the font's nominal character height.

**Execution**

If the current edit mode is INSERT, then a SET CHARACTER SPACING element is inserted in the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, then the new SET CHARACTER SPACING element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

When the SET CHARACTER SPACING element is traversed, the current character spacing entry in the traversal state list is set to character spacing. When the current character spacing ASF is set to INDIVIDUAL, the character spacing is applied to text strings in output primitives that follow in the structure network.

The character spacing attribute specifies the amount of space to be inserted between the character bodies in the text primitives. A positive value inserts additional space between characters, and a negative value causes the character bodies to overlap. The default value is 0.0: the character bodies are placed adjacent to one another, with no additional space beyond that in the font design. (A character's body is the rectangle that encloses the limits of the character.)

When the character spacing ASF is set to BUNDLED, the character spacing will be taken from the workstation's representation indicated by the current text index. In this case, the spacing set with SET CHARACTER SPACING has no effect.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

SET CHARACTER EXPANSION FACTOR (3P)
SET CHARACTER HEIGHT (3P)
SET INDIVIDUAL ASF (3P)
SET TEXT REPRESENTATION (3P)

**NAME**

SET CHARACTER UP VECTOR – create a structure element to set the *current character up vector* attribute

**SYNOPSIS**

**C Syntax**

```
void
pset_char_up_vec ( up_vect )
Pvec            *up_vect;          character up vector
```

**Required PHIGS Operating States**

(PHOP, ∗, STOP, ∗)

**DESCRIPTION**

**Purpose**

SET CHARACTER UP VECTOR creates a structure element containing a value for the current character up vector attribute, which determines the orientation of individual characters. This attribute applies to the following output primitives:

- TEXT
- TEXT 3

If the current edit mode is INSERT, then a SET CHARACTER UP VECTOR element is inserted into the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, then the new SET CHARACTER UP VECTOR element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

**C Input Parameter**

*up_vect* A pointer to the character up vector containing TLC values, defined as follows:

```
typedef struct {
        Pfloat  delta_x;        x coordinate
        Pfloat  delta_y;        y coordinate
} Pvec;
```

**Execution**

When the SET CHARACTER UP VECTOR element is traversed, the current character up vector entry in the traversal state list is set to character up vector. Characters in text output primitives that follow in the structure network will be oriented by the character up vector. The character base vector is defined as a vector of arbitrary length set at a right angle to the character up vector in the clockwise direction.

The default character up vector is (0,1); the default character base vector is (1,0). If the character up vector is determined to be degenerate (in other words, has length 0), then the defaults are used.

**ERRORS**

005 Ignoring function, function requires state (PHOP, ∗, STOP, ∗)

**SEE ALSO**

SET ANNOTATION TEXT CHARACTER UP VECTOR (3P)

**NAME**

SET CHOICE MODE – set choice device operating mode and echoing state

**SYNOPSIS**

**C Syntax**

```
void
pset_choice_mode ( ws, dev, mode, echo )
Pint            ws;         workstation identifier
Pint            dev;        choice device number
Pop_mode        mode;       operating mode
Pecho_switch    echo;       echo switch
```

**Required PHIGS Operating States**

(PHOP, WSOP, ∗, ∗)

**DESCRIPTION**

**Purpose**

SET CHOICE MODE sets the operating mode and echo switch for a specified choice device on a specified workstation.

**C Input Parameters**

*ws* The workstation identifier of the workstation associated with the device.

*dev* The device number of the choice device to be set. See the *Available Devices* section in INITIALIZE CHOICE 3 for a description of the available devices.

*mode* Specifies the operating mode for the specified choice device. Pop_mode is an enumeration defined in phigs.h as follows:

```
typedef enum {
        POP_REQ,
        POP_SAMPLE,
        POP_EVENT
} Pop_mode;
```

*echo* The echo switch value for the specified choice device. Pecho_switch is an enumeration defined in phigs.h as follows:

```
typedef enum {
        PSWITCH_NO_ECHO,
        PSWITCH_ECHO
} Pecho_switch;
```

**Execution**

SET CHOICE MODE sets the operating mode and echo flag of the specified device. The default operating mode is POP_REQ. The default echo switch is PSWITCH_ECHO.

The operating mode controls how the input from the device is obtained:

- If the operating mode is REQUEST, REQUEST CHOICE may be used to prompt for and receive input from a choice input device. REQUEST CHOICE will not return until the operator enters input to the device.

- If the operating mode is SAMPLE, SAMPLE CHOICE may be used to return the current input value of the device without waiting for operator input.

- If the operating mode is EVENT, the input values generated by the operator are added to the PHIGS input event queue. The functions AWAIT EVENT and GET CHOICE may then be used to retrieve these events from the input queue.

The echo switch may be usd by some devices to control whether or not values selected by the operator are echoed to the device.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

**SEE ALSO**

INITIALIZE CHOICE 3 (3P)
REQUEST CHOICE (3P)
GET CHOICE (3P)
SAMPLE CHOICE (3P)
INQUIRE CHOICE DEVICE STATE (3P)

**NAME**

SET COLOUR MAPPING INDEX – create a structure element to set the colour mapping index

**SYNOPSIS**

**C Syntax**

```
void
pset_colr_map_ind ( index )
Pint            index;          colour mapping index
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET COLOUR MAPPING INDEX creates an element in the currently open structure containing a colour mapping index. The index number corresponds to an entry in the table of defined colour mapping representations in the workstation state list.

**C Input Parameter**

*index* This is an index to the workstation colour mapping bundle table.

**Execution**

Depending on the edit mode, a SET COLOUR MAPPING INDEX element with a value colour mapping index is inserted into the open structure after the element pointer or replaces the element pointed to by the element pointer. The element pointer is then updated to point to this SET COLOUR MAPPING INDEX element.

When a SET COLOUR MAPPING INDEX structure element is traversed, the current colour mapping index value in the PHIGS Extension traversal state list is set to index. This value indexes a colour mapping representation in the table of defined colour mapping operations stored in the workstation state list. If the entry corresponding to the specified index is not defined, then an entry of zero is substituted in the index.

The current colour mapping index determines the colour mapping operation that will be applied to all primitives subsequently displayed.

**ERRORS**

005 Ignoring function, function requires state (PHOP,*,STOP,*)

121 Ignoring function, the colour mapping index is less than 0.

**SEE ALSO**

SET COLOUR MAPPING REPRESENTATION (3P+)
INQUIRE LIST OF COLOUR MAPPING INDICES (3P+)

**NAME**

SET COLOUR MAPPING REPRESENTATION – define a colour mapping representation entry in the workstation table of defined colour mapping representations

**SYNOPSIS**

**C Syntax**

**void**
**pset_colr_map_rep ( ws, index, map_method, map_data )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **index;** | *colour mapping bundle index* |
| **Pint** | **map_method;** | *mapping method* |
| **Pcolr_map_data** | ***map_data;** | *mapping method data record* |

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

SET COLOUR MAPPING REPRESENTATION defines or sets a colour mapping representation in the specified workstation's colour mapping table.

*ws* The workstation identifier specifies the workstation for which the colour mapping representation is to be defined.

*index* The index of the colour mapping representation to set.

*map_method*

The colour mapping method. The colour mapping methods are defined in phigs.h as follows:

1 PCOLR_MAP_TRUE
2 PCOLR_MAP_PSEUDO
3 PCOLR_MAP_PSEUDO_N

**C Output Parameters**

*map_data*

A pointer to a pointer to a Pcolr_map_data structure that contains the information specific to the data mapping method. Pcolr_map_data is defined in phigs.h as follows:

```
typedef union {
        struct {
                Pint            colr_model;
                Pfloat_list     weights;
                Pcolr_rep_list  colrs;
        } method_r2
        struct {
                Pint            colr_model;
                Pfloat_list_list colr_lists;
        } method_r3
} Pcolr_map_data;
```

Pfloat_list is defined in phigs.h as follows:

```
typedef struct {
        Pint            num_floats;     /* number of Pfloats in list */
        Pfloat          *floats;        /* list of floats *
} Pfloat_list;
```

Pfloat_list_list is defined in phigs.h as follows:

```
typedef struct {
```

```
        Pint            num_lists;      /* number of lists in list */
        Pfloat_list     *lists;         /* list of float lists */
} Pfloat_list_list;
```

Pcolr_rep_list is defined in phigs.h as follows:

```
typedef struct {
        Pint            num_colr_reps;  /* number of colours */
        Pcolr_rep       *colr_reps;     /* array of colours */
} Pcolr_rep_list;
```

Pcolr_rep is defined in phigs.h as follows:

```
typedef union {
        Prgb            rgb;            /* Red Green Blue colour specification */
        Pcieluv         cieluv;         /* CIE L*U*V* colour specification */
        Phls            hls;            /* Hue Lightness Saturation colour specification */
        Phsv            hsv;            /* Hue Saturation Value colour specification */
        Pdata           unsupp;         /* Colour in unsupported colour model */
} Pcolr_rep;
```

Prgb is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          red;            /* red, hue, etc */
        Pfloat          green;          /* green, saturation, lightness, etc */
        Pfloat          blue;           /* blue, value, saturation, etc */
} Prgb;
```

Pcieluv is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          cieluv_x;       /* x coefficient */
        Pfloat          cieluv_y;       /* y coefficient */
        Pfloat          cieluv_y_lum;   /* y luminance */
} Pcieluv;
```

Phsv is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          hue;            /* hue */
        Pfloat          satur;          /* saturation */
        Pfloat          value;          /* value */
} Phsv;
```

Phls is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          hue;            /* hue */
        Pfloat          lightness;      /* lightness */
        Pfloat          satur;          /* saturation */
} Phls;
```

Pdata is defined in phigs.h as follows:

```
typedef struct {
        size_t          size;           /* size of data */
        char            *data           /*pointer to data */
} Pdata;
```

There is no data associated with mapping method PCOLR_MAP_TRUE. The data for method PCOLR_MAP_PSEUDO is a list of weights and a list of direct colours. *colr_model* specifies the type of colours given in the list of colours, *colrs*. The colour models are defined in phigs.h as follows:

1 PMODEL_RGB
2 PMODEL_CIELUV
3 PMODEL_HSV
4 PMODEL_HLS

The data for mapping method PCOLR_MAP_PSEUDO_N is *N* lists of direct colours. These colours may be of the same types as for PCOLR_MAP_PSEUDO.

**Execution**

When SET COLOUR MAPPING REPRESENTATION is called, the specified entry in the table of defined colour mapping representations of the specified workstation is set to the values specified.

During structure traversal, the current colour mapping representation is selected from the workstation table by a structure element created by the SET COLOUR MAPPING INDEX subroutine.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT nor OUTIN)

103 Ignoring function, setting this bundle table entry would exceed the maximum number of entries allowed in the workstation bundle table

110 Ignoring function, the specified colour model is not available on the workstation

121 Ignoring function, colour mapping index is less than 0.

125 Ignoring function, the total of the colour range fields in all the table entries is too large.

126 Ignoring function, the specified colour mapping method is not available on the specified workstation.

136 Ignoring function, one of the components of the colour specification is out of range

138 Ignoring function, one or more of the fields in the specified data record is inconsistent with te specified type.

**SEE ALSO**

SET COLOUR MAPPING INDEX (3P+)
INQUIRE COLOUR MAPPING REPRESENTATION (3P+)

**NAME**

SET COLOUR MODEL – select colour model for workstation colour table

**SYNOPSIS**

**C Syntax**

```
void
pset_colr_model ( ws, model )
Pint        ws;          workstation identifier
Pint        model;       colour model
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

SET COLOUR MODEL selects the colour model used to represent colours in the workstation's table of defined colour representations, which is used to specify a primitive's colour attribute.

**C Input Parameters**

*ws* Set the colour model for the workstation with identifier *ws*.

*model* Use this colour model to interpret the colour values in the workstation colour table. Values defined in phigs.h are:

| | | |
|---|---|---|
| 0 | PINDIRECT | |
| 1 | PMODEL_RGB | *Red, Green, Blue* |
| 2 | PMODEL_CIELUV | *CIE universal colour definition system* |
| 3 | PMODEL_HSV | *Hue, saturation, value* |
| 4 | PMODEL_HLS | *Hue, lightness, saturation* |

**Execution**

When SET COLOUR MODEL is called, the current colour model entry in the workstation state list is set to colour model, which selects a method of interpreting the colour bundles in the workstation table of defined colour representations. Each entry on the colour table contains three colour values. When the current colour model is set to *Red, Green, Blue*, these values are interpreted as the red, green and blue components, respectively, of the colour in RGB colour space.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (in other words, the workstation category is neither OUTPUT, OUTIN, nor MO)

110 Ignoring function, the specified colour model is not available on the workstation.

**SEE ALSO**

INQUIRE COLOUR MODEL FACILITIES (3P)
INQUIRE COLOUR MODEL (3P)

**NAME**

SET COLOUR REPRESENTATION – define a colour representation entry in the workstation's colour table

**SYNOPSIS**

**C Syntax**

```
void
pset_colr_rep ( ws, index, rep )
Pint        ws;        workstation identifier
Pint        index;     colour bundle index
Pcolr_rep   *rep;      colour representation pointer
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

SET COLOUR REPRESENTATION defines an entry in the workstation's colour table. Each entry in this table contains three component values, defining a colour in the current colour model. The colour representations apply to all primitives.

**Note:** PEX-SI presently supports only the RGB colour model.

**C Input Parameters**

*ws* The identifier of the workstation for which the colour representation is being defined.

*index* The *colour index* of the entry being defined.

*rep* A pointer to a union containing the three *colour components* defining the colour representation. Pcolr_rep is defined in phigs.h as follows:

```
typedef union {
        Prgb        rgb;        /* Red Green Blue colour specification */
        Pcieluv     cieluv;     /* CIE L*U*V* colour specification */
        Phls        hls;        /* Hue Lightness Saturation colour specification */
        Phsv        hsv;        /* Hue Saturation Value colour specification */
        Pdata       unsupp;     /* Colour in unsupported colour model */
} Pcolr_rep;
```

Prgb is defined in phigs.h as follows:

```
typedef struct {
        Pfloat      red;        /* red, hue, etc */
        Pfloat      green;      /* green, saturation, lightness, etc */
        Pfloat      blue;       /* blue, value, saturation, etc */
} Prgb;
```

Pcieluv is defined in phigs.h as follows:

```
typedef struct {
        Pfloat      cieluv_x;       /* x coefficient */
        Pfloat      cieluv_y;       /* y coefficient */
        Pfloat      cieluv_y_lum;   /* y luminance */
} Pcieluv;
```

Phls is defined in phigs.h as follows:

```
typedef struct {
        Pfloat      hue;        /* hue */
        Pfloat      lightness;  /* lightness */
        Pfloat      satur;      /* saturation */
```

```
} Phls;
```

Phsv is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          hue;            /* hue */
        Pfloat          satur;          /* saturation */
        Pfloat          value;          /* value */
} Phsv;
```

Pdata is defined in phigs.h as follows:

```
typedef struct {
        size_t          size;           /* size of data */
        char            *data           /*pointer to data */
} Pdata;
```

**Execution**

When SET COLOUR REPRESENTATION is called, the colour index entry in the table of defined colour representations on the workstation is set to the three colour components specified.

For the RGB colour model, the only one implemented in this release, the range for these colour components is 0 to 1: 0 sets the colour component off; 1 sets the colour component at full intensity.

The colour table is numbered from 0, the background colour, to a workstation-dependent maximum.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (in other words, the workstation category is neither OUTPUT, OUTIN, nor MO)

113 Ignoring function, the colour index value is less than zero

103 Ignoring function, setting this bundle table entry would exceed the maximum number of entries allowed in the workstation bundle table

118 Ignoring function, one of the components of the colour specification is out of range. The valid range is dependent upon the current colour model

**SEE ALSO**

PHIGS WORKSTATION DESCRIPTION TABLE (7P)
WORKSTATION TYPE SET (3P)
SET POLYLINE COLOUR INDEX (3P)
SET POLYMARKER COLOUR INDEX (3P)
SET TEXT COLOUR INDEX (3P)
SET INTERIOR COLOUR INDEX (3P)
SET EDGE COLOUR INDEX (3P)
SET COLOUR MODEL (3P)
INQUIRE COLOUR REPRESENTATION (3P)
INQUIRE PREDEFINED COLOUR REPRESENTATION (3P)

**NAME**

SET CONFLICT RESOLUTION – set the conflict resolution flags

**SYNOPSIS**

**C Syntax**

```
void
pset_conf_res ( archival_resolution, retrieval_resolution )
Pconf_res       archival_resolution;      archival conflict resolution
Pconf_res       retrieval_resolution;     retrieval conflict resolution
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

The SET CONFLICT RESOLUTION subroutine sets the conflict resolution flags controlling how structure archival and retrieval conflicts are resolved. The flag value can be MAINTAIN, ABANDON, or UPDATE.

The default value of the archival conflict resolution flag is UPDATE. The default value of the retrieval conflict resolution flag is ABANDON.

**C Input Parameters**

*archival_resolution*

An enumerated variable specifying the value of the archival conflict resolution flag.

*retrieval_resolution*

An enumerated variable specifying the value of the retrieval conflict resolution flag.

Valid values (defined in phigs.h) for these two flags are:

PRES_MAINTAIN
PRES_ABANDON
PRES_UPD

**Execution**

The SET CONFLICT RESOLUTION subroutine sets the conflict resolution flags in the PHIGS workstation state list. The archival conflict resolution flag controls what happens when an attempt is made to archive structures whose identifiers already exist in the archive file. The retrieval conflict resolution flag controls what happens when an attempt is made to retrieve structures whose identifiers already exist in the Central Structure Store.

The flag values can be MAINTAIN, ABANDON, or UPDATE. A value of MAINTAIN prevents conflicting structures from being overwritten; a value of UPDATE allows conflicting structures to be overwritten; and a value of ABANDON indicates that the entire archival or retrieval operation should be abandoned, with no changes made to the archive or the Central Structure Store if there are any conflicts.

**ERRORS**

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

**SEE ALSO**

ARCHIVE STRUCTURES (3P)
INQUIRE ALL CONFLICTING STRUCTURES (3P)
INQUIRE CONFLICTING STRUCTURES IN NETWORK (3P)
INQUIRE CONFLICT RESOLUTION (3P)
RETRIEVE STRUCTURES (3P)

**NAME**

SET CURVE APPROXIMATION CRITERIA – create a structure element to set the curve approximation criteria

**SYNOPSIS**

**C Syntax**

```
void
pset_curve_approx ( type, value )
Pint        type;       approximation type
Pfloat      value;      approximation value
```

**Required PHIGS Operating States**

(PHOP, ∗, STOP, ∗)

**DESCRIPTION**

**Purpose**

SET CURVE APPROXIMATION CRITERIA places into the currently open structure a structure element that specifies the curve approximation type and value. The curve approximation type controls the display precision of non-uniform B-spline curves by specifying the method to use to render the curve.

When the current curve approximation criteria Aspect Source Flag (ASF) is set to INDIVIDUAL, the current value of this attribute defines the curve approximation criteria to be applied to the following output primitive:

- NON-UNIFORM B-SPLINE CURVE
- GENERALIZED DRAWING PRIMITIVE
- GENERALIZED DRAWING PRIMITIVE 3

**C Input Parameters**

*type* The curve approximation type. Possible values defined in phigs.h are:

| | |
|---|---|
| 1 | PCURV_WS_DEP |
| 2 | PCURV_CONSTANT_PARAMETRIC_BETWEEN_KNOTS |
| 3 | PCURV_CHORDAL_SIZE_WC |
| 4 | PCURV_CHORDAL_SIZE_NPC |
| 5 | PCURV_CHORDAL_SIZE_DC |
| 6 | PCURV_CHORDAL_DEVIATION_WC |
| 7 | PCURV_CHORDAL_DEVIATION_NPC |
| 8 | PCURV_CHORDAL_DEVIATION_DC |

*value* The value to use when applying the specified approximation method. Valid values depend on the approximation method specified. See the *Execution* section below for a description of the use of *value* by each approximation method.

**Execution**

Depending upon the edit mode, a SET CURVE APPROXIMATION CRITERIA element is inserted into the open structure after the element pointer, or replaces the element pointed at by the element pointer. The element pointer is then updated to point to this SET CURVE APPROXIMATION CRITERIA element.

The approximation methods corresponding to the defined approximation types and the use of the approximation value by each method are:

*Implementation Dependent, Type 1*

Use an implementation-dependent approximation method. This method is defined by the server.

*Constant Parametric Subdivision Between Knots, Type 2*

Tessellate using equal parametric increments between each pair of knots on the curve. The integer portion of the approximation value is used and interpreted as follows:

≤ 0: Evaluate the curve only at the parameter limits and at the knots that fall within the parameter limits.

> 0: Evaluate the curve at the parameter limits, at the knots that fall within the parameter limits, and at the specified number of positions between each pair of knots, but only if the position is within the parameter limits.

*Chordal Size in WC, Type 3*
Tessellate until the length in World Coordinates (WC) of every line segment (chord) representing the curve is less than the specified positive real-number approximation value.

*Chordal Size in NPC, Type 4*
Tessellate until the length in Normalized Projection Coordinates (NPC) of every line segment (chord) representing the curve is less than the specified positive real-number approximation value.

*Chordal Size in DC, Type 5*
Tessellate until the length in Device Coordinates (DC) of every line segment (chord) representing the curve is less than the specified positive real-number approximation value.

*Chordal Deviation in WC, Type 6*
Tessellate until the absolute value of the maximum deviation measured in WC between the curve and approximating line segment (chord) is less than the specified positive real-number approximation value.

*Chordal Deviation in NPC, Type 7*
Tessellate until the absolute value of the maximum deviation measured in NPC between the curve and approximation line segment (chord) is less than the specified positive real-number approximating value.

*Chordal Deviation in DC, Type 8*
Tessellate until the absolute value of the maximum deviation measured in DC between the curve and approximation line segment (chord) is less than the specified positive real-number approximating value.

At traversal time, the default approximation type is used if the specified approximation type is not supported by the workstation. If the approximation value is inconsistent with the specified approximation type, a value of 1.0 will be used. The approximation types supported by a workstation type can be determined with the function INQUIRE CURVE AND SURFACE FACILITIES, or by consulting the relevant entries in PHIGS WORKSTATION DESCRIPTION TABLE (7P).

When the current curve approximation criteria ASF is set to INDIVIDUAL, output primitives that follow in the structure network are filled using the current curve approximation criteria style.

When the current curve approximation criteria ASF is set to BUNDLED, the curve approximation criteria is taken from the workstation extended representation indicated by the current polyline index. In this case, the curve approximation criteria set with SET CURVE APPROXIMATION CRITERIA has no effect.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

NON-UNIFORM B-SPLINE CURVE (3P+)
INQUIRE CURVE AND SURFACE FACILITIES (3P+)

**NAME**

SET DEPTH CUE INDEX – create a structure element containing the value depth cue index

**SYNOPSIS**

**C Syntax**

```
void
pset_dcue_ind ( index )
Pint            index;          depth cue index
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET DEPTH CUE INDEX creates an element in the currently open structure containing a depth cue index number. The index number corresponds to an entry in the table of defined depth cue operations in the workstation state list.

**C Input Parameter**

*index* This is an index to the workstation depth cue bundle table.

The index number corresponds to an entry in the table of defined depth cue operations in the workstation state list. Up to 20 depth cue table entries may be defined. Entry zero is predefined to a depth cue mode of SUPPRESSED and cannot be changed. Entries other than zero can be changed with the SET DEPTH CUE REPRESENTATION subroutine.

**Execution**

Depending on the edit mode, a SET DEPTH CUE INDEX element with a value depth cue index is inserted into the open structure after the element pointer or replaces the element pointed to by the element pointer. The element pointer is then updated to point to this SET DEPTH CUE INDEX element.

When a SET DEPTH CUE INDEX structure element is traversed, the current depth cue index value in the PHIGS Extension traversal state list is set to index. This value indexes a depth cue representation in the table of defined depth cue operations stored in the workstation state list. If the entry corresponding to the specified index is not defined, then an entry of zero is substituted in the index.

The current depth cue index determines the depth cue operation that will be applied to all primitives in the display. Each entry in the workstation table of defined depth cue operations contains the following information:

- depth cue index number
- depth cue mode (SUPPRESSED or ALLOWED)
- depth cue reference planes
- depth cue scale factors
- depth cue color

If the depth cue mode is SUPPRESSED, depth cueing is not applied to subsequent primitives in the display. If the depth cue mode is ALLOWED, depth cueing is applied to all subsequent primitives using the operation defined by the reference planes, scale factors, and depth cue color.

When a workstation is opened, entries zero and one are initialized with the following representation:

| | *Entry 0* | *Entry 1* |
|---|---|---|
| depth cue mode | SUPPRESSED | ALLOWED |
| depth cue reference planes | 0.0, 1.0 | 0.0, 1.0 |
| depth cue scaling | 1.0, 1.0 | 0.0, 1.0 |
| depth cue colour | (INDIRECT, 0) | (INDIRECT, 0) |

For depth cue reference planes and depth cue scaling, the first value is the back reference plane and scaling.

**ERRORS**

005 Ignoring function, function requires state (PHOP,*,STOP,*)

063 Ignoring function, the depth cue index value is less than zero

**SEE ALSO**

SET DEPTH CUE REPRESENTATION (3P+)
INQUIRE LIST OF DEPTH CUE INDICES (3P+)

**NAME**

SET DEPTH CUE REPRESENTATION – define a depth cue representation entry in the workstation table of defined depth cue representations

**SYNOPSIS**

**C Syntax**

```
void
pset_dcue_rep ( ws, index, rep )
Pint            ws;          workstation identifier
Pint            index;       depth cue bundle index
Pdcue_bundle    *rep;        depth cue representation pointer
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

SET DEPTH CUE REPRESENTATION assigns parameter values defining a depth cue operation to a specified index number in the workstation table of defined depth cue representations. Each entry in this table contains a value for depth cue mode, reference planes, scaling, and colour. When enabled, depth cueing is applied to all output primitives subsequently displayed.

Depth cueing mixes a scaled portion of the depth cue colour with the primitive colour as a function of the *z* coordinate in NPC space. Using the background colour as the depth cue colour, this operation allows you to fade the colour of the parts of the primitives that are further from the viewer in the display.

**C Input Parameters**

*ws* The workstation identifier specifies the workstation for which the depth cue representation is to be defined. The workstation must have been opened with the OPEN WORKSTATION subroutine before calling SET DEPTH CUE REPRESENTATION.

*index* The index to the workstation depth cue bundle table.

The index number corresponds to an entry in the table of defined depth cue operations in the workstation state list. Up to 20 depth cue table entries can be defined. Entry zero is predefined to a depth cue mode of PSUPPRESSED and cannot be changed. Entries other than zero can be changed with the SET DEPTH CUE REPRESENTATION subroutine.

*rep* A pointer to a Pdcue_bundle data structure containing attribute values defining a depth cue representation. A Pdcue_bundle structure is defined in phigs.h as follows:

```
typedef struct {
        Pdcue_mode      mode;           /* depth cue mode */
        Pfloat          ref_planes[2];  /* depth cue reference planes */
        Pfloat          scaling[2];     /* depth cue scaling */
        Pgcolr          colr;           /* depth cue colour */
} Pdcue_bundle;
```

Pdcue_mode is defined as:

```
typedef enum {
        PSUPPRESSED,
        PALLOWED
} Pdcue_mode;
```

Refplanes and scaling entries of 0 are back reference plane and corresponding scaling factors. Refplanes and scaling entries of 1 are front reference plane and corresponding scaling factors. These scaling factors define the portion of the primitive colour that should be combined with the depth cue colour, as a function of *z* in NPC.

Pgcolr is defined as:

```
typedef struct {
        Pint                type;       /* indirect, RGB, CIE, HSV, HLS */
        union {
            Pint            ind;        /* index in workstation colour bundle table */
            struct {
                Pfloat      x;          /* red, hue, etc. */
                Pfloat      y;          /* green, saturation, lightness, etc. */
                Pfloat      z;          /* blue, value, saturation, etc. */
            } general;
        } val;
} Pgcolr;
```

Constants defined for colour type are:

| | | |
|---|---|---|
| 0 | PINDIRECT | *Indirect* |
| 1 | PMODEL_RGB | *Red, Green, Blue* |
| 2 | PMODEL_CIELUV | *CIE* |
| 3 | PMODEL_HSV | *Hue, Saturation, Value* |
| 4 | PMODEL_HLS | *Hue, Lightness, Saturation* |

**Execution**

When a SET DEPTH CUE REPRESENTATION is called, the entry index in the table of defined depth cue representations on the workstation identifier is set to the values in *rep*.

During structure traversal, the current depth cue representation is selected from the workstation table by a structure element created by the SET DEPTH CUE INDEX subroutine. While the depth cue mode of the current representation is ALLOWED, the depth cue operation defined by the other representation parameters is applied to all the following primitives in the structure network.

The depth cueing operation changes the primitive colours in the display as a function of their *z* coordinate in NPC space. The first number in scaling specifies the portion of the primitive colour that is combined with the depth cue colour at, or in front of, the front reference plane. The second scaling value specifies the portion of primitive colour applied at, or behind, the back reference plane. The portion of depth cue colour applied to primitives between the two planes is scaled by a value linearly interpolated between the two scaling values.

When a workstation is opened, entries zero and one are initialized with the following representation:

| | *Entry 0* | *Entry 1* |
|---|---|---|
| depth cue mode | SUPPRESSED | ALLOWED |
| depth cue reference planes | 0.0, 1.0 | 0.0, 1.0 |
| depth cue scaling | 1.0, 1.0 | 0.0, 1.0 |
| depth cue colour | (INDIRECT, 0) | (INDIRECT, 0) |

For depth cue reference planes and depth cue scaling, the first value is the back reference plane and scaling.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT, OUTIN, nor MO)

103 Ignoring function, setting this bundle table entry would exceed the maximum number of entries allowed in the workstation bundle table

110 Ignoring function, the specified colour model is not available on the workstation

603 Ignoring function, the depth cue index is less than zero

616 Ignoring function, invalid reference planes; *DQMIN > DQMAX*

**SEE ALSO**

SET DEPTH CUE INDEX (3P+)
INQUIRE DEPTH CUE REPRESENTATION (3P+)

**NAME**

SET DISPLAY UPDATE STATE – set Workstation Deferral and Modification Modes

**SYNOPSIS**

**C Syntax**

```
void
pset_disp_upd_st ( ws, def_mode, mod_mode )
Pint            ws;          workstation identifier
Pdefer_mode     def_mode;    deferral mode
Pmod_mode       mod_mode;    modification mode
```

**Required PHIGS Operating States**

(PHOP, WSOP, ∗, ∗)

**DESCRIPTION**

**Purpose**

SET DISPLAY UPDATE STATE sets the deferral and modification modes in the specified workstation's state list. These modes provide control over when and how changes to the structure store and workstation tables actually appear in the display. These values have a great effect on both visual results and performance.

The deferral mode controls when the display is updated (in other words, made entirely correct). This allows you to delay time-consuming or visually-distracting regeneration until a set of related changes are completed or until a correct display is absolutely necessary, as before an operator input interaction can proceed.

The modification mode controls the visual effects that take place while the workstation is postponing display updates. This allows you to selectively change the display, but to avoid regenerating the entire picture.

**C Input Parameters**

*ws* Identifier of the workstation whose update state to set.

*def_mode*

An enumerated value specifying the *deferral mode* to be set for this workstation. Valid values are defined in phigs.h:

| | |
|---|---|
| PDEFER_ASAP | *Make the display visually correct As Soon As Possible* |
| PDEFER_BNIG | *Make the display visually correct Before the Next Interaction Globally* |
| PDEFER_BNIL | *Make the display visually correct Before the Next Interaction Locally* |
| PDEFER_ASTI | *Make the display visually correct At Some Time* |
| PDEFER_WAIT | *Make the display visually correct When the Application Requests It* |

*mod_mode*

An enumerated value specifying the *modification mode* to be set for this workstation. Valid values are defined in phigs.h:

| | |
|---|---|
| PMODE_NIVE | *No Immediate Visual Effects* |
| PMODE_UWOR | *Update Without Regeneration* |
| PMODE_UQUM | *Use Quick Update Methods* |

**Execution**

The SET DISPLAY UPDATE STATE sets the *deferral mode* and *modification mode* entries in the specified workstation's state list. These modes control the degree to which the display must reflect the state of the central structure store and the workstation tables.

*Deferral Modes*

The deferral mode PDEFER_ASAP (*As Soon As Possible*) demands that the display be consistent with the structure store and the workstation state list at all times. This typically causes a regeneration (in other words, clearing of the display and traversal of all posted structure networks) for every change to a structure appearing on the workstation, and for every change to the workstation's state list. The modification mode is

unused when the deferral mode is ASAP.

The deferral mode PDEFER_BNIG (*Before the Next Interaction Globally*) behaves just like PDEFER_ASAP when any input device is active on *any* workstation. The deferral mode PDEFER_BNIL (*Before the Next Interaction Locally*) behaves just like PDEFER_ASAP when any input device is active on *this* workstation. Otherwise, they behave exactly like *At Some Time*, as described below. An input device is considered *active* the entire time it is in EVENT or SAMPLE mode, or while a REQUEST is pending for the device (although nothing but window redisplay can occur while a REQUEST is pending, because it necessarily blocks).

The deferral mode PDEFER_ASTI (*At Some Time*) updates the display only at certain times, which are at the discretion of the PEX implementation and are workstation dependent.

The deferral mode PDEFER_WAIT (*When the Application Requests It*) does not implicitly update the display. Explicit updates are requested by using REDRAW ALL STRUCTURES or UPDATE WORKSTATION with PERFORM.

*Modification Modes*

During modification mode PDEFER_NIVE (*No Immediate Visual Effects*), the only changes to the display are those that would be done in accordance with the deferral mode.

**Note:** *When the Application Requests It* and *No Immediate Visual Effects* used together prohibit any change to the display in response to changes in the central structure store or the workstation state lists.

The modification mode PDEFER_UWOR (*Update Without Regeneration*) performs all updates that can be realized immediately without regenerating the entire display. This includes the actions whose *dynamic modification accepted* entries in the workstation description table are set to *Immediately*.

The modification mode PDEFER_UQUM (*Use Quick Update Methods*) allows use of workstation dependent simulations of changes that cannot be performed immediately unless the display is regenerated. These simulations are described in the section *Available Quick Update Methods*. Actions for which a *quick update method* is available have *dynamic modification accepted* entries in the workstation description table set to *Can Be Simulated*. As in *Update Without Regeneration* mode, actions whose *dynamic modification accepted* entries in the workstation description table are set to *Immediately* are performed immediately. Unless an action can be simulated or the update performed immediately, the update is deferred, as if the modification mode were *No Immediate Visual Effects*.

*State of Visual Representation*

The workstation's *state of visual representation* indicates whether the display is *Correct*, *Simulated* (but no updates have been deferred), or *Deferred*. This state list value may be obtained using INQUIRE DISPLAY UPDATE STATE(3P).

Changing the *deferral mode* causes a regeneration, if the display is not correct and the new *deferral mode* requires it. Changing the *modification mode* has no retroactive effect on previous changes that have been simulated or deferred.

*Available Quick Update Methods*

Quick update methods are dependent on the server implementation. The PEX-SI API does not provide any quick updates.

*Interactions between Deferral and Modification Modes and X Window System*

PEX-SI regenerates the image from the CSS when the window is damaged by window system events. This can occur even if the deferral mode is *When the Application Requests It* (WAIT) and the modification mode is *No Immediate Visual Effects*, which used together prohibit implicit changes to the display. However, only the damaged portion of the PEX-SI canvas is repainted. Hence, the regeneration may use newer information than that which generated the older, out-of-date, but not damaged portion, which is also visible and may not match the newly drawn portion. To remove any visible mismatch, the entire window can be updated by selecting *Refresh* from the canvas's *Frame* menu (assuming that the user is using XView).

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (in other words, the workstation category is neither OUTPUT, OUTIN, nor MO)

**SEE ALSO**

UPDATE WORKSTATION (3P)
REDRAW ALL STRUCTURES (3P)
INQUIRE DISPLAY UPDATE STATE (3P)
INQUIRE DEFAULT DISPLAY UPDATE STATE (3P)

**NAME**

SET EDGE COLOUR – create structure element to set the current edge direct colour attribute

**SYNOPSIS**

**C Syntax**

```
void
pset_edge_colr ( colour )
Pgcolr          *colour;          edge colour
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET EDGE COLOUR creates a structure element to set the current edge direct colour attribute.

When the current edge colour Aspect Source Flag (ASF) is set to INDIVIDUAL, the current value of this attribute applies to the following output primitives:

- FILL AREA SET
- FILL AREA SET 3
- FILL AREA SET 3 WITH DATA
- GENERALIZED DRAWING PRIMITIVE (*Fill Area Set Primitives*)
- GENERALIZED DRAWING PRIMITIVE 3 (*Fill Area Set Primitives*)
- NON-UNIFORM B-SPLINE SURFACE
- SET OF FILL AREA SET 3 WITH DATA
- QUADRILATERAL MESH 3 WITH DATA
- TRIANGLE STRIP 3 WITH DATA

**C Input Parameters**

*colour* Edge colour. Pgcolr is defined as:

```
typedef struct {
        Pint            type;           /* indirect, RGB, CIE, HSV, HLS */
        union {
          Pint          ind;            /* colour table index */
          struct {
                Pfloat          x;      /* red, hue, etc. */
                Pfloat          y;      /* green, saturation, lightness, etc. */
                Pfloat          z;      /* blue, value, saturation, etc. */
          } general;
        } val;
} Pgcolr;
```

**Execution**

Depending on the edit mode, a SET EDGE COLOUR element with the value edge colour in the specified colour model is inserted into the open structure after the element pointer or replaces the element pointed at by the element pointer. The element pointer is then updated to point to this SET EDGE COLOUR element.

When the current edge colour ASF is set to INDIVIDUAL, filled area output primitives that follow in the structure network are drawn using the current edge colour.

When the current edge colour ASF is set to BUNDLED, the edge colour is taken from the workstation extended representation indicated by the current edge index. In this case, the edge colour set with SET EDGE COLOUR has no effect.

The standard PHIGS function SET EDGE COLOUR INDEX is equivalent to calling to SET EDGE COLOUR with type INDIRECT.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

COLOUR (7P)
SET EDGE REPRESENTATION PLUS (3P+)
SET EDGE COLOUR INDEX (3P)

**NAME**

SET EDGE COLOUR INDEX – create structure element to set the current edge colour index attribute

**SYNOPSIS**

**C Syntax**

```
void
pset_edge_colr_ind ( index )
Pint            index;          edge colour index
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET EDGE COLOUR INDEX creates a structure element containing a value for the current edge colour index attribute. When the current edge colour index Aspect Source Flag (ASF) is set to INDIVIDUAL, this attribute indexes the colour representation that applies to the output primitives:

- FILL AREA SET
- FILL AREA SET 3
- FILL AREA SET 3 WITH DATA
- GENERALIZED DRAWING PRIMITIVE *(Fill Area Set Primitives)*
- GENERALIZED DRAWING PRIMITIVE 3 *(Fill Area Set Primitives)*
- NON-UNIFORM B-SPLINE SURFACE
- SET OF FILL AREA SET 3 WITH DATA
- QUADRILATERAL MESH 3 WITH DATA
- TRIANGLE STRIP 3 WITH DATA

**C Input Parameter**

*index* An integer colour index, which selects a colour value from the workstation's colour table.

**Execution**

If the current edit mode is INSERT, then a SET EDGE COLOUR INDEX element is inserted into the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, then the new SET EDGE COLOUR INDEX element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

When the SET EDGE COLOUR INDEX element is traversed, the current edge colour index entry in the traversal state list is set to the colour index.

When the current edge colour index ASF is set to INDIVIDUAL, the fill area output primitives that follow in the structure network are filled with the colour representation selected by the current edge colour index from the workstation's colour table.

If the colour index specified is not available on the workstation, then colour index 1 is used.

When the current edge colour index ASF is set to BUNDLED, the edge colour index is taken from the workstation's representation indicated by the current edge index. In this case, the colour index set with SET EDGE COLOUR INDEX has no effect.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

113 Ignoring function, the colour index value is less than zero

**SEE ALSO**

SET COLOUR REPRESENTATION (3P)
SET EDGE FLAG (3P)

SET INDIVIDUAL ASF (3P)
SET EDGE REPRESENTATION (3P)

**NAME**

SET EDGE FLAG – create a structure element to set the current edge flag attribute

**SYNOPSIS**

**C Syntax**

```
void
pset_edge_flag ( edge_flag )
Pedge_flag          edge_flag;          edge flag
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET EDGE FLAG creates a structure element containing a value for the current edge flag attribute. When the current edge flag Aspect Source Flag (ASF) is set to INDIVIDUAL, this attribute defines whether or not edges are drawn on the following output primitives:

- FILL AREA SET
- FILL AREA SET 3
- FILL AREA SET 3 WITH DATA
- GENERALIZED DRAWING PRIMITIVE (*Fill Area Set Primitives*)
- GENERALIZED DRAWING PRIMITIVE 3 (*Fill Area Set Primitives*)
- SET OF FILL AREA SET 3 WITH DATA
- NON-UNIFORM B-SPLINE SURFACE
- QUADRILATERAL MESH 3 WITH DATA
- TRIANGLE STRIP 3 WITH DATA

**C Input Parameter**

*edge_flag*

The edge flag is an enumerated type defined in phigs.h to have the following values:

| | |
|---|---|
| PEDGE_OFF | *Off* |
| PEDGE_ON | *On* |

**Execution**

If the current edit mode is INSERT when SET EDGE FLAG is called, a SET EDGE FLAG element is inserted into the currently open structure after the element pointed to by the current *element pointer*. If the edit mode is REPLACE, the new SET EDGE FLAG element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

When the SET EDGE FLAG element is traversed, the current edge flag entry in the PHIGS traversal state list is set to edge flag. When the edge flag ASF is set to INDIVIDUAL the edges of the fill area set output primitives that follow in the structure network are drawn when edge flag is On, and are not drawn (the default) when edge flag is Off.

When the current edge flag ASF is set to BUNDLED, the edge flag setting is taken from the workstation's representation indicated by the current edge index. In this case, the edge flag set with SET EDGE FLAG has no effect.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

SET INDIVIDUAL ASF (3P)
SET EDGE REPRESENTATION (3P)
FILL AREA SET (3P)
FILL AREA SET 3 (3P)
FILL AREA SET 3 WITH DATA (3P+)
SET OF FILL AREA SET 3 WITH DATA (3P+)
NON-UNIFORM B-SPLINE SURFACE (3P+)
QUADRILATERAL MESH 3 WITH DATA (3P+)
TRIANGLE STRIP 3 WITH DATA (3P+)

**NAME**

SET EDGE INDEX – create a structure element containing an edge representation index attribute

**SYNOPSIS**

**C Syntax**

```
void
pset_edge_ind ( index )
Pint            index;          edge index
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET EDGE INDEX creates a structure element containing an edge index number that selects an entry from the workstation's edge bundle table. This attribute applies to the output primitives

- FILL AREA SET
- FILL AREA SET 3
- GENERALIZED DRAWING PRIMITIVE (*Fill Area Set Primitives*)
- GENERALIZED DRAWING PRIMITIVE 3 (*Fill Area Set Primitives*)
- NON-UNIFORM B-SPLINE SURFACE
- QUADRILATERAL MESH 3 WITH DATA
- TRIANGLE STRIP 3 WITH DATA

If the current edit mode is INSERT, a SET EDGE INDEX element is inserted into the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, then the new SET EDGE INDEX element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

**C Input Parameter**

*index* An edge index for edge representation on the workstation.

**Execution**

When the SET EDGE INDEX element is traversed, the current edge index value is set to the edge index that specifies an entry from the workstation's edge bundle table. The default edge index is 1, and if the edge index specified is not available on the workstation, then 1 is used.

An edge representation contains values for the following attributes:

- edge flag
- edgetype
- edgewidth scale factor
- edge colour index

FILL AREA output primitives, which follow in the structure network, use the values from the specified representation for those edge attributes whose Aspect Source Flag (ASF) is set to BUNDLED. When the ASFs are INDIVIDUAL, the attributes come from the appropriate SET *attribute* elements.

Edge representations are defined with SET EDGE REPRESENTATION.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

100 Ignoring function, the bundle index value is less than one

**SEE ALSO**

SET EDGE FLAG (3P)
SET INDIVIDUAL ASF (3P)
SET EDGE REPRESENTATION (3P)
SET EDGE REPRESENTATION PLUS (3P+)

**NAME**

SET EDGE REPRESENTATION – define an edge attribute bundle on the workstation

**SYNOPSIS**

**C Syntax**

```
void
pset_edge_rep ( ws, index, rep )
Pint            ws;          workstation identifier
Pint            index;       edge bundle index
Pedge_bundle    *rep;        edge representation pointer
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

SET EDGE REPRESENTATION defines a bundle of edge attributes for a specified entry in the workstation's edge bundle table. Depending on the ASF for each of the edge attributes, the bundled attributes may apply to the following primitives:

- FILL AREA SET
- FILL AREA SET 3
- GENERALIZED DRAWING PRIMITIVE (*Fill Area Set Primitives*)
- GENERALIZED DRAWING PRIMITIVE 3 (*Fill Area Set Primitives*)
- NON-UNIFORM B-SPLINE SURFACE
- SET OF FILL AREA SET 3 WITH DATA
- QUADRILATERAL MESH 3 WITH DATA
- TRIANGLE STRIP 3 WITH DATA

**C Input Parameters**

*ws* The identifier of the workstation for which the edge representation is being defined.

*index* The *edge index* of the entry being defined.

*rep* A pointer to a structure containing the attribute values defining the edge representation, defined as follows:

```
typedef struct {
        Pedge_flag  flag;       /* edge flag */
        Pint        type;       /* edgetype */
        Pfloat      width;      /* edgewidth scale factor */
        Pint        colr_ind;   /* edge colour index */
} Pedge_bundle;
```

*rep→flag* is an enumerated value, specifying whether edges are to be displayed or not. Valid values are:

| | |
|---|---|
| PEDGE_OFF | *Off* |
| PEDGE_ON | *On* |

*rep->type* is an enumerated value, specifying one of the following:

| | | |
|---|---|---|
| 1 | PLINE_SOLID | *Solid* |
| 2 | PLINE_DASH | *Dashed* |
| 3 | PLINE_DOT | *Dotted* |
| 4 | PLINE_DOT_DASH | *Dot-dashed* |

Support for edgetypes is workstation-dependent.

*rep→width* is an edgewidth scale factor value.

*rep→colr_ind* is the edge colour index, which selects a colour value from the workstation's colour table.

**Execution**

When SET EDGE REPRESENTATION is called, the edge index entry in the table of defined edge representations on the workstation is set to the edge flag, edgetype, edgewidth scale factor, and edge colour index values.

When area-defining output primitives are displayed, the edge representation specified by the *current edge index* entry in the traversal state list provides the edge attributes for which the Aspect Source Flag (ASF) is BUNDLED. For example, when the current edgetype ASF is set to BUNDLED, the effective edgetype is the edgetype attribute in the edge representation selected by the current edge index. The current edge index is set by SET EDGE INDEX elements.

The edge bundle table is numbered from 1.

See GENERALIZED DRAWING PRIMITIVE (3P) and GENERALIZED DRAWING PRIMITIVE 3 (3P) to determine which of the generalized primitives use the fill area set attributes.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (in other words, the workstation category is neither OUTPUT, OUTIN, nor MO)

100 Ignoring function, the bundle index value is less than one

103 Ignoring function, setting this bundle table entry would exceed the maximum number of entries allowed in the workstation bundle table

113 Ignoring function, the colour index value is less than zero

107 Ignoring function, the specified edge type is not available on the specified workstation

**SEE ALSO**

SET EDGE INDEX (3P)
SET EDGE FLAG (3P)
SET EDGE COLOUR INDEX (3P)
SET INDIVIDUAL ASF (3P)
INQUIRE EDGE REPRESENTATION (3P)
SET EDGE REPRESENTATION PLUS (3P+)

**NAME**

SET EDGE REPRESENTATION PLUS – set an extended edge attribute bundle on a workstation

**SYNOPSIS**

**C Syntax**

```
void
pset_edge_rep_plus ( ws, index, rep )
Pint            ws;          /* workstation identifier */
Pint            index;       /* edge bundle index number */
Pedge_bundle_plus *rep;      /* edge representation pointer */
```

**Required PHIGS Operating States**

(PHOP, WSOP, ∗, ∗)

**DESCRIPTION**

**Purpose**

SET EDGE REPRESENTATION PLUS sets an extended edge attribute bundle on a workstation.

**C Input Parameters**

*ws* The identifier of the workstation.

*index* This is an index to the workstation edge bundle table.

*rep* A pointer to a Pedge_bundle_plus structure. Pedge_bundle_plus is defined as:

```
typedef struct {
        Pedge_flag      flag;       /* edge flag */
        Pint            type;       /* edgetype */
        Pfloat          width;      /* edgewidth scale factor */
        Pgcolr          colr;       /* edge colour */
} Pedge_bundle_plus;
```

Pedge_flag is defined as:

```
typedef enum {
        PEDGE_OFF,
        PEDGE_ON
} Pedge_flag;
```

The edge type values are:

| | | |
|---|---|---|
| 1 | PLINE_SOLID | *Solid* |
| 2 | PLINE_DASH | *Dashed* |
| 3 | PLINE_DOT | *Dotted* |
| 4 | PLINE_DOT_DASH | *Dot-dashed* |

Pgcolr is defined as:

```
typedef struct {
        Pint                type;       /* indirect, RGB, CIE, HSV, HLS */
        union {
            Pint            ind;        /* index in workstation colour bundle table */
            struct {
                Pfloat      x;          /* red, hue, etc. */
                Pfloat      y;          /* green, saturation, lightness, etc. */
                Pfloat      z;          /* blue, value, saturation, etc. */
            } general;
        } val;
} Pgcolr;
```

**Execution**

In the edge bundle table of the workstation state list, the given index is associated with the specified parameters.

The edgewidth scale factor is a scale factor applied to nominal edgewidth. During structure traversal, the edgewidth scale factor is multiplied by the nominal edgewidth on a workstation. The result is mapped by the workstation to the nearest available edgewidth.

Edge colour is a general colour, either an indirect or a direct colour in a specified colour model. If an indirect colour is specified and the specified colour index is not available during structure traversal, then colour index one is used.

The extended edge bundle table in the workstation state list has predefined entries taken from the workstation description table. A number of these entries are predefined for every workstation of category OUTPUT or OUTIN. Any table entry, including the predefined entries, may be redefined with this function.

When area defining primitives are displayed during structure traversal, the current edge index in the PHIGS traversal state list refers to an entry in the extended edge bundle table. If the current edge index is not present in the extended edge bundle table, then edge index one is used. The setting of the corresponding Aspect Source Flags(ASFs) determines which aspects in the entry are used.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT, OUTIN, nor MO)

100 Ignoring function, the bundle index value is less than one

103 Ignoring function, setting this bundle table entry would exceed the maximum number of entries allowed in the workstation bundle table

107 Ignoring function, the specified edgetype is not available on the specified workstation

110 Ignoring function, the specified colour model is not available on the workstation

113 Ignoring function, the colour index value is less than zero

**SEE ALSO**

SET INDIVIDUAL ASF (3P)
INQUIRE EDGE REPRESENTATION PLUS (3P+)
SET EDGE REPRESENTATION (3P)

**NAME**

SET EDGETYPE – create a structure element to set the current edgetype attribute

**SYNOPSIS**

**C Syntax**

```
void
pset_edgetype ( edgetype )
Pint            edgetype;          edgetype
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET EDGETYPE creates a structure element structure containing a value for the current edgetype attribute.

When the current edgetype Aspect Source Flag (ASF) is set to INDIVIDUAL, this attribute defines the edgetype to be applied to the following output primitives:

- FILL AREA SET
- FILL AREA SET 3
- FILL AREA SET 3 WITH DATA
- SET OF FILL AREA SET 3 WITH DATA
- NON-UNIFORM B-SPLINE SURFACE
- QUADRILATERAL MESH 3 WITH DATA
- TRIANGLE STRIP 3 WITH DATA

**C Input Parameter**

*edgetype*

An integer value specifying one of the following edge types:

| | | |
|---|---|---|
| 1 | PLINE_SOLID | *Solid* |
| 2 | PLINE_DASH | *Dashed* |
| 3 | PLINE_DOT | *Dotted* |
| 4 | PLINE_DOT_DASH | *Dot-dashed* |

Support for edgetypes is workstation dependent.

**Execution**

If the current edit mode is INSERT, a SET EDGETYPE element is inserted into the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, then the new SET EDGETYPE element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

When the SET EDGETYPE element is traversed, the current edgetype entry in the PHIGS traversal state list is set to edgetype. When the edge flag attribute is set to ON and the current edgetype ASF is set to INDIVIDUAL, the edges of the fill area set output primitives that follow in the structure network are drawn with the edgetype specified.

If the edge flag is set to OFF, then these primitives are drawn without edges, and only the interior fill is displayed. If the edge flag is OFF and the interior style is *Empty*, then the primitive is invisible.

When the edge flag is ON and the current edgetype ASF is set to BUNDLED, the edgetype is taken from the workstation's representation indicated by the current edge index. In this case, the edgetype set with SET EDGETYPE has no effect.

If the edgetype specified is not available on the workstation to which the structure is posted, then edgetype 1 *(Solid)* is used.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

SET EDGE FLAG (3P)
SET INDIVIDUAL ASF (3P)
SET EDGE REPRESENTATION (3P)
INQUIRE EDGE REPRESENTATION (3P)
INQUIRE EDGE FACILITIES (3P)
FILL AREA SET (3P)
FILL AREA SET 3 (3P)
FILL AREA SET 3 WITH DATA (3P+)
SET OF FILL AREA SET 3 WITH DATA (3P+)
NON-UNIFORM B-SPLINE SURFACE (3P+)
QUADRILATERAL MESH 3 WITH DATA (3P+)
TRIANGLE STRIP 3 WITH DATA (3P+)

**NAME**

SET EDGEWIDTH SCALE FACTOR – create structure element to set the current edgewidth scale factor attribute

**SYNOPSIS**

**C Syntax**

```
void
pset_edgewidth ( scale )
Pfloat          scale;          edgewidth scale factor
```

**Required PHIGS Operating States**

(PHOP, ∗, STOP, ∗)

**DESCRIPTION**

**Purpose**

SET EDGEWIDTH SCALE FACTOR creates a structure element containing a value for the current edgewidth scale factor attribute. When the current edgewidth scale factor Aspect Source Flag (ASF) is set to INDIVIDUAL, this attribute defines the current edgewidth applied to the output primitives:

- FILL AREA SET
- FILL AREA SET 3
- FILL AREA SET 3 WITH DATA
- GENERALIZED DRAWING PRIMITIVE *(Fill Area Primitives)*
- GENERALIZED DRAWING PRIMITIVE 3 *(Fill Area Primitives)*
- NON-UNIFORM B-SPLINE SURFACE
- SET OF FILL AREA SET 3 WITH DATA
- QUADRILATERAL MESH 3 WITH DATA
- TRIANGLE STRIP 3 WITH DATA

**C Input Parameter**

*scale* A real value specifying the edgewidth scale factor.

**Execution**

If the current edit mode is INSERT when SET EDGEWIDTH SCALE FACTOR is called, a SET EDGEWIDTH SCALE FACTOR element is inserted into the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, the new element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

When the SET EDGEWIDTH SCALE FACTOR element is traversed, the current edgewidth scale factor entry in the traversal state list is set to edgewidth scale factor.

When the edge flag is set to ON and the current edgewidth scale factor ASF is set to INDIVIDUAL, the edges of the fill area output primitives that follow in the structure network are drawn with the edgewidth specified.

If the edge flag is OFF, these primitives are drawn without edges and only the interior fill is displayed. When the edge flag is ON and the edgewidth scale factor ASF is set to BUNDLED, the edgewidth scale factor is taken from the workstation's representation indicated by the current edge index. In this case, the edgewidth value set with SET EDGEWIDTH SCALE FACTOR has no effect.

The width of the edge drawn is determined by applying the current edgewidth scale factor to the nominal edgewidth, and this result is then mapped to the nearest edgewidth supported on the workstation.

**ERRORS**

005 Ignoring function, function requires state (PHOP, ∗, STOP, ∗)

**SEE ALSO**

SET EDGE FLAG (3P)

SET INDIVIDUAL ASF (3P)
SET EDGE REPRESENTATION (3P)

**NAME**

SET EDIT MODE – set the edit mode controlling how new structure elements are added to the open structure

**SYNOPSIS**

**C Syntax**

```
void
pset_edit_mode ( mode )
Pedit_mode          mode;                    edit mode
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

The SET EDIT MODE subroutine sets the edit mode controlling how PHIGS subroutines that create new structure elements will add the new element to the open structure. The mode may be Insert or Replace.

Insert is the default.

**C Input Parameter**

*mode* An enumerated variable specifying the edit mode to be used to add new elements to the open structure. Valid values (defined in phigs.h) are:

PEDIT_INSERT
PEDIT_REPLACE

**Execution**

The SET EDIT MODE subroutine sets the edit mode in the traversal state list to Insert or Replace. The value in this entry controls how PHIGS subroutines that create new structure elements add the new element to the currently open structure.

While the edit mode is Insert, new structure elements are inserted into the open structure after the element pointed to by the current element pointer.

While the edit mode is Replace, new structure elements replace the element in the open structure pointed to by the current element pointer.

**ERRORS**

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

**SEE ALSO**

INQUIRE EDIT MODE (3P)
DELETE ELEMENT (3P)

**NAME**

SET ELEMENT POINTER – set element pointer to specified pointer value

**SYNOPSIS**

**C Syntax**

```
void
pset_elem_ptr ( ep_value )
Pint            ep_value;          element pointer value
```

**Required PHIGS Operating States**

(PHOP, ∗, STOP, ∗)

**DESCRIPTION**

**Purpose**

Use SET ELEMENT POINTER to set the element pointer in the currently open structure to point to the element numbered *ep_value*. Editing of the structure contents, such as adding or deleting elements, is done in relation to the current element pointer position.

**C Input Parameter**

*ep_value*

Specifies the number of the structure element in the currently open structure to which the element pointer should be set.

**Execution**

The SET ELEMENT POINTER subroutine sets the current element pointer to *ep_value*. The element pointer value is the number of the element in the currently open structure at which operations on the structure contents begin. For example, subroutines that add new structure elements put the new element into the currently open structure in relation to the element pointer; the new element is inserted into the structure after the element pointed to by the element pointer, or replaces the element pointed to by the element pointer.

If the pointer value is specified as less than 0, the element pointer is set to 0. If the pointer value is greater than the number of elements in the open structure, the element pointer will be set to the last element in the structure.

**ERRORS**

005 Ignoring function, function requires state (PHOP, ∗, STOP, ∗)

**SEE ALSO**

OFFSET ELEMENT POINTER (3P)
INQUIRE ELEMENT POINTER (3P)
INQUIRE CURRENT ELEMENT TYPE AND SIZE (3P)
SET ELEMENT POINTER AT LABEL (3P)
SET ELEMENT POINTER AT PICK IDENTIFIER (3P+)
ELEMENT SEARCH (3P)

**NAME**

SET ELEMENT POINTER AT LABEL – set the element pointer to the next occurrence of the specified label

**SYNOPSIS**

**C Syntax**

```
void
pset_elem_ptr_label ( label_id )
Pint            label_id;          label identifier
```

**Required PHIGS Operating States**

(PHOP, ∗, STOP, ∗)

**DESCRIPTION**

**Purpose**

SET ELEMENT POINTER AT LABEL sets the element pointer of the currently open structure to the next occurrence in the structure of an element containing the specified label identifier. Label structure elements are created by the LABEL subroutine.

**C Input Parameter**

*label_id* Specifies the label identifier in the open structure at which to set the element pointer.

**Execution**

SET ELEMENT POINTER AT LABEL searches for the label identifier from the structure element following the current element pointer to the end of the open structure. If an occurrence of *label identifier* is not found between the current element pointer and the end of the structure, the subroutine does not move the element pointer and generates an error.

**ERRORS**

005 Ignoring function, function requires state (PHOP, ∗, STOP, ∗)

205 Ignoring function, the label does not exist in the open structure between the element pointer and the end of the structure

**SEE ALSO**

OFFSET ELEMENT POINTER (3P)
INQUIRE ELEMENT POINTER (3P)
INQUIRE CURRENT ELEMENT TYPE AND SIZE (3P)
SET ELEMENT POINTER (3P)
SET ELEMENT POINTER AT PICK IDENTIFIER (3P+)
ELEMENT SEARCH (3P)
LABEL (3P)

**NAME**

SET ERROR HANDLING – Set the error handling function -- C binding only

**SYNOPSIS**

**C Syntax**

```
void
pset_err_hand ( new_err_hand, old_err_hand )
void          (*new_err_hand) ();      application's error handling function
void          (**old_err_hand) ();     OUT old error handling function
```

**Required PHIGS Operating States**

(*, *, *, *)

**DESCRIPTION**

SET ERROR HANDLER sets the error handling function that will be called when PHIGS reports an error. The default function is perr_hand. See the ERROR HANDLING man page for a description of the parameters passed to the error handling function.

**Purpose**

Use SET ERROR HANDLER is used to set the error handling function to an application's own error handling function.

When the error handling mode is ON, ERROR HANDLING is called by PHIGS functions when they detect an error.

An application may replace the default ERROR HANDLING function, using this function, in order to modify the default response to errors, which is simply to call ERROR LOGGING. Any such replacement must have the name above and accept the same parameters.

**C Input Parameters**

*new_err_hand*

A function that is to be the new error handling function.

*old_err_hand*

A pointer to a function pointer where the current error handling function will be returned.

**Execution**

SET ERROR HANDLER sets the error handling function that will be called when PHIGS reports an error. The error handling function is called by PHIGS when an error is detected in a non-inquiry function and the error handling mode is ON.

See OPEN PHIGS for a description of the PHIGS interaction with the error file.

**ERRORS**

No Errors

**SEE ALSO**

ERROR HANDLING (3P)
ERROR LOGGING (3P)
SET ERROR HANDLING MODE (3P)

**NAME**

SET ERROR HANDLING MODE – set the current error handling mode

**SYNOPSIS**

**C Syntax**

```
void
pset_err_hand_mode ( mode )
Perr_mode          mode;          error handling mode
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

SET ERROR HANDLING MODE enables (ON) or disables (OFF) PHIGS error reporting.

The default error handling mode is ON.

**C Input Parameter**

*mode* An enumerated value specifying whether PHIGS error reporting should be enabled or disabled. Valid values (defined in phigs.h) for this parameter are:

PERR_OFF
PERR_ON

**Execution**

If the error handling mode is on when an error is detected by a PHIGS function, the function calls ERROR HANDLING. If the error handling mode is off, the PHIGS function detecting the error does not call ERROR HANDLING.

When the ERROR HANDLING function is called, it accepts the identification of the error condition, the identification of the subroutine calling it, and the name of the error file. The default PHIGS ERROR HANDLING routine simply calls the ERROR LOGGING function to write this information to the error file.

**ERRORS**

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

**SEE ALSO**

ERROR HANDLING (3P)
INQUIRE ERROR HANDLING MODE (3P)
OPEN PHIGS (3P)

**NAME**

SET FACE CULLING MODE – create a structure element to set the face culling mode

**SYNOPSIS**

**C Syntax**

```
void
pset_face_cull_mode ( mode )
Pcull_mode          mode;                    culling mode
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET FACE CULLING MODE creates a structure element to set the face culling mode. The attribute face culling mode defines control over the visibility of front and back facing portions of area defining primitives. See SET FACE DISTINGUISHING MODE (3P+) for details on front and back face determination.

**C Input Parameters**

*mode* A pointer to a Pcull_mode structure. A Pcull_mode structure is defined as:

```
typedef enum {
        PCULL_NONE,
        PCULL_BACKFACE,
        PCULL_FRONTFACE
} Pcull_mode;
```

**Execution**

Depending on the edit mode, a SET FACE CULLING MODE element is inserted into the open structure after the element pointer or replaces the element pointed at by the element pointer. The element pointer is then updated to point to this SET FACE CULLING MODE element.

When face culling mode is *None*, both front and back facing portions are visible. When face culling mode is *Backface*, back facing portions are invisible. When it is *Frontface*, front facing portions are invisible.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

SET FACE DISTINGUISHING MODE (3P+)

**NAME**

SET FACE DISTINGUISHING MODE – create a structure element to set the face distinguishing mode

**SYNOPSIS**

**C Syntax**

```
void
pset_face_disting_mode ( mode )
Pdisting_mode      mode;                    distinguishing mode
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET FACE DISTINGUISHING MODE creates a structure element to set the face distinguishing mode. The attribute face distinguishing mode defines control over the attributes applied when rendering front and back facing portions of area defining primitives.

An application indirectly specifies which of the two faces of an area-defining primitive is the front face, by the ordering of the vertex data.

A facet of an area defining primitive is defined to be back facing if its geometric normal has a negative *z* component when transformed to NPC; otherwise, the facet is front facing.

If facet normals are supplied, they are used as geometric normals; otherwise, a geometric normal is computed for each facet using a method that depends on the primitive type:

*Fill Area* or *Fill Area Set*

A geometric normal is computed by first determining three points: *A*, *B*, and *C*. Point *A* is the first of the vertices. Point *B* is the next point in the list that is non-coincident with *A*. Point *C* is the next point in the list (after *B*) that is non-colinear with *A* and *B*. The geometric normal is the cross product of the vector extending from *A* to *B* with the vector extending from *A* to *C*. The utility function COMPUTE FILL AREA SET GEOMETRIC NORMAL (3P+) can be used to determine the geometric normal for a fill area set primitive according to the above procedure.

*Triangle Strip Primitive*

Determination of the geometric normal for each triangle is computed as for a fill area, using the vertices of the triangle. Order of facets in triangles is reversed for even numbered facets, with facets numbering starting at 1.

*Quadrilateral Mesh Primitive*

Determination of the geometric normal for each quadrilateral is calculated by taking the cross product of its diagonals.

*Non-uniform B-Spline Surface*

The geometric normal is computed as the normalized cross product of the tangent vector along the *u* direction and the tangent vector along the *v* direction.

**C Input Parameters**

*mode* A pointer to a Pdisting_mode structure. A Pdisting_mode structure is defined as:

```
typedef enum {
        PDISTING_NO,
        PDISTING_YES
} Pdisting_mode;
```

**Execution**

Depending on the edit mode, a SET FACE DISTINGUISHING MODE element is inserted into the open structure after the element pointer, or replaces the element pointed at by the element pointer. The element pointer is then updated to point to the new SET FACE DISTINGUISHING MODE (3P+) element.

When the face distinguishing mode is *No*, the back face attributes are not used, and the front face attributes are used for both front facing and back facing portions of the primitives. When the face distinguishing mode is *Yes*, the back face attributes are used for back facing portions.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

SET FACE CULLING MODE (3P+)

**NAME**

SET GLOBAL TRANSFORMATION – create structure element containing a 2D global modelling transformation matrix

**SYNOPSIS**

**C Syntax**

```
void
pset_global_tran ( xform )
Pmatrix          xform;          transformation matrix
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET GLOBAL TRANSFORMATION creates a structure element containing a 2D global modelling transformation matrix, which, during traversal, replaces the *current global modelling transformation*. Then the local modelling transformation is composited with the new global modelling transformation to create a new composite modelling transformation, which maps the modelling coordinates (MC) used to define individual output primitives to a unified world coordinate (WC) space.

If the current edit mode is INSERT, the SET GLOBAL TRANSFORMATION element is inserted into the open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, the SET GLOBAL TRANSFORMATION element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

**C Input Parameter**

*xform* The $3 \times 3$ homogeneous *transformation matrix*, of type:

```
typedef Pfloat Pmatrix[3][3];
```

**Execution**

When traversal of a structure begins, the initial current local modelling transformation (L) and the current global modelling transformation (G) are both the 3D, $4 \times 4$ identity matrix. The composite modelling transformation (C) within a structure traversal is formed by the matrix multiplication of the current local modelling transformation (L) and the current global modelling transformation (G) as follows:

$$C \leftarrow G \times L$$

PHIGS assumes representation of points as column vectors. Hence, the order of composition in $C \leftarrow G \times L$ (and throughout) is ‘‘post-concatenation’’ or ‘‘post-multiply.’’

When a new structure is traversed, one step in the invocation of the referenced structure is to save the current modelling transformations (L, G, and C). The child structure inherits the parent’s composite modelling transformation (C) as its current global modelling transformation (G), and an identity matrix as its initial current local modelling transformation (L). The parent and child have equal composite modelling transformations (C), because L is the identity. After traversal of the child structure network, the saved transformations are restored so that the parent is unaffected by the execution of a child.

When the SET GLOBAL TRANSFORMATION element is traversed, first the element’s *transformation matrix* is expanded to the 3D equivalent *transformation matrix* (T) and it replaces the current global modelling transformation (G):

$$G' \leftarrow T$$

The current local modelling transformation (L), is then composited with the new current global modelling transformation (G) to calculate the new composite modelling transformation (C).

$$C \leftarrow G' \times L$$

The current composite modelling transformation maps the modelling coordinates, used to define individual output primitives, to world coordinates. Mapping the primitives to the world coordinate space establishes the relation between different objects of the image by redefining the parts in terms of a unified coordinate

space. This allows the application to define different parts of the image in different local modelling coordinates convenient to the objects being defined, and then to apply transformations that will map the local coordinate systems of each part to a single world coordinate (WC) space.

Finally, the viewing mechanism maps WC to device coordinates on the workstation's physical display surface.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

SET LOCAL TRANSFORMATION (3P)
BUILD TRANSFORMATION MATRIX (3P)
TRANSFORM POINT (3P)
SET VIEW REPRESENTATION (3P)
SET GLOBAL TRANSFORMATION 3 (3P)

**NAME**

SET GLOBAL TRANSFORMATION 3 – create structure element containing 3D global modelling transformation matrix

**SYNOPSIS**

**C Syntax**

```
void
pset_global_tran3 ( xform )
Pmatrix3        xform;          transformation matrix
```

**Required PHIGS Operating States**

(PHOP, ∗, STOP, ∗)

**DESCRIPTION**

**Purpose**

SET GLOBAL TRANSFORMATION 3 creates a structure element containing a 3D global modelling transformation matrix, which, during traversal, replaces the *current global modelling transformation*. Then the local modelling transformation is composed with the new global modelling transformation to create a new composite modelling transformation, which maps the modelling coordinates (MC) used to define individual output primitives to a unified world coordinate (WC) space.

If the current edit mode is INSERT, the SET GLOBAL TRANSFORMATION 3 element is inserted into the open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, the SET GLOBAL TRANSFORMATION 3 element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

**C Input Parameter**

*xform* The $4 \times 4$ homogeneous *transformation matrix (T)*, of type:

```
typedef Pfloat Pmatrix3[4][4];
```

**Execution**

When traversal of a structure begins, the initial current local modelling transformation (L) and the current global modelling transformation (G) are both the 3D, $4 \times 4$ identity matrix. The composite modelling transformation (C) within a structure traversal is formed by the matrix multiplication of the current local modelling transformation (L) and the current global modelling transformation (G) as follows:

$$C \leftarrow G \times L$$

PHIGS assumes representation of points as column vectors. Hence, the order of composition in $C \leftarrow G \times L$ (and throughout) is ‘‘post-concatenation’’ or ‘‘post-multiply.’’

When a new structure is traversed, one step in the invocation of the referenced structure is to save the current modelling transformations (L, G, and C). The child structure inherits the parent’s composite modelling transformation (C) as its current global modelling transformation (G), and an identity matrix as its initial current local modelling transformation (L). The parent and child have equal composite modelling transformations (C), because L is the identity. After traversal of the child structure network, the saved transformations are restored so that the parent is unaffected by the execution of a child.

When the SET GLOBAL TRANSFORMATION 3 element is traversed, the element’s *transformation matrix* (T) replaces the current global modelling transformation (G):

$$G' \leftarrow T$$

The current local modelling transformation (L), is then composited with the new current global modelling transformation (G) to calculate the new composite modelling transformation (C).

$$C \leftarrow G' \times L$$

The current composite modelling transformation maps the modelling coordinates, used to define individual output primitives, to world coordinates. Mapping the primitives to the world coordinate space establishes the relation between different objects of the image by redefining the parts in terms of a unified coordinate space. This allows the application to define different parts of the image in different local modelling

coordinates convenient to the objects being defined, and then to apply transformations that will map the local coordinate systems of each part to a single world coordinate (WC) space.

Finally, the viewing mechanism maps WC to device coordinates on the workstation's physical display surface.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

SET LOCAL TRANSFORMATION 3 (3P)
BUILD TRANSFORMATION MATRIX 3 (3P)
TRANSFORM POINT 3 (3P)
SET VIEW REPRESENTATION 3 (3P)
SET GLOBAL TRANSFORMATION (3P)

**NAME**

SET HIGHLIGHTING FILTER – set the workstation's name set filter that determines which primitives are highlighted

**SYNOPSIS**

**C Syntax**

```
void
pset_highl_filter ( ws, filter)
Pint        ws;          workstation identifier
Pfilter     *filter;     highlighting filter
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

SET HIGHLIGHTING FILTER sets the workstation's *highlighting filter*, which is compared to the traversal-time *current name set* of each primitive to determine if the primitive is highlighted.

The filter contains an *inclusion set* and an *exclusion set* of names. During traversal, a primitive is eligible for highlighting if at least one name in the current name set is in the inclusion set and no name in the exclusion set. Each name in the inclusion name set and exclusion name set is a small positive integer.

**C Input Parameters**

*ws* The identifier of the workstation whose highlighting filter is to be set.

*filter* A pointer to a Pfilter structure containing the inclusion set and exclusion set of names. Pfilter is defined in phigs.h as follows:

```
typedef struct {
        Pint_list    incl_set;    /* inclusion set */
        Pint_list    excl_set;    /* exclusion set */
} Pfilter;
```

The Pint_list structure is defined in phigs.h as follows:

```
typedef struct {
        Pint    num_ints;    /* number of Pints in list */
        Pint    *ints;       /* list of integers */
} Pint_list;
```

**Execution**

SET HIGHLIGHTING FILTER sets the workstation's highlighting filter, which contains an inclusion set and an exclusion set of names, both empty by default. A primitive is eligible for highlighting if at least one name in the current name set is in the inclusion set and no name in the current name set is in the exclusion set. This means the exclusion set has precedence over the inclusion set. If the workstation's highlighting filter inclusion set is empty, then no primitives are eligible for highlighting.

If the current name set is empty, then subsequent primitives are not eligible. When traversal of a posted structure network starts, the current name set is empty. During traversal, the member names specified by the ADD NAMES TO SET element are added to the current name set by the union operation on the sets. REMOVE NAMES FROM SET elements remove names from the current name set. The actual appearance of highlighting is workstation dependent.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (in other words, the

workstation category is neither OUTPUT, OUTIN, nor MO)

**SEE ALSO**

ADD NAMES TO SET (3P)
REMOVE NAMES FROM SET (3P)
GENERALIZED STRUCTURE ELEMENT (3P)
INQUIRE HIGHLIGHTING FILTER (3P)

**NAME**

SET HLHSR IDENTIFIER – create a structure element to set the current hidden line/hidden surface removal attribute

**SYNOPSIS**

**C Syntax**

```
void
pset_hlhsr_id ( id )
Pint        id;        HLHSR identifier
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET HLHSR IDENTIFIER creates a structure element containing a value for the Hidden Line and Hidden Surface Removal (HLHSR) primitive attribute. During traversal, this attribute replaces the current HLHSR identifier and is applied to all output primitives that follow in the structure network in a workstation-dependent way.

The HLHSR identifier in the structure network is used in conjunction with the HLHSR mode on the workstation during traversal. Presently, both must be one to enable Hidden Surface Removal.

If the current edit mode is INSERT, then a SET HLHSR IDENTIFIER element is inserted into the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, then the new SET HLHSR IDENTIFIER element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

**C Input Parameter**

*id* The HLHSR identifier value. Presently supported values are:

| | | |
|---|---|---|
| 0 | PHIGS_HLHSR_ID_OFF | *Remove Nothing* |
| 1 | PHIGS_HLHSR_ID_ON | *Remove Hidden Surfaces and Hidden Lines Using Z-buffer* |

**Execution**

When the SET HLHSR IDENTIFIER element is traversed, the current HLHSR identifier entry in the traversal state list is set to HLHSR identifier. The current HLHSR identifier is applied to output primitives that follow in the structure network. The current HLHSR identifier traversal state from the structure network is compared to the current HLHSR mode on the workstation.

HLHSR is performed using a Z-buffer algorithm.

The Z-buffer processing does not work with primitives filled with hatches. When a hatched filled area-defining primitive is to be Z buffered, PEX-SI does not apply the Z-buffer process.

The HLHSR mode for a workstation is set with the SET HLHSR MODE function. The supported HLHSR modes are workstation-dependent.

For systems without Z-buffered hardware, HLHSR is done in software, but with a significant loss of performance.

A generic problem with Z-buffering is that edges on polygons can be drawn with missing pixels. When the high quality Z-buffer identifier is used, PEX-SI attempts to improve the quality of edges on polygons, possibly at the expense of lower performance. The severity of the problem and the amount of improvement is dependent on the hardware in use.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

SET HLHSR MODE (3P)
INQUIRE HLHSR FACILITIES (3P)

**NAME**

SET HLHSR MODE – enable or disable hidden line and hidden surface removal on a workstation

**SYNOPSIS**

**C Syntax**

```
void
pset_hlhsr_mode ( ws, mode )
Pint        ws;        workstation identifier
Pint        mode;      HLHSR mode
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

The SET HLHSR MODE requests a certain Hidden Line and Hidden Surface Removal (HLHSR) mode of the workstation. The workstation's current HLHSR mode enables or disables HLHSR for a workstation. The current HLHSR identifier from the structure network is used in conjunction with the HLHSR mode on the workstation during traversal.

**C Input Parameters**

*ws* The identifier of the workstation whose HLHSR mode is being set.

*mode* The HLHSR mode value. Presently defined values are:

| | | |
|---|---|---|
| 0 | PHIGS_HLHSR_MODE_NONE | *Disable Z-buffering* |
| 1 | PHIGS_HLHSR_MODE_ZBUFF | *Enable Z-buffering* |
| 2 | PHIGS_HLHSR_MODE_PAINTERS | |
| 3 | PHIGS_HLHSR_MODE_SCANLINE | |
| 4 | PHIGS_HLHSR_MODE_LINE_ONLY | |

**Execution**

If the requested HLHSR mode value is supported on the specified workstation, then SET HLHSR MODE immediately sets the requested HLHSR mode entry in the PHIGS workstation state list to the specified mode. The effect of calling SET HLHSR MODE is not visible until the *requested* HLHSR mode replaces the *current* HLHSR mode. The time at which this occurs depends on the workstation's display update state.

This assignment is performed immediately and the HLHSR update state is set to *Not Pending* if any one of the following is true:

- The workstation display update state allows update.
- The workstation modification mode is any value other than *No Immediate Visual Effect*, and the dynamic modification accepted for HLHSR mode entry in the workstation description table is set to *Immediate*.
- The *display space empty* status in the workstation state list is EMPTY.

Otherwise, the HLHSR update state is set to *Pending* and the requested HLHSR mode will not replace the current HLHSR mode until the next time the workstation is updated. The HLHSR update state will be set to *Not Pending* at that time.

During traversal, the workstation's current HLHSR mode entry and the primitive's current HLHSR identifier attribute are both checked.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (that is, the

workstation category is neither OUTPUT, OUTIN, nor MO)

111 Ignoring function, the specified HLHSR mode is not available on the specified workstation

**SEE ALSO**

SET HLHSR IDENTIFIER (3P)
INQUIRE HLHSR MODE (3P)

**NAME**

SET INDIVIDUAL ASF – create structure element to set the current Aspect Source Flag (ASF) for an attribute

**SYNOPSIS**

**C Syntax**

```
void
pset_indiv_asf ( att_id, att_source )
Paspect     att_id;        attribute identifier
Pasf        att_source;    attribute source
```

**Required PHIGS Operating States**

(PHOP, ∗, STOP, ∗)

**DESCRIPTION**

**Purpose**

SET INDIVIDUAL ASF creates a structure element containing the Aspect Source Flag (ASF) value for a specified output primitive attribute. The ASF setting for each attribute determines if the value used for that attribute is taken from a defined attribute bundle (for example, from the polymarker representation selected by the current polymarker index) or from an attribute value defined individually (for example, from the current marker type set by SET MARKER TYPE (3P)).

If the current edit mode is INSERT, the structure element created is inserted into the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, the new SET INDIVIDUAL ASF element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

**C Input Parameters**

*att_id* An enumerated attribute identifier value specifying the attribute for which the ASF is to be set. Possible values are:

```
typedef enum {
        PASPECT_LINETYPE,
        PASPECT_LINEWIDTH,
        PASPECT_LINE_COLR_IND,
        PASPECT_MARKER_TYPE,
        PASPECT_MARKER_SIZE,
        PASPECT_MARKER_COLR_IND,
        PASPECT_TEXT_FONT,
        PASPECT_TEXT_PREC,
        PASPECT_CHAR_EXPAN,
        PASPECT_CHAR_SPACE,
        PASPECT_TEXT_COLR_IND,
        PASPECT_INT_STYLE,
        PASPECT_INT_STYLE_IND,
        PASPECT_INT_COLR_IND,
        PASPECT_EDGE_FLAG,
        PASPECT_EDGETYPE,
        PASPECT_EDGEWIDTH,
        PASPECT_EDGE_COLR_IND,
        PASPECT_LINE_SHAD_METH,
        PASPECT_INT_SHAD_METH,
        PASPECT_REFL_PROPS,
        PASPECT_INT_REFL_EQN,
        PASPECT_BACK_INT_STYLE,
        PASPECT_BACK_INT_STYLE_IND,
```

```
        PASPECT_BACK_INT_COLR,
        PASPECT_BACK_INT_SHAD_METH,
        PASPECT_BACK_REFL_PROPS,
        PASPECT_BACK_INT_REFL_EQN,
        PASPECT_CURVE_APPROX_CRIT,
        PASPECT_SURF_APPROX_CRIT
    } Paspect;
```

*att_source*

An enumerated ASF value specifying the source that should be used for the attribute identifier or *att_id*. Values defined are:

```
typedef enum {
        PASF_BUNDLED,
        PASF_INDIV
} Pasf;
```

**Execution**

When the SET INDIVIDUAL ASF element is encountered during traversal, the ASF entry in the traversal state list for the attribute identifier is set to the ASF value.

The current setting of the ASF for an attribute determines whether the current value of that attribute is taken from an attribute bundle or from an individually specified value set by the appropriate SET INDIVIDUAL ASF attribute function.

For example, when the current text font ASF is set to INDIVIDUAL, the value of the current text font (as set by SET TEXT FONT) is used for text output primitives. When the current text font ASF is set to BUNDLED, the text font from the workstation's text representation indicated by the current text index is used instead.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

PHIGS WORKSTATION DESCRIPTION TABLE (7P)
SET POLYLINE REPRESENTATION (3P)
SET POLYMARKER REPRESENTATION (3P)
SET TEXT REPRESENTATION (3P)
SET INTERIOR REPRESENTATION (3P)
SET EDGE REPRESENTATION (3P)
SET EDGE REPRESENTATION PLUS (3P+)
SET INTERIOR REPRESENTATION PLUS (3P+)
SET POLYLINE REPRESENTATION PLUS (3P+)
SET POLYMARKER REPRESENTATION PLUS (3P+)
SET TEXT REPRESENTATION PLUS (3P+)

**NAME**

SET INTERIOR COLOUR – create structure element to set the interior colour attribute

**SYNOPSIS**

**C Syntax**

```
void
pset_int_colr ( colour )
Pgcolr          *colour;          interior colour
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET INTERIOR COLOUR creates a structure element to set the interior colour attribute.

When the current interior colour Aspect Source Flag (ASF) is set to INDIVIDUAL, the current value of this attribute defines the interior colour to be applied to the following output primitives:

- FILL AREA
- FILL AREA 3
- FILL AREA SET
- FILL AREA SET 3
- FILL AREA SET 3 WITH DATA
- NON-UNIFORM B-SPLINE SURFACE
- SET OF FILL AREA SET 3 WITH DATA
- QUADRILATERAL MESH 3 WITH DATA
- TRIANGLE STRIP 3 WITH DATA

**C Input Parameters**

*colour* Interior colour. Pgcolr is defined as:

```
typedef struct {
        Pint            type;           /* indirect, RGB, CIE, HSV, HLS */
        union {
          Pint          ind;            /* colour table index */
          struct {
                Pfloat          x;      /* red, hue, etc. */
                Pfloat          y;      /* green, saturation, lightness, etc. */
                Pfloat          z;      /* blue, value, saturation, etc. */
          } general;
        } val;
} Pgcolr;
```

**Execution**

Depending on the edit mode, a SET INTERIOR COLOUR element is inserted into the open structure after the element pointer or replaces the element pointed at by the element pointer. The element pointer is then updated to point to this SET INTERIOR COLOUR element.

When the current interior colour ASF is set to INDIVIDUAL, filled-area output primitives that follow in the structure network are filled using the current interior colour.

When the current interior colour ASF is set to BUNDLED, the interior colour is taken from the workstation extended representation indicated by the current interior index. In this case, the interior colour set with SET INTERIOR COLOUR has no effect.

The standard PHIGS function SET INTERIOR COLOUR INDEX is equivalent to calling to SET INTERIOR COLOUR with type INDIRECT.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

COLOUR (7P)
SET INTERIOR REPRESENTATION PLUS (3P+)
SET INTERIOR COLOUR INDEX (3P)

**NAME**

SET INTERIOR COLOUR INDEX – create structure element to set the current interior colour index attribute

**SYNOPSIS**

**C Syntax**

```
void
pset_int_colr_ind ( index )
Pint            index;          interior colour index
```

**Required PHIGS Operating States**

(PHOP, ∗, STOP, ∗)

**DESCRIPTION**

**Purpose**

SET INTERIOR COLOUR INDEX creates a structure element containing a value for the current interior colour index attribute.

When the current interior colour index Aspect Source Flag (ASF) is set to INDIVIDUAL, this attribute indexes the current colour representation that applies to the following output primitives:

- FILL AREA
- FILL AREA 3
- FILL AREA SET
- FILL AREA SET 3
- NON-UNIFORM B-SPLINE SURFACE
- FILL AREA SET 3 WITH DATA
- QUADRILATERAL MESH 3 WITH DATA
- TRIANGLE STRIP 3 WITH DATA
- SET OF FILL AREA SET 3 WITH DATA

**C Input Parameter**

*index* An integer colour index, which selects a colour value from the workstation's colour table.

**Execution**

If the current edit mode is INSERT, a SET INTERIOR COLOUR INDEX element is inserted into the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, then the new SET INTERIOR COLOUR INDEX element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

When the SET INTERIOR COLOUR INDEX element is traversed, the current interior colour index entry in the traversal state list is set to the colour index.

When the interior colour index ASF is set to INDIVIDUAL, the fill area output primitives that follow in the structure network are filled with the colour representation selected by the current interior colour index from the workstation's colour table.

If the colour index specified is not available on the workstation, colour index 1 is used.

When the current interior colour index ASF is set to BUNDLED, the interior colour index is taken from the workstation's representation indicated by the current interior index. In this case, the colour index set with SET INTERIOR COLOUR INDEX has no effect.

**ERRORS**

005 Ignoring function, function requires state (PHOP, ∗, STOP, ∗)

113 Ignoring function, the colour index value is less than zero

**SEE ALSO**

SET COLOUR REPRESENTATION (3P)
SET INTERIOR STYLE (3P)
SET INDIVIDUAL ASF (3P)
SET INTERIOR REPRESENTATION (3P)
SET INTERIOR COLOUR (3P+)

**NAME**

SET INTERIOR INDEX – create a structure element containing an interior representation index attribute

**SYNOPSIS**

**C Syntax**

```
void
pset_int_ind ( index )
Pint            index;          interior index
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET INTERIOR INDEX creates a structure element containing an interior index number, which selects an entry from the workstation's interior bundle table. This attribute applies to the output primitives

- FILL AREA
- FILL AREA 3
- FILL AREA SET
- FILL AREA SET 3
- NON-UNIFORM B-SPLINE SURFACE
- FILL AREA 3 WITH DATA
- FILL AREA SET 3 WITH DATA
- QUADRILATERAL MESH 3 WITH DATA
- TRIANGLE STRIP 3 WITH DATA

**C Input Parameter**

*index* An *interior index* for an interior representation on the workstation.

**Execution**

If the current edit mode is INSERT, a SET INTERIOR INDEX element is inserted into the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, the new SET INTERIOR INDEX element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

When the SET INTERIOR INDEX element is traversed, the current interior index value is set to the *interior index*, which specifies an entry from the workstation's interior bundle table. The default *interior index* is 1, and if the *interior index* specified is not available on the workstation, 1 will be used.

An interior representation contains values for the following attributes:

- interior style
- interior style index
- interior colour index

Area-defining output primitives which follow in the structure network will use the values from the specified representation for those interior attributes whose Aspect Source Flag (ASF) is set to BUNDLED. When the ASFs are INDIVIDUAL , the attributes come from the appropriate SET attribute elements.

Interior representations are defined with SET INTERIOR REPRESENTATION.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

100 Ignoring function, the bundle index value is less than one

**SEE ALSO**

SET INDIVIDUAL ASF (3P)
SET INTERIOR REPRESENTATION (3P)
SET INTERIOR REPRESENTATION PLUS (3P+)

**NAME**

SET INTERIOR REPRESENTATION – define an interior attribute bundle on the workstation

**SYNOPSIS**

**C Syntax**

```
void
pset_int_rep ( ws, index, rep )
Pint            ws;             workstation identifier
Pint            index;          interior bundle index
Pint_bundle     *rep;           interior representation pointer
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

SET INTERIOR REPRESENTATION defines a bundle of interior attributes for a specified entry in the workstation's interior bundle table.

Depending on the Aspect Source Flag (ASF) for each of the interior attributes, the bundled attributes may apply to the following primitives:

- FILL AREA
- FILL AREA 3
- FILL AREA SET
- FILL AREA SET 3
- NON-UNIFORM B-SPLINE SURFACE
- FILL AREA SET 3 WITH DATA
- QUADRILATERAL MESH 3 WITH DATA
- TRIANGLE STRIP 3 WITH DATA
- SET OF FILL AREA SET 3 WITH DATA

**C Input Parameters**

*ws* The identifier of the workstation for which the interior representation is being defined.

*index* The interior index of the entry being defined.

*rep* A pointer to a Pint_bundle structure containing the interior representation attributes. Pint_bundle is defined as:

```
typedef struct {
        Pint_style  style;      /* interior style */
        Pint        style_ind;  /* interior style index */
        Pint        colr_ind;   /* interior colour index */
} Pint_bundle;
```

*style* is an enumerated value specifying one of the following interior styles. Pint_style is defined as:

```
typedef enum {
        PSTYLE_HOLLOW,    Hollow
        PSTYLE_SOLID,     Solid
        PSTYLE_PAT,       Patterned
        PSTYLE_HATCH,     Hatched
        PSTYLE_EMPTY,     Empty
} Pint_style;
```

*style_ind* is the interior style index, which selects from the workstation's PATTERN or HATCH table.

*colr_ind* is the interior colour index, which selects a colour value from the workstation's colour table.

**Execution**

When SET INTERIOR REPRESENTATION is called, the interior index in the table of defined interior representations on the workstation is set to the interior style, interior style index, and interior colour index values.

When area-defining output primitives are displayed, the interior representation specified by the current interior index entry in the traversal state list provides the interior attributes for which the ASF is BUNDLED. For example, when the current interior style ASF is set to BUNDLED, the *effective* interior style is the interior style attribute in the interior representation selected by the current interior index. The current interior index is set by SET INTERIOR INDEX elements.

The interior bundle table is numbered from 1.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (in other words, the workstation category is neither OUTPUT, OUTIN, nor MO)

100 Ignoring function, the bundle index value is less than one

103 Ignoring function, setting this bundle table entry would exceed the maximum number of entries allowed in the workstation bundle table

108 Ignoring function, the specified interior style is not available on the workstation

112 Ignoring function, the pattern index value is less than one

113 Ignoring function, the colour index value is less than zero

**SEE ALSO**

SET INTERIOR INDEX (3P)
SET INTERIOR STYLE (3P)
SET INDIVIDUAL ASF (3P)
INQUIRE INTERIOR REPRESENTATION (3P)
SET INTERIOR REPRESENTATION PLUS (3P+)

**NAME**

SET INTERIOR REPRESENTATION PLUS – set an extended interior attribute bundle on a workstation

**SYNOPSIS**

**C Syntax**

```
void
pset_int_rep_plus ( wsid, index, rep )
Pint            wsid;           workstation id
Pint            index;          interior index
Pint_bundle_plus *rep;          representation
```

**Required PHIGS Operating States**

(PHOP, WSOP, ∗, ∗)

**DESCRIPTION**

**Purpose**

SET INTERIOR REPRESENTATION PLUS sets a bundle table entry in the workstation extended interior bundle table.

**C Input Parameters**

*wsid* The identifier of the workstation for which the representation is being set.

*index* The index of the entry being set.

*rep* A pointer to a Pint_bundle_plus structure containing the extended interior attributes. Pint_bundle_plus is defined as:

```
typedef struct {
        Pint_style      style;          /* interior style */
        Pint            style_ind;      /* interior style index */
        Pgcolr          colr;           /* interior colour */
        Pint            refl_eqn;       /* reflectance equation */
        Pint            shad_meth;      /* shading method */
        Prefl_props     refl_props;     /* area properties */
        Pint_style      back_style;     /* interior style */
        Pint            back_style_ind; /* interior style index */
        Pgcolr          back_colr;      /* interior colour */
        Pint            back_refl_eqn;  /* back reflectance equation */
        Pint            back_shad_meth; /* back shading method */
        Prefl_props     back_refl_props; /* back area properties */
        Pint            approx_type;    /* approximation method */
        Pfloat          approx_val[2];  /* approximation values, u and v */
} Pint_bundle_plus;
```

Pint_style is defined as:

```
typedef enum {
        PSTYLE_HOLLOW,
        PSTYLE_SOLID,
        PSTYLE_PAT,
        PSTYLE_HATCH,
        PSTYLE_EMPTY
} Pint_style;
```

See SET INTERIOR STYLE for a description of each style.

Pgcolr is defined as:

```
typedef struct {
        Pint                type;           /* indirect, RGB, CIE, HSV, HLS */
        union {
            Pint            ind;            /* index in workstation colour bundle table */
            struct {
                Pfloat      x;              /* red, hue, etc. */
                Pfloat      y;              /* green, saturation, lightness, etc. */
                Pfloat      z;              /* blue, value, saturation, etc. */
            } general;
        } val;
} Pgcolr;
```

Prefl_props is defined as:

```
typedef struct {
        Pfloat      ambient_coef;       /* ambient reflectance coefficient */
        Pfloat      diffuse_coef;       /* diffuse reflectance coefficient */
        Pfloat      specular_coef;      /* specular reflectance coefficient */
        Pgcolr      specular_colr;      /* specular colour */
        Pfloat      specular_exp;       /* specular exponent */
        Pfloat      transpar_coef;      /* transparency coefficient */
} Prefl_props;
```

The values for ambient, diffuse, specular, and transparency coefficients must all be in the range [0,1]. The specular exponent must be greater than zero.

Constants defined for colour type are:

| | | |
|---|---|---|
| 0 | PINDIRECT | *Indirect* |
| 1 | PMODEL_RGB | *Red, Green, Blue* |
| 2 | PMODEL_CIELUV | *CIE* |
| 3 | PMODEL_HSV | *Hue, Saturation, Value* |
| 4 | PMODEL_HLS | *Hue, Lightness, Saturation* |

The index member of the val union is used for type PINDIRECT. The general member is used for the other types.

Predefined reflectance equation values are:

| | | |
|---|---|---|
| 1 | PREFL_NONE | *No Reflectance Calculation Performed* |
| 2 | PREFL_AMBIENT | *Use Ambient Term* |
| 3 | PREFL_AMB_DIFF | *Use Ambient and Diffuse Terms* |
| 4 | PREFL_AMB_DIFF_SPEC | *Use Ambient, Diffuse, and Specular Terms* |

Predefined constants for interior shading method are:

| | | |
|---|---|---|
| 1 | PSD_NONE | *No Shading* |
| 2 | PSD_COLOUR | *Colour Interpolation Shading* |
| 3 | PSD_DOT_PRODUCT | *Dot Product Interpolation Shading* |
| 4 | PSD_NORMAL | *Normal Interpolation Shading* |

Constants defined for surface approximation type are:

1 PSURF_WS_DEP
2 PSURF_CONSTANT_PARAMETRIC_BETWEEN_KNOTS
3 PSURF_CHORDAL_SIZE_WC
4 PSURF_CHORDAL_SIZE_NPC
5 PSURF_CHORDAL_SIZE_DC
6 PSURF_PLANAR_DEVIATION_WC
7 PSURF_PLANAR_DEVIATION_NPC
8 PSURF_PLANAR_DEVIATION_DC

See SET SURFACE APPROXIMATION CRITERIA for a description of these constants.

**Execution**

The extended interior bundle table contains bundled entries of the PHIGS and PHIGS PLUS interior attributes. SET INTERIOR REPRESENTATION PLUS sets the individual entries of this table. During traversal, the attribute values of these bundles are used when the corresponding aspect source flag in the workstation state list is set to BUNDLED.

The PHIGS interior bundle table is a portion of the extended table. The function SET INTERIOR REPRESENTATION sets the following attributes in the extended table: interior style, interior style index, and colour index (colour type of *Indirect* is assumed).

During structure traversal the current interior index refers to an entry in the workstation extended interior bundle table. Depending on the settings of the aspect source flags, the contents of that bundle table entry are used to determine the attributes associated with output primitives that use the extended interior bundle table.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT, OUTIN, nor MO)

100 Ignoring function, the bundle index value is less than one

103 Ignoring function, setting this bundle table entry would exceed the maximum number of entries allowed in the workstation bundle table

108 Ignoring function, the specified interior style is not available on the workstation

110 Ignoring function, the specified colour model is not available on the workstation

112 Ignoring function, the pattern index value is less than one

**SEE ALSO**

SET INDIVIDUAL ASF (3P)
INQUIRE INTERIOR FACILITIES PLUS (3P+)
INQUIRE INTERIOR REPRESENTATION PLUS (3P+)
SET TRIMMING CURVE APPROXIMATION CRITERIA (3P+)
SET INTERIOR STYLE INDEX (3P)
SET INTERIOR REPRESENTATION (3P)

**NAME**

SET INTERIOR SHADING METHOD – create a structure element that sets the current interior shading method

**SYNOPSIS**

**C Syntax**

```
void
pset_int_shad_meth ( method )
Pint            method;          shading method
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET INTERIOR SHADING METHOD places a structure element with the value shading method into the open structure. When the current interior shading method Aspect Source Flag (ASF) is set to INDIVIDUAL, the current value of this attribute applies to the following output primitives:

- FILL AREA
- FILL AREA SET
- FILL AREA SET 3 WITH DATA
- GENERALIZED DRAWING PRIMITIVE *(Fill Area Primitives)*
- GENERALIZED DRAWING PRIMITIVE 3 *(Fill Area Primitives)*
- QUADRILATERAL MESH 3 WITH DATA
- TRIANGLE STRIP 3 WITH DATA
- SET OF FILL AREA SET 3 WITH DATA
- NON-UNIFORM B-SPLINE SURFACE

**C Input Parameters**

*method* The interior shading method.

| | | |
|---|---|---|
| 1 | PSD_NONE | *No Shading* |
| 2 | PSD_COLOUR | *Colour Interpolation Shading* |
| 3 | PSD_DOT_PRODUCT | *Dot Product Interpolation Shading* |
| 4 | PSD_NORMAL | *Normal Interpolation Shading* |

**Execution**

Depending on the edit mode, a SET INTERIOR SHADING METHOD element is inserted into the open structure after the element pointer, or replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the newly created SET INTERIOR SHADING METHOD element.

When the structure is traversed, the current interior shading method entry in the traversal state list is set to the specified value. This value is used with subsequent filled area output primitives when the current interior shading method ASF entry in the traversal state list is INDIVIDUAL. During traversal, if the specified shading method is not available on a workstation, method one (*None*) will be used on that workstation.

When the current interior shading method ASF is set to BUNDLED, the interior shading method is taken from the workstation extended representation indicated by the current interior index. In this case, the interior shading method set with SET INTERIOR SHADING METHOD has no effect.

The interior shading methods supported by a workstation type can be determined with the function INQUIRE INTERIOR FACILITIES PLUS.

When the shading method is NONE, each fill area or fill area set will be rendered in a single colour (this is sometimes called flat shading). If the shading method is NONE and the primitive data implies different colours for each vertex of the primitive, such as when vertex colour is specified or when lighting is active, the colour of the first vertex will be used for all the vertices of the primitive.

When the shading method is COLOUR and the workstation is in index colour mode, then the colours will be interpolated as colour indices instead of direct colours.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

FILL AREA (3P)
FILL AREA SET (3P)
FILL AREA SET 3 WITH DATA (3P+)
QUADRILATERAL MESH 3 WITH DATA (3P+)
TRIANGLE STRIP 3 WITH DATA (3P+)
SET OF FILL AREA SET 3 WITH DATA (3P+)
SET BACK INTERIOR SHADING METHOD (3P+)
SET INDIVIDUAL ASF (3P)
SET INTERIOR REPRESENTATION PLUS (3P+)
INQUIRE INTERIOR FACILITIES PLUS (3P+)
NON-UNIFORM B-SPLINE SURFACE (3P+)

**NAME**

SET INTERIOR STYLE – create structure element to set the current interior style attribute

**SYNOPSIS**

**C Syntax**

**void**
**pset_int_style ( style )**
**Pint_style** **style;** *interior style*

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET INTERIOR STYLE creates a structure element containing a value for the current interior style attribute.

When the current interior style Aspect Source Flag (ASF) is set to INDIVIDUAL, the current value of this attribute defines the interior style to be applied to the following output primitives:

- FILL AREA
- FILL AREA 3
- FILL AREA SET
- FILL AREA SET 3
- NON-UNIFORM B-SPLINE SURFACE
- FILL AREA SET 3 WITH DATA
- QUADRILATERAL MESH 3 WITH DATA
- TRIANGLE STRIP 3 WITH DATA

**C Input Parameters**

All of the following data types are predefined in phigs.h.

*style* An enumerated value specifying one of the following interior styles:

```
typedef enum {
        PSTYLE_HOLLOW,    Hollow
        PSTYLE_SOLID,     Solid
        PSTYLE_PAT,       Patterned
        PSTYLE_HATCH,     Hatched
        PSTYLE_EMPTY      Empty
} Pint_style;
```

**Execution**

If the current edit mode is INSERT, a SET INTERIOR STYLE element is inserted into the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, the new SET INTERIOR STYLE element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

When the SET INTERIOR STYLE element is encountered during traversal, the current interior style entry in the traversal state list is set to the specified interior style.

When the current interior style ASF is set to INDIVIDUAL, area-defining output primitives which follow in the structure network are filled using the current interior style. The interior styles have the following meanings:

*Hollow* No filling, but draw the bounding polyline using the colour specified by the interior colour index. This is the default style and is also used if an interior style is specified that is not available on the workstation.

*Solid* Fill the interior using the colour specified by the interior colour index.

*Patterned* Fill the interior using the interior style index as an index into the workstation pattern table. See SET PATTERN REPRESENTATION.

*Hatched* Fill the interior using the colour specified by the interior colour index and the interior style index as an index into the workstation table of available hatches. Hatches are predefined; they cannot be defined by the application.

*Empty* No filling, but draw the edges in accordance with the edge flag. FILL AREA and FILL AREA 3 primitives will be invisible, but could still be pickable. See SET EDGE FLAG for more information.

When the current interior style ASF is set to BUNDLED, the interior style is taken from the workstation representation indicated by the current interior index. In this case, the interior style set with SET INTERIOR STYLE has no effect.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

SET INDIVIDUAL ASF (3P)
SET INTERIOR REPRESENTATION (3P)
SET EDGE FLAG (3P)
SET INTERIOR STYLE INDEX (3P)
SET PATTERN REPRESENTATION (3P)

**NAME**

SET INTERIOR STYLE INDEX – create a structure element to set the current interior style index attribute

**SYNOPSIS**

**C Syntax**

```
void
pset_int_style_ind ( index )
Pint            index;          interior style index
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET INTERIOR STYLE INDEX creates a structure element containing a value for the current interior style index attribute.

When the current interior style index ASF is set to INDIVIDUAL, the current value of this attribute applies to the following output primitives:

- FILL AREA
- FILL AREA 3
- FILL AREA SET
- FILL AREA SET 3
- FILL AREA SET 3 WITH DATA
- NON-UNIFORM B-SPLINE SURFACE
- QUADRILATERAL MESH 3 WITH DATA
- TRIANGLE STRIP 3 WITH DATA

If the current edit mode is INSERT, a SET INTERIOR STYLE INDEX element is inserted into the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, the new SET INTERIOR STYLE INDEX element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

**C Input Parameter**

*index* The interior style index, which selects from the workstation's PATTERN or HATCH.

**Execution**

When the SET INTERIOR STYLE INDEX element is traversed, the current interior style index entry in the traversal state list is set to the interior style index. The default interior style index is 1.

When the current interior style index ASF is set to INDIVIDUAL, filled area output primitives that follow in the structure network are filled using the interior style index specified. If the interior style index specified is not available on the workstation, HATCH -1 (or GENERAL INTERIOR 1) is used.

When the interior style (either individual or bundled, depending on the interior style ASF) is *Hollow*, *Solid*, or *Empty*, the interior style index does not apply. When the interior style is *Patterned*, the interior style index determines which of the workstation's pattern table entries is used; the interior style index may be called the pattern index in this case. When the interior style is *Hatched*, the interior style index determines which of the workstation's predefined hatches is used; the interior style index may be called the hatch index in this case.

When the current interior style index ASF is set to BUNDLED, the interior style index is taken from the workstation representation indicated by the current interior index. In this case, the interior style index set with SET INTERIOR STYLE INDEX has no effect.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

SET INTERIOR STYLE (3P)
SET PATTERN REPRESENTATION (3P)
INQUIRE INTERIOR FACILITIES (3P)
SET INDIVIDUAL ASF (3P)
SET INTERIOR REPRESENTATION (3P)

**NAME**

SET INVISIBILITY FILTER – set the workstation's name set filter that determines which primitives are invisible

**SYNOPSIS**

**C Syntax**

```
void
pset_invis_filter ( ws, filter )
Pint        ws;        workstation identifier
Pfilter     *filter;   highlighting filter
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

SET INVISIBILITY FILTER sets the workstation's *invisibility filter*, which is compared to the traversal-time *current name set* of each primitive to determine if the primitive is invisible.

The filter contains an *inclusion set* and an *exclusion set* of names. During traversal, a primitive is invisible if at least one name in the current name set is in the inclusion set and no name in the current name set is in the exclusion set. This means the exclusion set has precedence over the inclusion set. Each name in the current name set, inclusion set, and exclusion set is a small positive integer.

**C Input Parameters**

*ws* The identifier of the workstation whose invisibility filter is to be set.

*filter* A pointer to a Pfilter structure containing the inclusion set and exclusion set of names. Pfilter is defined in phigs.h as follows:

```
typedef struct {
        Pint_list       incl_set;       /* inclusion set */
        Pint_list       excl_set;       /* exclusion set */
} Pfilter;
```

The Pint_list structure is defined in phigs.h as follows:

```
typedef struct {
        Pint    num_ints;       /* number of Pints in list */
        Pint    *ints;          /* list of integers */
} Pint_list;
```

**Execution**

SET INVISIBILITY FILTER sets the workstation's invisibility filter, which contains an inclusion set and an exclusion set of names, both empty by default. A primitive is invisible if at least one name in the current name set is in the inclusion set and no name in the current name set is in the exclusion set. If the workstation's invisibility filter inclusion set is empty, then no primitives are invisible.

If the current name set is empty, then subsequent primitives are not invisible. When traversal of a posted structure network starts, the current name set is empty. During traversal, the member names specified by the ADD NAMES TO SET element are added to the current name set by the union operation on the sets. REMOVE NAMES FROM SET elements remove names from the current name set.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (in other words, the workstation category is neither OUTPUT, OUTIN, nor MO)

**SEE ALSO**

ADD NAMES TO SET (3P)
REMOVE NAMES FROM SET (3P)
INQUIRE INVISIBILITY FILTER (3P)

**NAME**

SET LIGHT SOURCE REPRESENTATION – define a light source representation entry in the workstation table of defined light source representations

**SYNOPSIS**

**C Syntax**

```
void
pset_light_src_rep ( ws, index, rep )
Pint                ws;          workstation identifier
Pint                index;       light table index
Plight_src_bundle   *rep;        light source representation
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

SET LIGHT SOURCE REPRESENTATION defines a light source representation for a specified index number on the workstation table of defined light source representations. Each integer entry on this table specifies a light source type (*Ambient, Directional, Positional,* or *Spot*) and a data record containing the parameters needed to define the light source type.

Defined light sources are activated or deactivated with the SET LIGHT SOURCE STATE subroutine. When lighting and shading are applied to a display, the currently active light sources are used, with the current surface properties and interior lighting and shading methods, to render the following area defining primitives: FILL AREA 3, FILL AREA SET 3, and FILL AREA SET 3 WITH DATA.

**C Input Parameters**

*ws* The workstation identifier specifies the workstation for which the light source representation is to be defined. The workstation must have been opened with the OPEN WORKSTATION subroutine before calling SET LIGHT SOURCE REPRESENTATION.

*index* The number of the entry to be defined on the table of defined light source representations, in the specified workstation state list.

*rep* A pointer to light source representation. The Plight_src_bundle data structure contains values needed to define the specified type of light source representation. Plight_src_bundle is defined as:

```
typedef struct {
        Pint            type;       /* light source type */
        Plight_src_rec  rec;        /* light source data record */
} Plight_src_bundle;
```

The predefined light source types are:

| | | |
|---|---|---|
| 1 | PLIGHT_AMBIENT | *Ambient Light Source* |
| 2 | PLIGHT_DIRECTIONAL | *Directional Light Source* |
| 3 | PLIGHT_POSITIONAL | *Positional Light Source* |
| 4 | PLIGHT_SPOT | *Spot Light Source* |

Plight_src_rec is defined as:

```
typedef union {
        Pamb_light_src_rec      ambient;
        Pdir_light_src_rec      directional;
        Ppos_light_src_rec      positional;
        Pspot_light_src_rec     spot;
} Plight_src_rec;
```

Pamb_light_src_rec is defined as:

```
typedef struct {
        Pgcolr          colr;           /* light source colour */
} Pamb_light_src_rec;
```

Pgcolr is defined as:

```
typedef struct {
        Pint                    type;           /* indirect, RGB, CIE, HSV, HLS */
        union {
            Pint                ind;            /* index in workstation colour bundle table */
            struct {
                Pfloat          x;              /* red, hue, etc. */
                Pfloat          y;              /* green, saturation, lightness, etc. */
                Pfloat          z;              /* blue, value, saturation, etc. */
            } general;
        } val;
} Pgcolr;
```

Pdir_light_src_rec is defined as:

```
typedef struct {
        Pgcolr          colr;           /* light source colour */
        Pvec3           dir;            /* light source direction */
} Pdir_light_src_rec;
```

Pvec3 is defined as:

```
typedef struct {
        Pfloat          delta_x;        /* x magnitude */
        Pfloat          delta_y;        /* y magnitude */
        Pfloat          delta_z;        /* z magnitude */
} Pvec3;
```

Ppos_light_src_rec is defined as:

```
typedef struct {
        Pgcolr          colr;           /* light source colour */
        Ppoint3         pos;            /* light source position */
        Pfloat          coef[2];        /* attenuation coefficients */
} Ppos_light_src_rec;
```

Ppoint3 is defined as:

```
typedef struct {
        Pfloat          x;      /* x coordinate */
        Pfloat          y;      /* y coordinate */
        Pfloat          z;      /* z coordinate */
} Ppoint3;
```

Pspot_light_src_rec is defined as:

```
typedef struct {
        Pgcolr          colr;           /* light source colour */
        Ppoint3         pos;            /* light source position */
        Pvec3           dir;            /* light source direction */
        Pfloat          exp;            /* concentration exponent */
        Pfloat          coef[2];        /* attenuation coefficients */
        Pfloat          angle;          /* spread angle */
} Pspot_light_src_rec;
```

**Execution**

When SET LIGHT SOURCE REPRESENTATION is called, the light source index entry in the table of defined light source representations on the workstation is set to the new attribute values.

**ERRORS**

003 Ignoring function, function requires state (PHOP,WSOP,*,*)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT, OUTIN, nor MO)

103 Ignoring function, setting this bundle table entry would exceed the maximum number of entries allowed in the workstation bundle table

169 Ignoring function, the specified light direction is invalid

614 Ignoring function, the light source index is less than one

617 Ignoring function, the specified light source type is not available on the workstation

618 Ignoring function, the specified spot light spread angle is out of range

**SEE ALSO**

SET REFLECTANCE PROPERTIES (3P+)
SET REFLECTANCE EQUATION (3P+)
INQUIRE LIST OF LIGHT SOURCE INDICES (3P+)
INQUIRE LIGHT SOURCE REPRESENTATION (3P+)
INQUIRE LIGHT SOURCE FACILITIES (3P+)
INQUIRE PREDEFINED LIGHT SOURCE REPRESENTATION (3P+)

**NAME**

SET LIGHT SOURCE STATE – create structure element to enable and disable light sources

**SYNOPSIS**

**C Syntax**

```
void
pset_light_src_state ( activation, deactivation )
Pint_list       *activation;        activation list
Pint_list       *deactivation;      deactivation list
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET LIGHT SOURCE STATE creates an element in the currently open structure enabling and disabling specified light source representations on the table of defined light sources stored on the workstation state list. The current light source state determines the number and type of light sources used to shade the primitives that follow in the structure network.

**C Input Parameters**

*activation*

This is a list of light sources to be activated. This structure specifies both an integer value indicating the number of light source representations to be activated and a pointer to a list specifying the index numbers of the light source representations to be activated. Pint_list is defined as:

```
typedef struct {
        Pint    num_ints;       /* number of light sources to activate */
        Pint    *ints;          /* list of light sources to activate */
} Pint_list;
```

*deactivation*

This is a list of light sources to be deactivated. This structure specifies both an integer value indicating the number of light source representations to be deactivated and a pointer to a list specifying the index numbers of the light source representations to be deactivated. Pint_list is defined as:

```
typedef struct {
        Pint    num_ints;       /* number of light sources to activate */
        Pint    *ints;          /* list of light sources to activate */
} Pint_list;
```

**Execution**

Depending on the edit mode, a SET LIGHT SOURCE STATE element is inserted into the open structure after the element pointer or replaces the element pointed at by the element pointer. The element pointer is then updated to point to this SET LIGHT SOURCE STATE element.

When the SET LIGHT SOURCE STATE element is traversed, the light source representations listed in the activation list are added to the current light source state entry in the traversal state list, and the representations listed in the deactivation list are deleted. The light source representations are defined with the SET LIGHT SOURCE REPRESENTATION subroutine.

The current light source state list defines the ambient, diffuse, and specular light sources that may be used when applying lighting and shading to area-defining or polyline primitives. The types of light sources used are defined by the SET INTERIOR SHADING METHOD and SET POLYLINE SHADING METHOD subroutines.

**ERRORS**

005 Ignoring function, function requires state (PHOP,*,STOP,*)

133 Ignoring function, one of the entries in the activation list or deactivation list is less than one (1)

135 Ignoring function, the same entry exists in both the activation list and the deactivation list

**SEE ALSO**

SET LIGHT SOURCE REPRESENTATION (3P+)
INQUIRE LIGHT SOURCE FACILITIES (3P+)

**NAME**

SET LINETYPE – create structure element to set the current linetype attribute

**SYNOPSIS**

**C Syntax**

```
void
pset_linetype ( linetype )
Pint            linetype;          linetype
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET LINETYPE creates a structure element containing a value for the current linetype attribute. When the current linetype ASF (Aspect Source Flag) is set to INDIVIDUAL, the attribute defines the linetype to be applied to the following output primitives:

- POLYLINE
- POLYLINE 3
- POLYLINE SET 3 WITH DATA
- NON-UNIFORM B-SPLINE CURVE

If the current edit mode is INSERT, then a SET LINETYPE element is inserted into the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, then the new SET LINETYPE element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

**C Input Parameter**

*linetype* An integer value specifying a *linetype*; the following line types are defined:

| | | |
|---|---|---|
| 1 | PLINE_SOLID | *Solid* |
| 2 | PLINE_DASH | *Dashed* |
| 3 | PLINE_DOT | *Dotted* |
| 4 | PLINE_DOT_DASH | *Dot-dashed* |

Support for *linetypes* is workstation dependent.

**Execution**

When the SET LINETYPE element is traversed, the current linetype entry in the PHIGS traversal state list is set to *linetype*. When the current linetype ASF is set to INDIVIDUAL, line output primitives which follow in the structure network will be drawn with the *linetype* specified.

When the current linetype ASF is set to BUNDLED, the effective linetype is the linetype attribute in the polyline representation selected by the current polyline index. In this case, the *linetype* set with SET LINETYPE has no effect.

If the *linetype* specified is not available on the workstation, linetype 1 (*Solid*) will be used.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

SET INDIVIDUAL ASF (3P)
SET POLYLINE REPRESENTATION (3P)
INQUIRE POLYLINE REPRESENTATION (3P)
POLYLINE (3P)
POLYLINE 3 (3P)
POLYLINE SET 3 WITH DATA (3P+)
NON-UNIFORM B-SPLINE CURVE (3P+)

**NAME**

SET LINEWIDTH SCALE FACTOR – create structure element to set the current linewidth scale factor attribute

**SYNOPSIS**

**C Syntax**

```
void
pset_linewidth ( width )
Pfloat          width;          linewidth scale factor
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET LINEWIDTH SCALE FACTOR creates a structure element containing a value for the *current linewidth scale factor* attribute. When the *linewidth scale factor ASF* is set to INDIVIDUAL, this attribute defines the current linewidth to be applied to the output primitives:

- POLYLINE
- POLYLINE 3

If the current edit mode is INSERT, a SET LINEWIDTH SCALE FACTOR element is inserted into the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, the new SET LINEWIDTH SCALE FACTOR element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

**C Input Parameter**

*width* A real value specifying the *linewidth scale factor*.

**Execution**

When the SET LINEWIDTH SCALE FACTOR element is traversed, the *current linewidth scale factor* entry in the traversal state list is set to *linewidth scale factor*. When the *current linewidth scale factor ASF* (Aspect Source Flag) is set to INDIVIDUAL, line output primitives that follow in the structure network are drawn using the *linewidth scale factor* specified.

The width of the line drawn is determined by applying the current linewidth scale factor to the *nominal linewidth* as defined in the workstation description table, and this result is then mapped to the nearest linewidth supported on the workstation.

When the *current linewidth scale factor ASF* is set to BUNDLED, the effective linewidth scale factor is the linewidth attribute in the polyline representation selected by the *current polyline index*. In this case, the *linewidth scale factor* value set with SET LINEWIDTH SCALE FACTOR will have no effect.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

SET EDGEWIDTH SCALE FACTOR (3P)
SET INDIVIDUAL ASF (3P)
SET POLYLINE REPRESENTATION (3P)

**NAME**

SET LOCAL TRANSFORMATION – create a structure element containing a 2D Local Modelling Transformation matrix

**SYNOPSIS**

**C Syntax**

```
void
pset_local_tran ( xform, compose_type )
Pmatrix          xform;          transformation matrix
Pcompose_type    compose_type;   composition type
```

**Required PHIGS Operating States**

(PHOP, ∗, STOP, ∗)

**DESCRIPTION**

**Purpose**

SET LOCAL TRANSFORMATION creates a structure element containing a 2D local modelling transformation matrix and a composition type, allowing various objects in the image to be defined in various convenient modelling coordinate systems.

During traversal, the element will modify the *current local modelling transformation* according to the composition type. Then the modified local modelling transformation is composed with the current global modelling transformation to create a new composite modelling transformation. The composite modelling transformation then maps the modelling coordinates (MC) used to define individual output primitives to a unified world coordinate (WC) space.

If the current edit mode is INSERT, the SET LOCAL TRANSFORMATION element is inserted into the open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, the SET LOCAL TRANSFORMATION element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

**C Input Parameters**

*xform* The $3 \times 3$ homogeneous *transformation matrix*, of type:

```
typedef  Pfloat  Pmatrix[3][3];
```

*compose_type*

The *composition type* is an enumerated value, one of:

- PTYPE_PRECONCAT *Preconcatenate*
- PTYPE_POSTCONCAT *Postconcatenate*
- PTYPE_REPLACE *Replace*

**Execution**

When traversal of a structure begins, the initial current local modelling transformation (L) and the current global modelling transformation (G) are both the 3D, $4 \times 4$ identity matrix. The composite modelling transformation (C) within a structure traversal is formed by the matrix multiplication of the current local modelling transformation (L) and the current global modelling transformation (G) as follows:

$$C \leftarrow G \times L$$

PHIGS assumes representation of points as column vectors. Hence, the order of composition in $C \leftarrow G \times L$ (and throughout) is ‘‘post-concatenation’’ or ‘‘post-multiply.’’

When a new structure is traversed, one step in the invocation of the referenced structure is to save the current modelling transformations (L, G, and C). The child structure inherits the parent’s composite modelling transformation (C) as its current global modelling transformation (G), and an identity matrix as its initial current local modelling transformation (L). The parent and child have equal composite modelling transformations (C), because L is the identity. After traversal of the child structure network, the saved transformations are restored so that the parent is unaffected by the execution of a child.

When the SET LOCAL TRANSFORMATION element is traversed, first the element's *transformation matrix* is expanded to the 3D equivalent *transformation matrix* (T). Then the *compose type* specifies the role of the current local modelling transformation (L) in composing the new value for the current local modelling transformation (L´), which is then combined with the current global modelling transformation (G) to calculate the new composite modelling transformation (C).

*Preconcatenate*

The *transformation matrix* (T) is multiplied by the current local modelling transformation (L):

$L' \leftarrow L \times T$

$C \leftarrow G \times L'$

*Postconcatenate*

The current local modelling transformation (L) is multiplied by the *transformation matrix* (T):

$L' \leftarrow T \times L$

$C \leftarrow G \times L'$

*Replace* The *transformation matrix* (T) replaces the value of current local modelling transformation (L).

$L' \leftarrow T$

$C \leftarrow G \times L'$

The current composite modelling transformation maps the modelling coordinates, used to define individual output primitives, to world coordinates. Mapping the primitives to the world coordinate space establishes the relation between different objects of the image by redefining the parts in terms of a unified coordinate space. This allows the application to define different parts of the image in different local modelling coordinates convenient to the objects being defined, and then to apply transformations that will map the local coordinate systems of each part to a single world coordinate (WC) space. Finally, the viewing mechanism maps WC to device coordinates on the workstation's physical display surface.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

SET GLOBAL TRANSFORMATION (3P)
SET VIEW REPRESENTATION (3P)
BUILD TRANSFORMATION MATRIX (3P)
TRANSFORM POINT (3P)
SET LOCAL TRANSFORMATION 3 (3P)

**NAME**

SET LOCAL TRANSFORMATION 3 – create a structure element containing a 3D local modelling transformation matrix

**SYNOPSIS**

**C Syntax**

```
void
pset_local_tran3 ( xform, compose_type )
Pmatrix3        xform;          transformation matrix
Pcompose_type   compose_type;   composition type
```

**Required PHIGS Operating States**

(PHOP, ∗, STOP, ∗)

**DESCRIPTION**

**Purpose**

SET LOCAL TRANSFORMATION 3 creates a structure element containing a 3D local modelling transformation matrix and a composition type, allowing various objects in the image to be defined in various convenient modelling coordinate systems.

During traversal the element will modify the *current local modelling transformation* according to the composition type. Then the modified local modelling transformation is composed with the current global modelling transformation to create a new composite modelling transformation. The composite modelling transformation then maps the modelling coordinates (MC) used to define individual output primitives to a unified world coordinate (WC) space.

If the current edit mode is INSERT, the SET LOCAL TRANSFORMATION 3 element is inserted into the open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, the SET LOCAL TRANSFORMATION 3 element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

**C Input Parameters**

*xform* The $4 \times 4$ homogeneous *transformation matrix*, of type:

```
typedef Pfloat Pmatrix3[4][4];
```

*compose_type*

The *composition type* is an enumerated value, one of:

- PTYPE_PRECONCAT ‘‘Preconcatenate’’
- PTYPE_POSTCONCAT ‘‘Postconcatenate’’
- PTYPE_REPLACE ‘‘Replace’’

**Execution**

When traversal of a structure begins, the initial current local modelling transformation (L) and the current global modelling transformation (G) are both the 3D, $4 \times 4$ identity matrix. The composite modelling transformation (C) within a structure traversal is formed by the matrix multiplication of the current local modelling transformation (L) and the current global modelling transformation (G) as follows:

$$C \leftarrow G \times L$$

PHIGS assumes representation of points as column vectors. Hence, the order of composition in $C \leftarrow G \times L$ (and throughout) is ‘‘post-concatenation’’ or ‘‘post-multiply.’’

When a new structure is traversed, one step in the invocation of the referenced structure is to save the current modelling transformations (L, G, and C). The child structure inherits the parent’s composite modelling transformation (C) as its current global modelling transformation (G), and an identity matrix as its initial current local modelling transformation (L). The parent and child have equal composite modelling transformations (C), because L is the identity. After traversal of the child structure network, the saved transformations are restored so that the parent is unaffected by the execution of a child.

When the SET LOCAL TRANSFORMATION 3 element is traversed, the *compose type* specifies the role of the current local modelling transformation (L) in composing the new value for the current local modelling transformation (L´), which is then combined with the current global modelling transformation (G) to calculate the new composite modelling transformation (C).

''Preconcatenate''

The *transformation matrix* (T) is multiplied by the current local modelling transformation (L):

$L' \leftarrow L \times T$

$C \leftarrow G \times L'$

''Postconcatenate''

The current local modelling transformation (L) is multiplied by the *transformation matrix* (T):

$L' \leftarrow T \times L$

$C \leftarrow G \times L'$

''Replace''

The *transformation matrix* (T) replaces the value of current local modelling transformation (L).

$L' \leftarrow T$

$C \leftarrow G \times L'$

The current composite modelling transformation maps the modelling coordinates, used to define individual output primitives, to world coordinates. Mapping the primitives to the world coordinate space establishes the relation between different objects of the image by redefining the parts in terms of a unified coordinate space. This allows the application to define different parts of the image in different local modelling coordinates convenient to the objects being defined, and then to apply transformations that will map the local coordinate systems of each part to a single world coordinate (WC) space. Finally, the viewing mechanism maps WC to device coordinates on the workstation's physical display surface.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

SET GLOBAL TRANSFORMATION 3 (3P)
SET VIEW REPRESENTATION 3 (3P)
BUILD TRANSFORMATION MATRIX 3 (3P)
TRANSFORM POINT 3 (3P)
SET LOCAL TRANSFORMATION (3P)

**NAME**

SET LOCATOR MODE – set locator device operating mode and echoing state

**SYNOPSIS**

**C Syntax**

```
void
pset_loc_mode ( ws, dev, mode, echo )
Pint            ws;         workstation identifier
Pint            dev;        locator device number
Pop_mode        mode;       operating mode
Pecho_switch    echo;       echo switch
```

**Required PHIGS Operating States**

(PHOP, WSOP, ∗, ∗)

**DESCRIPTION**

**Purpose**

SET LOCATOR MODE sets the operating mode and echo switch for a specified locator device on a specified workstation.

**C Input Parameters**

*ws* The workstation identifier of the workstation associated with the device.

*dev* The device number of the locator device to be set. See the *Available Devices* section in INITIALIZE LOCATOR 3 for a description of the available devices.

*mode* Specifies the operating mode for the specified locator device. Pop_mode is an enumeration defined in phigs.h as follows:

```
typedef enum {
        POP_REQ,
        POP_SAMPLE,
        POP_EVENT
} Pop_mode;
```

*echo* The echo switch value for the specified locator device. Pecho_switch is an enumeration defined in phigs.h as follows:

```
typedef enum {
        PSWITCH_NO_ECHO,
        PSWITCH_ECHO
} Pecho_switch;
```

**Execution**

SET LOCATOR MODE sets the operating mode and echo flag of the specified device. The default operating mode is POP_REQ. The default echo switch is PSWITCH_ECHO.

The operating mode controls how the input from the device is obtained:

- If the operating mode is REQUEST, REQUEST LOCATOR {3} may be used to prompt for and receive input from a locator input device. REQUEST LOCATOR {3} will not return until the operator enters input to the device.
- If the operating mode is SAMPLE, SAMPLE LOCATOR {3} may be used to return the current input value of the device without waiting for operator input.
- If the operating mode is EVENT, the input values generated by the operator are added to the PHIGS input event queue. The functions AWAIT EVENT and GET LOCATOR {3} may then be used to retrieve these events from the input queue.

The echo switch may be used by some devices to control whether or not values selected by the operator are echoed to the device.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

**SEE ALSO**

REQUEST LOCATOR 3 (3P)
SAMPLE LOCATOR 3 (3P)
GET LOCATOR 3 (3P)
INQUIRE LOCATOR DEVICE STATE 3 (3P)
INITIALIZE LOCATOR 3 (3P)

**NAME**

SET MARKER SIZE SCALE FACTOR – create structure element to set the current marker size scale factor attribute

**SYNOPSIS**

**C Syntax**

```
void
pset_marker_size ( size )
Pfloat          size;          marker size scale factor
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET MARKER SIZE SCALE FACTOR creates a structure element containing a value for the current marker size scale factor attribute.

When the current marker size scale factor Aspect Source Flag (ASF) is set to INDIVIDUAL, the current value of this attribute defines the marker size scale factor to be applied to the output primitives:

- POLYMARKER
- POLYMARKER 3

**C Input Parameter**

*size* A real value specifying the marker size scale factor.

**Execution**

If the current edit mode is INSERT, a SET MARKER SIZE SCALE FACTOR element is inserted after the element pointed to by the current element pointer. If the edit mode is REPLACE, the new SET MARKER SIZE SCALE FACTOR element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

When the SET MARKER SIZE SCALE FACTOR element is traversed, the current marker size scale factor entry in the traversal state list set to marker size scale factor.

When the current marker size scale factor ASF is INDIVIDUAL, POLYMARKER and POLYMARKER 3 output primitives that follow in the structure network are drawn using the marker size scale factor specified. This scaling factor is multiplied by the nominal marker size defined in the workstation description table, and the nearest available size on the workstation is used.

When the current marker size scale factor ASF is BUNDLED, the effective marker size scale factor is the marker size scale factor attribute in the polymarker representation selected by the current polymarker index. In this case, the marker size scale factor set with SET MARKER SIZE SCALE FACTOR has no effect.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

SET INDIVIDUAL ASF (3P)
SET POLYMARKER REPRESENTATION (3P)

**NAME**

SET MARKER TYPE – create structure element to set the current marker type attribute

**SYNOPSIS**

**C Syntax**

```
void
pset_marker_type ( markertype )
Pint            markertype;          marker type
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET MARKER TYPE creates a structure element containing a value for the current marker type attribute.

When the current marker type Aspect Source Flag (ASF) is set to INDIVIDUAL, the current value of this attribute defines the marker type to be applied to the output primitives:

- POLYMARKER
- POLYMARKER 3

**C Input Parameter**

*markertype*

An integer value specifying a marker type; the following marker types are defined:

| | | |
|---|---|---|
| 1 | PMARKER_DOT | *Point* (·) |
| 2 | PMARKER_PLUS | *Plus sign* (+) |
| 3 | PMARKER_ASTERISK | *Asterisk* (*) |
| 4 | PMARKER_CIRCLE | *Circle* (◯) |
| 5 | PMARKER_CROSS | *X-mark* (×) |

Support for marker types is workstation dependent.

**Execution**

If the current edit mode is INSERT, a SET MARKER TYPE element is inserted into the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, the new SET MARKER TYPE element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

When the SET MARKER TYPE element is traversed, the current marker type entry in the PHIGS traversal state list is set to marker type.

When the current marker type ASF is set to INDIVIDUAL, POLYMARKER and POLYMARKER 3 output primitives which follow in the structure network are then drawn using the marker type specified.

The default marker type is 3, *Asterisk* (*). If the marker type specified is not available on the workstation, the default will be used.

When the current marker type ASF is set to BUNDLED, the effective marker type is the marker type attribute in the polymarker representation selected by the current polymarker index. In this case, the marker type set with SET MARKER TYPE has no effect.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

SET INDIVIDUAL ASF (3P)
SET POLYMARKER REPRESENTATION (3P)
INQUIRE POLYMARKER REPRESENTATION (3P)
POLYMARKER (3P)

POLYMARKER 3 (3P)

**NAME**

SET MODELLING CLIPPING INDICATOR – create structure element to set the current modelling clipping indicator

**SYNOPSIS**

**C Syntax**

```
void
pset_model_clip_ind ( ind )
Pclip_ind          ind;          clipping indicator
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET MODELLING CLIPPING INDICATOR creates a structure element containing a value for the current modelling clipping indicator attribute, which affects all output primitives.

**C Input Parameter**

*ind* The clipping indicator is an enumerated value, from:

| | |
|---|---|
| PIND_NO_CLIP | *Do not clip* |
| PIND_CLIP | *Clip* |

**Execution**

If the current edit mode is INSERT, a SET MODELLING CLIPPING INDICATOR element is inserted into the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, the new element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

During traversal, this attribute enables or disables modelling clipping to the volume controlled by SET MODELLING CLIPPING VOLUME, SET MODELLING CLIPPING VOLUME 3, and RESTORE MODELLING CLIPPING VOLUME elements.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

INQUIRE MODELLING CLIPPING FACILITIES (3P)
SET MODELLING CLIPPING VOLUME (3P)
SET MODELLING CLIPPING VOLUME 3 (3P)
RESTORE MODELLING CLIPPING VOLUME (3P)

**NAME**

SET MODELLING CLIPPING VOLUME – create structure element to set the current modelling clipping volume

**SYNOPSIS**

**C Syntax**

```
void
pset_model_clip_vol ( op, half_spaces )
Pint              op;              operator
Phalf_space_list  *half_spaces;    list of half-spaces
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET MODELLING CLIPPING VOLUME creates a structure element containing the operator and half-spaces specified, and places the element in the current structure. During traversal, this element is used to modify the value of the current modelling clipping volume attribute, which may affect all output primitives.

A SET MODELLING CLIPPING VOLUME element contains two-dimensional half-spaces. Each specified half-space is expanded to a three-dimensional half-space by setting the *z*-coordinate of the point and normal vector to zero. These half-spaces are then transformed by the current composite modelling transformation. These half-spaces are intersected to form a clipping volume. If no half-spaces are provided, then the clipping volume is defined to be all of modelling coordinate space. Clipping volume is combined as specified by OPERATOR with *current modelling clipping volume* to form a new value of *current modelling clipping volume*. This volume is used to clip subsequent output primitives during structure traversal.

**C Input Parameters**

*op* The operator that describes the application of the half-spaces in this element to the current modelling clipping volume. Operator values defined in phigs.h are:

| | |
|---|---|
| PMC_REPLACE | *Replace* |
| PMC_INTERSECT | *Intersect* |

*half_spaces*

The list of half-spaces to store in the element. Phalf_space_list is defined in phigs.h as:

```
typedef struct {
        Pint        num_half_spaces;    /* number of half-spaces */
        Phalf_space *half_spaces;       /* list of half-spaces */
} Phalf_space_list;
```

The number is the number of half-spaces in the array of Phalf_space elements pointed to by half-spaces, as specified by a point, and the normal to the plane containing the point, in the direction of the half-space. Phalf_space is defined in phigs.h as:

```
typedef struct {
        Ppoint  point;    /* point */
        Pvec    norm;     /* normal */
} Phalf_space;
```

Ppoint and Pvec are similar structures (but have different semantics), defined by phigs.h:

```
typedef struct {
        Pfloat    x;    /* x coordinate */
        Pfloat    y;    /* y coordinate */
} Ppoint;
```

```
typedef struct {
        Pfloat    delta_x;    /* x magnitude */
        Pfloat    delta_y;    /* y magnitude */
} Pvec;
```

The 2D point and normal vector are expanded to 3D by using *z* components equal to 0.

**Execution**

If the current edit mode is INSERT, a SET MODELLING CLIPPING VOLUME element is inserted into the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, the new element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

During traversal, this element is used to modify the value of the current modelling clipping volume attribute, which affects all output primitives. This volume is used to clip subsequent output primitives during structure traversal.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

INQUIRE MODELLING CLIPPING FACILITIES (3P)
SET MODELLING CLIPPING INDICATOR (3P)
RESTORE MODELLING CLIPPING VOLUME (3P)
SET MODELLING CLIPPING VOLUME 3 (3P)

**NAME**

SET MODELLING CLIPPING VOLUME 3 – create structure element to set the current 3D modelling clipping volume

**SYNOPSIS**

**C Syntax**

```
void
pset_model_clip_vol3 ( op, half_spaces )
Pint                op;                 operator
Phalf_space_list3   *half_spaces;       list of half-spaces
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET MODELLING CLIPPING VOLUME 3 creates a structure element containing the operator and half-spaces specified, and places the element in the current structure. During traversal, this element is used to modify the value of the current modelling clipping volume attribute, which may affect all output primitives.

A SET MODELLING CLIPPING VOLUME 3 element contains half-spaces in the model space. Each specified half-space is transformed by the current composite modelling transformation. These half-spaces are intersected to form a clipping volume. If no half-spaces are provided, the clipping volume is defined to be all of modelling coordinate space. Clipping volume is combined as specified by OPERATOR with *current modelling clipping volume* to form a new value of *current modelling clipping volume*. This volume is used to clip subsequent output primitives during structure traversal.

**C Input Parameters**

*op* The operator that describes the application of the half-spaces in this element to the current modelling clipping volume. Operator values defined in phigs.h are:

| | |
|---|---|
| PMC_REPLACE | *Replace* |
| PMC_INTERSECT | *Intersect* |

*half_spaces*

The list of half-spaces to store in the element. Phalf_space_list3 is defined in phigs.h as:

```
typedef struct {
        Pint            num_half_spaces;    /* number of half-spaces */
        Phalf_space3    *half_spaces;       /* list of half-spaces */
} Phalf_space_list3;
```

The number is the number of half-spaces in the array of Phalf_space3 elements pointed to by half-spaces, as specified by a point, and the normal to the plane containing the point, in the direction of the half-space. Phalf_space3 is defined in phigs.h as:

```
typedef struct {
        Ppoint3  point;    /* point */
        Pvec3    norm;     /* normal */
} Phalf_space3;
```

Ppoint3 and Pvec3 are similar structures (but have different semantics), defined by phigs.h:

```
typedef struct {
        Pfloat    x;    /* x coordinate */
        Pfloat    y;    /* y coordinate */
        Pfloat    z;    /* z coordinate */
} Ppoint3;
```

```
typedef struct {
        Pfloat    delta_x;    /* x magnitude */
        Pfloat    delta_y;    /* y magnitude */
        Pfloat    delta_z;    /* z magnitude */
} Pvec3;
```

**Execution**

If the current edit mode is INSERT, a SET MODELLING CLIPPING VOLUME 3 element is inserted into the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, the new element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

During traversal, this element is used to modify the value of the current modelling clipping volume attribute, which affects all output primitives. This volume is used to clip subsequent output primitives during structure traversal.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

INQUIRE MODELLING CLIPPING FACILITIES (3P)
SET MODELLING CLIPPING INDICATOR (3P)
RESTORE MODELLING CLIPPING VOLUME (3P)
SET MODELLING CLIPPING VOLUME (3P)

**NAME**

SET OF FILL AREA SET 3 WITH DATA – create a 3D set of fill area set 3 with data structure element that includes colour and shading data

**SYNOPSIS**

**C Syntax**

**void**
**pset_of_fill_area_set3_data ( fflag, eflag, vflag, colour_model, num_sets, fdata, edata, vlist, vdata )**

| | | |
|---|---|---|
| **Pint** | **fflag;** | *data per facet flag* |
| **Pint** | **eflag;** | *data per edge flag* |
| **Pint** | **vflag;** | *data per vertex flag* |
| **Pint** | **colour_model;** | *colour model* |
| **Pint** | **num_sets;** | *number of facets (fill area sets)* |
| **Pfacet_data_arr3** | ***fdata;** | *array of L({COLRV}{,NORM}) facet data* |
| **Pedge_data_list_list** | ***edata;** | *array of L(L(E)) edge data* |
| **Pint_list_list** | ***vlist;** | *array of L(L(I)) vertex indices* |
| **Pfacet_vdata_list3** | ***vdata;** | *array of L(P3,{COLRV}{,NORM}) facet vertex data* |

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET OF FILL AREA SET 3 WITH DATA creates a set of fill area set 3 primitive with colour and shading data.

It generates a group of fill area sets (or facets) whose vertices are selected by indexing into a single list of vertices. If the facets share common vertices, the primitive can require less data than the corresponding collection of FILL AREA SET 3 WITH DATA primitives.

**C Input Parameters**

All of the following data types are predefined in phigs.h.

*fflag* The data per facet flag specifies the available data for each facet in the set of fill area set 3.

| | | |
|---|---|---|
| 0 | PFACET_NONE | *No Facet Data Specified* |
| 1 | PFACET_COLOUR | *Facet Colours Specified* |
| 2 | PFACET_NORMAL | *Facet Normals Specified* |
| 3 | PFACET_COLOUR_NORMAL | *Facet Normals and Colours Specified* |

*eflag* The data per edge flag specifies the available information for each edge of each facet in the set of fill area set 3.

| | | |
|---|---|---|
| 0 | PEDGE_NONE | *No Edge Data Specified* |
| 1 | PEDGE_VISIBILITY | *Edge Visibility Flags Specified* |

*vflag* The data per vertex flag specifies the available data for each vertex of the primitive.

| | | |
|---|---|---|
| 0 | PVERT_COORD | *Coordinates Specified* |
| 1 | PVERT_COORD_COLOUR | *Coordinates and Vertex Colours Specified* |
| 2 | PVERT_COORD_NORMAL | *Coordinates and Vertex Normals Specified* |
| 3 | PVERT_COORD_COLOUR_NORMAL | *Coordinates, Vertex Colours, and Vertex Normals Specified* |

*colour_model*

The colour model for specified vertex colours.

| | | |
|---|---|---|
| 0 | PINDIRECT | *Colour Index Specified* |
| 1 | PMODEL_RGB | *Red, Green, and Blue* |

2 PMODEL_CIELUV *CIE Colour Model*
3 PMODEL_HSV *Hue, Saturation, and Value*
4 PMODEL_HLS *Hue, Lightness, and Saturation*

*num_sets*
The number of facets. Each facet is one fill area set.

*fdata* A pointer to a Pfacet_data_arr3 structure that specifies the colour and/or norm information for each facet. Pfacet_data_arr3 is defined as:

```
typedef union {
        Pcoval      *colrs;     /* array (num_sets) of colours */
        Pvec3       *norms;     /* array (num_sets) of unit normals */
        Pconorm3    *conorms;   /* array (num_sets) of colours and normals */
        /* implementation-dependent types can go here */
} Pfacet_data_arr3;
```

Pcoval is defined as:

```
typedef union {
        Pint        ind;    /* index in workstation colour bundle table */
        Pcolr_rep   direct; /* direct colour components */
} Pcoval;
```

When colour model is PINDIRECT, index is used; otherwise direct is used.

Pcolr_rep is defined in phigs.h as follows:

```
typedef union {
        Prgb        rgb;        /* Red Green Blue colour specification */
        Pcieluv     cieluv;     /* CIE L*U*V* colour specification */
        Phls        hls;        /* Hue Lightness Saturation colour specification */
        Phsv        hsv;        /* Hue Saturation Value colour specification */
        Pdata       unsupp;     /* Colour in unsupported colour model */
} Pcolr_rep
```

Prgb is defined in phigs.h as follows:

```
typedef struct {
        Pfloat      red;        /* red, hue, etc */
        Pfloat      green;      /* green, saturation, lightness, etc */
        Pfloat      blue;       /* blue, value, saturation, etc */
} Prgb;
```

Pcieluv is defined in phigs.h as follows:

```
typedef struct {
        Pfloat      cieluv_x;       /* x coefficient */
        Pfloat      cieluv_y;       /* y coefficient */
        Pfloat      cieluv_y_lum;   /* y luminance */
} Pcieluv;
```

Phls is defined in phigs.h as follows:

```
typedef struct {
        Pfloat      hue;        /* hue */
```

```
        Pfloat          lightness;      /* lightness */
        Pfloat          satur;          /* saturation */
} Phls;
```

Phsv is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          hue;            /* hue */
        Pfloat          satur;          /* saturation */
        Pfloat          value;          /* value */
} Phsv;
```

Pdata is defined in phigs.h as follows:

```
typedef struct {
        size_t          size;           /* size of data */
        char            *data           /* pointer to data */
} Pdata;
```

Pconorm3 is defined as:

```
typedef struct {
        Pcoval  colr;   /* colour */
        Pvec3   norm;   /* unit normal */
} Pconorm3;
```

Pcoval is defined above. Pvec3 is defined as:

```
typedef struct {
        Pfloat  delta_x;    /* x magnitude */
        Pfloat  delta_y;    /* y magnitude */
        Pfloat  delta_z;    /* z magnitude */
} Pvec3;
```

*edata* A pointer to an array (num_sets) of Pedge_data_list_list structures. Each Pedge_data_list_list structure specifies the edge data list for the corresponding facet (fill area set). Each Pedge_data_list structure contains a list of edge flags for a single bound in a facet. The first element of the Pedge_data_arr edge data array contains the information pertaining to the edge that falls between the first and second vertex index. The number of edges will be the same as the number of vertices in the vlist data structure. Pedge_data_list_list (edge data for a single fill area set) is defined as:

```
typedef struct {
        Pint            num_lists;  /* number of Pedge_data_lists (bounds) in list */
        Pedge_data_list *edgelist;  /* list of edge data list */
} Pedge_data_list_list;
```

Pedge_data_list (list of edges for one bound) is defined as:

```
typedef struct {
        Pint            num_edges;  /* number of edges */
        Pedge_data_arr  edgedata;   /* edge data */
} Pedge_data_list;
```

Pedge_data_arr is defined as:

```
typedef union {
```

```
        Pedge_flag    *edges;   /* array of edge flags */
        /* implementation-dependent data */
} Pedge_data_arr;
```

Pedge_flag is defined as:

```
typedef enum {
        PEDGE_OFF,
        PEDGE_ON
} Pedge_flag;
```

*vlist* A pointer to an array (num_sets) of Pint_list_list structures. Each Pint_list_list structure specifies the integer list for the corresponding facet (fill area set). Each Pint_list structure contains the indices into the vertex data that specify a single bound of each facet. Pint_list_list (bounds for a single fill area set) is defined as:

```
typedef struct {
        Pint        num_lists;    /* number of Pint_lists in list */
        Pint_list   *lists;       /* list of integer lists */
} Pint_list_list;
```

Pint_list (vertex indices for all vertices in a single bound) is defined as:

```
typedef struct {
        Pint    num_ints;    /* number of Pints in list */
        Pint    *ints;       /* list of integers */
} Pint_list;
```

*vdata* A pointer to a Pfacet_vdata_list3 structure that specifies the set of fill area set 3 vertices and optionally associated colour and norm information. Pfacet_vdata_list3 is defined as:

```
typedef struct {
        Pint                  num_vertices;    /* number of vertices */
        Pfacet_vdata_arr3     vertex_data;     /* facet vertex data */
} Pfacet_vdata_list3;
```

Pfacet_vdata_arr3 is defined as:

```
typedef union {
        Ppoint3         *points;        /* array of points */
        Pptco3          *ptcolrs;       /* array of points and colours */
        Pptnorm3        *ptnorms;       /* array of points and normals */
        Pptconorm3      *ptconorms;     /* array of points, colours, and normals */
        /* implementation-dependent data pointer */
} Pfacet_vdata_arr3;
```

Ppoint3 is defined as:

```
typedef struct {
        Pfloat    x;    /* x coordinate */
        Pfloat    y;    /* y coordinate */
        Pfloat    z;    /* z coordinate */
} Ppoint3;
```

Pptco3 is defined as:

```
typedef struct {
        Ppoint3    point;      /* point coordinates */
        Pcoval     colr;       /* colour */
} Pptco3;
```

Ppoint3 and Pcoval are defined above.

Pptnorm3 is defined as:

```
typedef struct {
        Ppoint3    point;    /* point coordinates */
        Pvec3      norm;     /* unit normal */
} Pptnorm3;
```

Ppoint3 and Pvec3 are defined above.

Pptconorm3 is defined as:

```
typedef struct {
        Ppoint3    point;    /* point coordinates */
        Pcoval     colr;     /* colour */
        Pvec3      norm;     /* unit normal */
} Pptconorm3;
```

Ppoint3, Pcoval, and Pvec3 are defined above.

**Execution**

Depending on the edit mode, a SET OF FILL AREA SET 3 WITH DATA element is inserted into the open structure after the element pointer, or it replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the newly created SET OF FILL AREA SET 3 WITH DATA element.

When the structure is traversed, SET OF FILL AREA SET 3 WITH DATA generates a group of implicitly closed facets. Each facet is rendered as a fill area set. The collection of facets need not be coplanar and need not necessarily form a closed surface. Vertices that define the facet contours are selected through an array of indices. Each array references specific coordinates from the vertex list. The odd winding rule defines the facet interior. The application should insure that each facet's points are coplanar. Non-coplanar facet display is device-dependent. Facets containing fewer than three vertices can be created, but are ignored during structure traversal.

SET OF FILL AREA SET 3 WITH DATA uses the same attributes as the FILL AREA SET 3 WITH DATA structure element.

Vertex and facet colours and normals are used to calculate the set colour and shading. The SET OF FILL AREA SET 3 WITH DATA element can specify the colour and normal vector of each facet, as well as colour and normal vectors for each vertex. Note that unexpected lighting effects will result if the supplied normals are not of unit length.

Each edge has an optional visibility flag to determine if that edge is rendered. There must be as many lists of edge-visibility flags as there are lists of vertex indices, and corresponding lists must be the same length. The first edge-visibility array entry controls edge visibility between the first two corresponding contour vertices. The last edge-visibility array entry controls edge visibility between the last and the first vertices of the corresponding contour. If edge visibility flags are specified, an edge is displayed only when the EDGE FLAG attribute and the edge visibility flag are both ON. If edge visibility flags are not specified, no edges are displayed.

**Attributes Applied**

The attributes listed below are used to display the SET OF FILL AREA SET 3 WITH DATA primitive when the structure is traversed. The Aspect Source Flags (ASFs) tell where to access the output display attributes. These attributes can come directly from the traversal state list, or they can be accessed indirectly, using the appropriate index in the traversal state list and the corresponding bundled representation in the workstation state list.

| | |
|---|---|
| interior colour | interior colour index ASF |
| back interior colour | back interior colour ASF |
| interior style | interior style ASF |
| back interior style | back interior style ASF |
| interior style index | interior style index ASF |
| back interior style index | back interior style index ASF |
| interior shading method | interior shading method ASF |
| back interior shading method | back interior shading method ASF |
| interior reflectance equation | interior reflectance equation ASF |
| back interior reflectance equation | back interior reflectance equation ASF |
| area properties | area properties ASF |
| back area properties | back area properties ASF |
| interior index | |
| edge colour | edge colour ASF |
| edge flag | edge flag ASF |
| edgetype | edgetype ASF |
| edgewidth scale factor | edgewidth scale factor ASF |
| edge index | |
| face distinguishing mode | |
| face culling mode | |
| depth cue index | |
| light source state | |
| name set | |

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

504 Ignoring function, one or more of the vertex indices is out of range

513 Ignoring function, inconsistent edge flag specification.

**SEE ALSO**

FILL AREA SET 3 WITH DATA (3P+)
INTRO PHIGS PLUS (3P+)

**NAME**

SET PARAMETRIC SURFACE CHARACTERISTICS – create a structure element to set the current parametric surface characteristics

**SYNOPSIS**

**C Syntax**

```
void
pset_para_surf_characs( psc_type, data )
Pint                psc_type;          type
Ppara_surf_characs  *data;             OUT data record
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

SET PARAMETRIC SURFACE CHARACTERISTICS sets current parametric surface characteristics to be used when displaying subsequent parametric surface output primitives.

**C Input Parameters**

*psc_type*

The parametric surface characteristic types.

| | | |
|---|---|---|
| 1 | PSC_NONE | *No Surface Characteristics Specified* |
| 2 | PSC_WS_DEP | *Workstation Dependent* |
| 3 | PSC_ISOPARAMETRIC_CURVES | *Isoparametric Curves Specified* |
| 4 | PSC_LEVEL_CURVES_MC | *Level Curves Specified in Modelling Coordinates* |
| 5 | PSC_LEVEL_CURVES_WC | *Level Curves Specified in World Coordinates* |

**C Output Parameters**

*data* The surface characteristics data record. A pointer to a structure Ppara_surf_characs. Ppara_surf_characs is defined in phigs.h as follows:

```
typedef union {
struct {
        Pint                unused;
} psc_1;
struct {
        Pint                unused;
} psc_2;
struct {
        Pcurve_placement    placement;
        Pint                u_count;
        Pint                v_count;
} psc_3;
struct {
        Ppoint3             origin;
        Pvec3               direction;
        Pfloat_list         params;
} psc_4;
struct {
        Ppoint3             origin;
        Pvec3               direction;
        Pfloat_list         params;
} psc_5;
} Ppara_surf_characs;
```

Pcurve_placement is defined in phigs.h as follows:

```
typedef enum {
        PCP_UNIFORM,
        PCP_NON_UNIFORM
} Pcurve_placement;
```

**Execution**

When the structure is traversed, SET PARAMETRIC SURFACE CHARACTERISTICS updates the current parametric surface characteristics entry in the traversal state list. These values are used when displaying subsequent parametric surface output primitives. When an element of this type is interpreted, if the specified parametric surface characteristic type is not available on a workstation, or the data record are invalid, type 1 is used and data record contents are workstation-dependent.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

138 Ignoring function, one or more of the fields in the specified data record is inconsistent with the specified type

**SEE ALSO**

NON-UNIFORM B-SPLINE SURFACE (3P+)

**NAME**

SET PATTERN REFERENCE POINT – create structure element to set the current pattern reference point

**SYNOPSIS**

**C Syntax**

```
void
pset_pat_ref_point ( ref_pt )
Ppoint          *ref_pt;          pattern reference point
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET PATTERN REFERENCE POINT creates a structure element containing a value for the *current pattern reference point* attribute. The structure implicitly resets the *current pattern reference vectors* attribute as well. When the interior style is PATTERN, these attributes apply to the following output primitives:

- FILL AREA
- FILL AREA 3
- FILL AREA SET
- FILL AREA SET 3
- FILL AREA SET 3 WITH DATA
- NON-UNIFORM B-SPLINE SURFACE
- SET OF FILL AREA SET 3 WITH DATA
- TRIANGLE STRIP 3 WITH DATA
- QUADRILATERAL MESH 3 WITH DATA

**Note:** This function places the appropriate data in the Central Structure Store and in PEX-SI archives, but the structure element is currently ignored during traversal.

If the current edit mode is INSERT, a SET PATTERN REFERENCE POINT element is inserted into the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, the new element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

**C Input Parameter**

*ref_pt* The *pattern reference point* is given in Modelling Coordinates (MC), using a 2D shorthand; 0 is used as the *z* coordinate. The Ppoint type is defined in phigs.h as:

```
typedef struct {
        Pfloat      x;          /* x coordinate */
        Pfloat      y;          /* y coordinate */
} Ppoint;
```

**Execution**

When the SET PATTERN REFERENCE POINT element is traversed, the current pattern reference point entry in the traversal state list is set to the *x* and *y* values contained in this element, and the *z* value 0. The current pattern reference vectors are set to (1,0,0) and (0,1,0). When the interior style (either individual or bundled, as selected by the *current interior style ASF*) is set to PATTERN, these attributes are used in conjunction with the current pattern size for the area-defining primitives that follow in the structure network.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

INQUIRE PATTERN FACILITIES (3P)
SET PATTERN SIZE (3P)
SET PATTERN REPRESENTATION (3P)
SET PATTERN REFERENCE POINT AND VECTORS (3P)

**NAME**

SET PATTERN REFERENCE POINT AND VECTORS – create structure element to set the current pattern reference point and vectors

**SYNOPSIS**

**C Syntax**

```
void
pset_pat_ref_point_vecs ( ref_pt, pat_ref_vec )
Ppoint3          *ref_pt;          pattern reference point
Pvec3            pat_ref_vec[2];   direction vectors; 1=X axis of pattern,
                                   2=Y axis of pattern
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET PATTERN REFERENCE POINT AND VECTORS creates a structure element containing a value for the *current pattern reference point* and *current pattern reference vectors* attributes. When the interior style is PATTERN, these attributes apply to the following output primitives:

- FILL AREA
- FILL AREA 3
- FILL AREA SET
- FILL AREA SET 3
- FILL AREA SET 3 WITH DATA
- NON-UNIFORM B-SPLINE SURFACE
- SET OF FILL AREA SET 3 WITH DATA
- TRIANGLE STRIP 3 WITH DATA
- QUADRILATERAL MESH 3 WITH DATA

**Note:** This function places the appropriate data in the Central Structure Store and in PEX-SI archives, but the structure element is currently ignored during traversal.

If the current edit mode is INSERT, a SET PATTERN REFERENCE POINT AND VECTORS element is inserted into the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, the new element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

**C Input Parameters**

*ref_pt* The *pattern reference point*, in Modelling Coordinates (MC). The Ppoint3 structure is defined in phigs.h as:

```
typedef struct {
        Pfloat    x;      /* x coordinate */
        Pfloat    y;      /* y coordinate */
        Pfloat    z;      /* z coordinate */
} Ppoint3;
```

*pat_ref_vec*

A pointer to an array of two elements of type Pvec3. Array element 1 is the X axis of pattern in MC. Array element 2 is the Y axis. The Pvec3 structure is defined in phigs.h as:

```
typedef struct {
        Pfloat    delta_x;      /* x magnitude */
        Pfloat    delta_y;      /* y magnitude */
        Pfloat    delta_z;      /* z magnitude */
} Pvec3;
```

**Execution**

When the SET PATTERN REFERENCE POINT AND VECTORS element is traversed, the current pattern reference point and current pattern reference vectors entries in the traversal state list are set to the values contained in this element. When the interior style (either individual or bundled, as selected by the current interior style

ASF) is set to PATTERN, these attributes are used in conjunction with the current pattern size for the area-defining primitives that follow in the structure network.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

INQUIRE PATTERN FACILITIES (3P)
SET PATTERN REFERENCE POINT (3P)
SET PATTERN REPRESENTATION (3P)
SET PATTERN SIZE (3P)

**NAME**

SET PATTERN REPRESENTATION – set a pattern attribute bundle on a specified workstation

**SYNOPSIS**

**C Syntax**

```
void
pset_pat_rep ( ws, index, rep )
Pint        ws;          workstation identifier
Pint        index;       pattern bundle index
Ppat_rep    *rep;        pattern representation pointer
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

SET PATTERN REPRESENTATION sets a pattern attribute bundle on a workstation.

**Purpose**

SET PATTERN REPRESENTATION sets a pattern attribute bundle on a workstation.

**C Input Parameters**

*ws* The identifier of the workstation.

*index* This is an index to the workstation pattern bundle table.

*rep* A pointer to a Ppat_rep structure. Ppat_rep is defined as:

```
typedef struct {
        Pint_size       dims;           /* pattern's dimensions */
        Pint            *colr_array;    /* array of colours */
} Ppat_rep;
```

Pint_size is defined as:

```
typedef struct {
        Pint            size_x;         /* dimension (number of divisions) along X */
        Pint            size_y;         /* dimension (number of divisions) along Y */
} Pint_size;
```

**Execution**

In the pattern table of the workstation state list, the given index is associated with the specified parameters.

A grid of *DX* by *DY* cells is specified with *DX* being horizontal and *DY* being vertical. The colour is given individually for each cell by the colour index, which is an index into the colour table of the workstation.

If the workstation supports interior style PATTERN, the pattern table in the workstation state list has predefined entries taken from the workstation description table. A number of these entries are predefined for every workstation of category OUTPUT and OUTIN supporting interior style PATTERN. Any table entry, including the predefined entries, may be redefined with this function.

When area-defining primitives are displayed during structure traversal, if the currently selected interior style is PATTERN, the current or back interior style index in the PHIGS state list refers to an entry in the pattern table. The selection is made either via the interior bundle or individually, depending on the corresponding Aspect Source Flags(ASFs). If the current or back interior style index is not present in the pattern table, then interior style index one is used. If interior style index one is not present, the result is workstation dependent and the interior style PATTERN is not supported for this workstation. The setting of the corresponding ASFs determines which aspects in the entry are used.

During structure traversal, if a pattern colour index specified in the pattern colour array is not available on the workstation, then colour index one is used.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (in other words, the workstation category is neither OUTPUT, OUTIN, nor MO)

103 Ignoring function, setting this bundle table entry would exceed the maximum number of entries allowed in the workstation bundle table

112 Ignoring function, the pattern index value is less than one

113 Ignoring function, the colour index value is less than zero

116 Ignoring function, one of the dimensions of pattern colour array is less than one

**SEE ALSO**

INQUIRE LIST OF PATTERN INDICES (3P)
INQUIRE PATTERN FACILITIES (3P)
INQUIRE PATTERN REPRESENTATION (3P)
SET INDIVIDUAL ASF (3P)
SET PATTERN REFERENCE POINT AND VECTORS (3P)
SET PATTERN SIZE (3P)

**NAME**

SET PATTERN REPRESENTATION PLUS – set an extended pattern attribute bundle on a workstation

**SYNOPSIS**

**C Syntax**

```
void
pset_pat_rep_plus ( ws, index, rep )
Pint            ws;             workstation identifier
Pint            index;          pattern bundle index
Ppat_rep_plus   *rep;           pattern representation pointer
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

SET PATTERN REPRESENTATION PLUS sets an extended pattern attribute bundle on a workstation.

**C Input Parameters**

*ws* The identifier of the workstation.

*index* This is an index to the workstation pattern bundle table.

*rep* A pointer to a Ppat_rep_plus structure. Ppat_rep_plus is defined as:

```
typedef struct {
        Pint_size       dims;           /* pattern's dimensions */
        Pint            type;           /* colour model */
        Pcoval          *colr_array;    /* array of colours */
} Ppat_rep_plus;
```

Pint_size is defined as:

```
typedef struct {
        Pint            size_x;         /* dimension (number of divisions) along X */
        Pint            size_y;         /* dimension (number of divisions) along Y */
} Pint_size;
```

Pcoval is defined as:

```
typedef union {
        Pint            ind;            /* index in workstation colour bundle table */
        Pcolr_rep       direct;         /* direct colour components */
}Pcoval;
```

Pcolr_rep is defined as:

```
typedef union {
        Prgb            rgb;            /* Red Green Blue colour specification */
        Pcieluv         cieluv;         /* CIE L*U*V* colour specification */
        Phls            hls;            /* Hue Lightness Saturation colour specification */
        Phsv            hsv;            /* Hue Saturation Value colour specification */
        Pdata           unsupp;         /* Colour in unsupported colour model */
} Pcolr_rep;
```

Prgb is defined in phigs.h as follows:

```
typedef struct {
        Pfloat          red;            /* red, hue, etc */
        Pfloat          green;          /* green, saturation, lightness, etc */
        Pfloat          blue;           /* blue, value, saturation, etc */
```

```
} Prgb;

Pcieluv is defined in phigs.h as follows:
typedef struct {
        Pfloat          cieluv_x;       /* x coefficient */
        Pfloat          cieluv_y;       /* y coefficient */
        Pfloat          cieluv_y_lum;   /* y luminance */
} Pcieluv;

Phls is defined in phigs.h as follows:
typedef struct {
        Pfloat          hue;            /* hue */
        Pfloat          lightness;      /* lightness */
        Pfloat          satur;          /* saturation */
} Phls;

Phsv is defined in phigs.h as follows:
typedef struct {
        Pfloat          hue;            /* hue */
        Pfloat          satur;          /* saturation */
        Pfloat          value;          /* value */
} Phsv;

Pdata is defined in phigs.h as follows:
typedef struct {
        size_t          size;           /* size of data */
        char            *data           /*pointer to data */
} Pdata;
```

**Execution**

In the pattern table of the workstation state list, the given index is associated with the specified parameters.

A grid of *DX* by *DY* cells is specified with *DX* being horizontal and *DY* being vertical. If the colour type is indirect colour, then colour is given individually for each cell by the colour index, which is an index into the colour table of the workstation. Otherwise, the colour is given individually for each cell by a direct colour value in the specified colour model.

If the workstation supports interior style PATTERN, the pattern table in the workstation state list has predefined entries taken from the workstation description table. A number of these entries are predefined for every workstation of category OUTPUT and OUTIN supporting interior style PATTERN. Any table entry, including the predefined entries, may be redefined with this function.

When area-defining primitives are displayed during structure traversal, if the currently selected interior style is PATTERN, the current or back interior style index in the PHIGS state list refers to an entry in the extended pattern table. The selection is made either via the interior bundle or individually, depending on the corresponding Aspect Source Flags(ASFs). If the current or back interior style index is not present in the extended pattern table, then interior style index one is used. If interior style index one is not present, the result is workstation dependent and the interior style PATTERN is not supported for this workstation. The setting of the corresponding ASFs determines which aspects in the entry are used.

During structure traversal, if the colour type of a specified pattern table entry is indirect colour and if a pattern colour index specified in the pattern colour array is not available on the workstation, then colour index one is used.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT, OUTIN, nor MO)

103 Ignoring function, setting this bundle table entry would exceed the maximum number of entries allowed in the workstation bundle table

110 Ignoring function, the specified colour model is not available on the workstation

112 Ignoring function, the pattern index value is less than one

113 Ignoring function, the colour index value is less than zero

116 Ignoring function, one of the dimensions of the pattern colour array is less than one

136 Ignoring function, one of the components of the colour specification is out of range

**SEE ALSO**

SET INDIVIDUAL ASF (3P)
INQUIRE LIST OF PATTERN INDICES (3)
INQUIRE PATTERN FACILITIES (3P)
INQUIRE PATTERN REPRESENTATION PLUS (3P+)
INQUIRE PREDEFINED PATTERN REPRESENTATION PLUS (3P+)
SET PATTERN REPRESENTATION (3P)

**NAME**

SET PATTERN SIZE – create structure element to set the current pattern size

**SYNOPSIS**

**C Syntax**

```
void
pset_pat_size ( size_x, size_y )
Pfloat      size_x;      pattern size x
Pfloat      size_y;      pattern size y
```

**Required PHIGS Operating States**

(PHOP, ∗, STOP, ∗)

**DESCRIPTION**

**Purpose**

SET PATTERN SIZE creates a structure element containing a value for the current pattern width vector and current pattern height vector attributes. When the interior style is PATTERN, these attributes apply to the following output primitives:

- FILL AREA
- FILL AREA 3
- FILL AREA SET
- FILL AREA SET 3
- FILL AREA SET 3 WITH DATA
- NON-UNIFORM B-SPLINE SURFACE
- SET OF FILL AREA SET 3 WITH DATA
- TRIANGLE STRIP 3 WITH DATA
- QUADRILATERAL MESH 3 WITH DATA

**Note:** This function places the appropriate data in the Central Structure Store and in PEX-SI archives, but the structure element is currently ignored during traversal.

If the current edit mode is INSERT when SET PATTERN SIZE is called, the SET PATTERN SIZE element is inserted into the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, the new element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

**C Input Parameter**

*size_x* The *x* coordinate of the pattern size, specified in Modelling Coordinates (MC).

*size_y* The *y* coordinate of the pattern size, specified in Modelling Coordinates (MC).

**Execution**

When the SET PATTERN SIZE element is traversed, the current pattern size entry in the traversal state list is set to the element's *pattern size*.

When the interior style (either individual or bundled, as selected by the current interior style ASF) is set to PATTERN, the attribute is used in conjunction with the current pattern reference point and vectors for the area-defining primitives that follow in the structure network.

**ERRORS**

005 Ignoring function, function requires state (PHOP, ∗, STOP, ∗)

**SEE ALSO**

INQUIRE PATTERN FACILITIES (3P)
SET PATTERN REFERENCE POINT AND VECTORS (3P)
SET PATTERN REPRESENTATION (3P)
SET PATTERN REPRESENTATION PLUS (3P+)

**NAME**

SET PICK FILTER – set a PICK input device's pick filter, which determines which primitives are pickable

**SYNOPSIS**

**C Syntax**

```
void
pset_pick_filter ( ws, dev_num, filter )
Pint        ws;         workstation identifier
Pint        dev_num;    pick device number
Pfilter     *filter;    pick filter
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

SET PICK FILTER sets the *pick filter* of a PICK input device, which is compared to the traversal-time *current name set* of each primitive to determine if the primitive is pickable by that device.

The filter contains an *inclusion set* and an *exclusion set* of names. During traversal, a primitive is pickable by the PICK input device if at least one name in the current name set is in the inclusion set and no name in the 'current name set' is in the exclusion set. This means the exclusion set has precedence over the inclusion set. Each name in the name set, inclusion set, and exclusion set is a small positive integer.

**C Input Parameters**

*ws* The identifier of the workstation.

*dev_num*

The device number of the PICK input device on this workstation whose pick filter is to be set.

*filter* A pointer to a Pfilter structure containing the inclusion set and exclusion set of names. Pfilter is defined in phigs.h as follows:

```
typedef struct {
        Pint_list       incl_set;       /* inclusion set */
        Pint_list       excl_set;       /* exclusion set */
} Pfilter;
```

The Pint_list structure is defined in phigs.h as follows:

```
typedef struct {
        Pint    num_ints;       /* number of Pints in list */
        Pint    *ints;          /* list of integers */
} Pint_list;
```

**Execution**

SET PICK FILTER sets the PICK input device's pick filter, which contains an inclusion set and an exclusion set of names, both empty by default. A primitive is pickable by a workstation's PICK input device if at least one name in the current name set is in the PICK device's pick filter inclusion set and no name in the current name set is in the device's pick filter exclusion set. If the device's pick filter inclusion set is empty, no primitives are pickable by that device.

If the current name set is empty, subsequent primitives are not pickable by any PICK device. When the conceptual traversal of a posted structure network (to determine what primitive is being picked) starts, the current name set is empty. During traversal, the member names specified by the ADD NAMES TO SET element are added to the current name set by the union operation on the sets; REMOVE NAMES FROM SET elements remove names from the current name set.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

| | |
|---|---|
| 054 | Ignoring function, the specified workstation is not open |
| 060 | Ignoring function, specified workstation is not of category OUTIN |
| 250 | Ignoring function, the specified device is not available on the specified workstation |

**SEE ALSO**

ADD NAMES TO SET (3P)
REMOVE NAMES FROM SET (3P)
INITIALIZE PICK (3P)
SET PICK MODE (3P)
REQUEST PICK (3P)
GET PICK (3P)
SAMPLE PICK (3P)
INQUIRE PICK DEVICE STATE (3P)
INQUIRE PICK DEVICE STATE 3 (3P)

**NAME**

SET PICK IDENTIFIER – create structure element to set the current pick identifier attribute

**SYNOPSIS**

**C Syntax**

```
void
pset_pick_id ( pick_id )
Pint            pick_id;            pick identifier
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET PICK IDENTIFIER creates a structure element containing a value for the current pick identifier attribute, which applies to all output primitives which follow in the structure network.

If the current edit mode is INSERT, a SET PICK IDENTIFIER element is inserted into the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, the new SET PICK IDENTIFIER element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

**C Input Parameter**

*pick_id* The *pick identifier* value.

**Execution**

When the conceptual traversal of a posted structure network (to determine what primitive is being picked) starts, the current pick identifier is 0. When the SET PICK IDENTIFIER element is traversed, the current pick identifier is set to the element's *pick identifier* value, and is associated with all output primitives that follow in the structure network.

If a primitive is picked, the pick identifier at each level in the pick path is part of a PICK input device's measure, and can provide the application auxiliary information about the picked primitive.

The *pick identifier* is **not** used to determine which primitives are pickable; ADD NAMES TO SET, REMOVE NAMES FROM SET, and SET PICK FILTER perform this activity.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

INITIALIZE PICK (3P)
SET PICK FILTER (3P)
REQUEST PICK (3P)
GET PICK (3P)
SAMPLE PICK (3P)

**NAME**

SET PICK MODE – set pick device operating mode and echoing state

**SYNOPSIS**

**C Syntax**

**void**
**pset_pick_mode ( ws, dev, mode, echo )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **dev;** | *pick device number* |
| **Pop_mode** | **mode;** | *operating mode* |
| **Pecho_switch** | **echo;** | *echo switch* |

**Required PHIGS Operating States**

(PHOP, WSOP, ∗, ∗)

**DESCRIPTION**

**Purpose**

SET PICK MODE sets the operating mode and echo switch for a specified pick device on a specified workstation.

**C Input Parameters**

*ws* The workstation identifier of the workstation associated with the device.

*dev* The device number of the pick device to be set. See the *Available Devices* section in INITIALIZE PICK 3 for a description of the available devices.

*mode* Specifies the operating mode for the specified pick device. Pop_mode is an enumeration defined in phigs.h as follows:

```
typedef enum {
        POP_REQ,
        POP_SAMPLE,
        POP_EVENT
} Pop_mode;
```

*echo* The echo switch value for the specified pick device. Pecho_switch is an enumeration defined in phigs.h as follows:

```
typedef enum {
        PSWITCH_NO_ECHO,
        PSWITCH_ECHO
} Pecho_switch;
```

**Execution**

SET PICK MODE sets the operating mode and echo flag of the specified device. The default operating mode is POP_REQ. The default echo switch is PSWITCH_ECHO.

The operating mode controls how the input from the device is obtained:

- If the operating mode is REQUEST, REQUEST PICK may be used to prompt for and receive input from a pick input device. REQUEST PICK will not return until the operator enters input to the device.

- If the operating mode is SAMPLE, SAMPLE PICK may be used to return the current input value of the device without waiting for operator input.

- If the operating mode is EVENT, the input values generated by the operator are added to the PHIGS input event queue. The functions AWAIT EVENT and GET PICK may then be used to retrieve these events from the input queue.

The echo switch may be used by some devices to control whether or not values selected by the operator are echoed to the device.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

060 Ignoring function, specified workstation is not of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

**SEE ALSO**

INITIALIZE PICK (3P)
SET PICK FILTER (3P)
SET PICK IDENTIFIER (3P)
REQUEST PICK (3P)
GET PICK (3P)
SAMPLE PICK (3P)
INQUIRE PICK DEVICE STATE (3P)

**NAME**

SET POLYLINE COLOUR – create a structure element to set the polyline colour attribute

**SYNOPSIS**

**C Syntax**

```
void
pset_line_colr ( colour )
Pgcolr          *colour;          polyline colour
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET POLYLINE COLOUR creates a structure element to set the polyline colour attribute.

When the current polyline colour Aspect Source Flag (ASF) is set to INDIVIDUAL, the current value of this attribute applies to the following output primitives:

- POLYLINE
- POLYLINE 3
- POLYLINE SET 3 WITH DATA
- NON-UNIFORM B-SPLINE CURVE
- ANNOTATION TEXT RELATIVE (when ‘‘connected’’ style is in use)
- ANNOTATION TEXT RELATIVE 3 (when ‘‘connected’’ style is in use)
- CELL ARRAY (when minimal simulation is performed)
- CELL ARRAY 3 (when minimal simulation is performed)

**C Input Parameters**

*colour* Polyline colour. Pgcolr is defined as:

```
typedef struct {
        Pint            type;           /* indirect, RGB, CIE, HSV, HLS */
        union {
          Pint          ind;            /* colour table index */
          struct {
                Pfloat          x;      /* red, hue, etc. */
                Pfloat          y;      /* green, saturation, lightness, etc. */
                Pfloat          z;      /* blue, value, saturation, etc. */
          } general;
        } val;
} Pgcolr;
```

**Execution**

Depending on the edit mode, a SET POLYLINE COLOUR element with the value polyline colour in the specified colour model is inserted into the open structure after the element pointer or replaces the element pointed at by the element pointer. The element pointer is then updated to point to this SET POLYLINE COLOUR element.

When the current polyline colour ASF is set to INDIVIDUAL, linear curve output primitives that follow in the structure network are drawn using the current polyline colour.

When the current polyline colour ASF is set to BUNDLED, the polyline colour is taken from the workstation extended representation indicated by the current polyline index. In this case, the polyline colour set with SET POLYLINE COLOUR has no effect.

The standard PHIGS function SET POLYLINE COLOUR INDEX is equivalent to calling to SET POLYLINE COLOUR with type INDIRECT.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

COLOUR (7P)
SET INDIVIDUAL ASF (3P)
INQUIRE POLYLINE REPRESENTATION PLUS (3P+)
SET POLYLINE COLOUR INDEX (3P)
SET POLYLINE REPRESENTATION (3P)

**NAME**

SET POLYLINE COLOUR INDEX – create structure element to set the current polyline colour index attribute

**SYNOPSIS**

**C Syntax**

```
void
pset_line_colr_ind ( colour )
Pint            colour;          polyline colour index
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET POLYLINE COLOUR INDEX creates a structure element containing a value for the current polyline colour index attribute.

When the current polyline colour index Aspect Source Flag (ASF) is set to INDIVIDUAL, this attribute indexes the current colour representation that applies to the following output primitives:

- POLYLINE
- POLYLINE 3
- POLYLINE SET 3 WITH DATA
- NON-UNIFORM B-SPLINE CURVE
- ANNOTATION TEXT RELATIVE (when ‘‘connected’’ style is in use)
- ANNOTATION TEXT RELATIVE 3 (when ‘‘connected’’ style is in use)
- CELL ARRAY (when minimal simulation is performed)
- CELL ARRAY 3 (when minimal simulation is performed)

**C Input Parameter**

*colour* An integer colour index, which selects a colour value from the workstation’s colour table.

**Execution**

If the current edit mode is INSERT, a SET POLYLINE COLOUR INDEX element is inserted into the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, the new SET POLYLINE COLOUR INDEX element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

When the SET POLYLINE COLOUR INDEX element is traversed, the current polyline colour index entry in the traversal state list is set to the colour index.

When the current polyline colour index ASF is set to INDIVIDUAL, the polyline output primitives that follow in the structure network are drawn with the colour representation selected by the current polyline colour index from the workstation’s colour table.

If the colour index specified is not available on the workstation, colour index 1 is used.

When the current polyline colour index ASF is set to BUNDLED, the polyline colour index is taken from the workstation’s representation indicated by the current polyline index. In this case, the colour index set with SET POLYLINE COLOUR INDEX has no effect.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

113 Ignoring function, the colour index value is less than zero

**SEE ALSO**

COLOUR (7P)
SET COLOUR REPRESENTATION (3P)
SET INDIVIDUAL ASF (3P)
SET POLYLINE REPRESENTATION (3P)
SET POLYLINE COLOUR (3P+)

**NAME**

SET POLYLINE INDEX – create structure element containing polyline index attribute

**SYNOPSIS**

**C Syntax**

```
void
pset_line_ind ( index )
Pint            index;          polyline index
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET POLYLINE INDEX creates a structure element containing a polyline index number, which selects an entry from the workstation's polyline bundle table. This attribute applies to the output primitives:

- POLYLINE
- POLYLINE 3
- POLYLINE SET 3 WITH DATA
- NON-UNIFORM B-SPLINE CURVE
- ANNOTATION TEXT RELATIVE (when ''connected'' style is in use)
- ANNOTATION TEXT RELATIVE 3 (when ''connected'' style is in use)
- CELL ARRAY (when minimal simulation is performed)
- CELL ARRAY 3 (when minimal simulation is performed)

**C Input Parameter**

*index* A *polyline index* for a polyline representation on the workstation.

**Execution**

If the current edit mode is INSERT, a SET POLYLINE INDEX element is inserted into the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, the new SET POLYLINE INDEX element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

When the SET POLYLINE INDEX element is traversed, the current polyline index value is set to the *polyline index*, which specifies an entry from the workstation's polyline bundle table. The default *polyline index* is 1, and if the *polyline index* specified is not available on the workstation, 1 will be used.

A polyline representation contains values for the following attributes:

- linetype
- linewidth scale factor
- polyline colour

Line output primitives which follow in the structure network will be drawn using the values from the specified representation for those attributes whose Aspect Source Flag (ASF) is set to BUNDLED. When the ASFs are INDIVIDUAL, the attributes come from the appropriate SET *attribute* elements.

Polyline representations are defined with SET POLYLINE REPRESENTATION.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

100 Ignoring function, the bundle index value is less than one

**SEE ALSO**

SET INDIVIDUAL ASF (3P)
SET POLYLINE REPRESENTATION (3P)
SET POLYLINE REPRESENTATION PLUS (3P+)

**NAME**

SET POLYLINE REPRESENTATION – define a polyline attribute bundle on a workstation

**SYNOPSIS**

**C Syntax**

```
void
pset_line_rep ( ws, index, rep )
Pint            ws;          workstation identifier
Pint            index;       polyline bundle index
Pline_bundle    *rep;        polyline representation pointer
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

SET POLYLINE REPRESENTATION defines a bundle of polyline attributes for a specified entry in the workstation's polyline bundle table. Depending on the ASF for each of the polyline attributes, the bundled attributes may apply to the following primitives:

- POLYLINE
- POLYLINE 3
- POLYLINE SET 3 WITH DATA
- NON-UNIFORM B-SPLINE CURVE
- ANNOTATION TEXT RELATIVE (when ''connected'' style is in use)
- ANNOTATION TEXT RELATIVE 3 (when ''connected'' style is in use)
- CELL ARRAY (when minimal simulation is performed)
- CELL ARRAY 3 (when minimal simulation is performed)

**C Input Parameters**

*ws* The identifier of the workstation for which the polyline representation is being defined.

*index* The *polyline index* of the entry being defined.

*rep* A pointer to a structure containing the attribute values defining the polyline representation, defined as follows:

```
typedef struct {
        Pint    type;       /* line type */
        Pfloat  width;      /* linewidth scale factor */
        Pint    colr_ind;   /* colour index */
} Pline_bundle;
```

*rep->type* is an integer value specifying a linetype; the following line types are defined:

| | | |
|---|---|---|
| 1 | PLINE_SOLID | *Solid* |
| 2 | PLINE_DASH | *Dashed* |
| 3 | PLINE_DOT | *Dotted* |
| 4 | PLINE_DOT_DASH | *Dot-dashed* |

Support for *linetypes* is workstation dependent.

*rep->width* is the *linewidth scale factor*.

*rep->colr_ind* is the *polyline colour* index, which selects a colour value from the workstation's colour table.

**Execution**

When SET POLYLINE REPRESENTATION is called, the polyline index entry in the table of defined polyline representations on the workstation is set to the linetype, linewidth scale factor, and polyline colour index values.

When polyline output primitives are displayed, the polyline representation specified by the current polyline index entry in the traversal state list provides the polyline attributes for which the Aspect Source Flag (ASF) is BUNDLED. For example, when the current linetype ASF is set to BUNDLED, the effective linetype is the linetype attribute in the polyline representation selected by the current polyline index. The current polyline index is set by SET POLYLINE INDEX elements.

The polyline bundle table is numbered from 1.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT, OUTIN, nor MO)

100 Ignoring function, the bundle index value is less than one

103 Ignoring function, setting this bundle table entry would exceed the maximum number of entries allowed in the workstation bundle table

104 Ignoring function, the specified line type is not available on the specified workstation

113 Ignoring function, the colour index value is less than zero

**SEE ALSO**

SET POLYLINE INDEX (3P)
SET INDIVIDUAL ASF (3P)
INQUIRE POLYLINE FACILITIES (3P)
INQUIRE POLYLINE REPRESENTATION (3P)
SET LINETYPE (3P)
SET LINEWIDTH SCALE FACTOR (3P)
SET POLYLINE COLOUR INDEX (3P)
SET POLYLINE COLOUR (3P+)
SET POLYLINE REPRESENTATION PLUS (3P+)

**NAME**

SET POLYLINE REPRESENTATION PLUS – set an extended polyline attribute bundle on a workstation

**SYNOPSIS**

**C Syntax**

```
void
pset_line_rep_plus ( ws, index, rep )
Pint                ws;          workstation identifier
Pint                index;       polyline bundle index
Pline_bundle_plus   *rep;        extended polyline representation pointer
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

SET POLYLINE REPRESENTATION PLUS sets a bundle of extended polyline attributes for a specified entry in a workstation extended polyline bundle table.

**C Input Parameters**

*ws* The identifier of the workstation for which the extended polyline representation is being set.

*index* The index of the entry being set.

*rep* A pointer to a Pline_bundle_plus structure. Pline_bundle_plus is defined as:

```
typedef struct {
        Pint        type;           /* line type */
        Pfloat      width;          /* linewidth scale factor */
        Pgcolr      colr;           /* polyline colour */
        Pint        shad_meth;      /* polyline shading method */
        Pint        approx_type;    /* curve approximation criteria */
        Pfloat      approx_val;     /* curve approximation criteria */
} Pline_bundle_plus;
```

Constants defined for linetype are:

| | | |
|---|---|---|
| 1 | PLINE_SOLID | *Solid* |
| 2 | PLINE_DASH | *Dashed* |
| 3 | PLINE_DOT | *Dotted* |
| 4 | PLINE_DOT_DASH | *Dot-dashed* |

Pgcolr is defined as:

```
typedef struct {
        Pint                type;       /* indirect, RGB, CIE, HSV, HLS */
        union {
            Pint            ind;        /* index in workstation colour bundle table */
            struct {
                Pfloat      x;          /* red, hue, etc. */
                Pfloat      y;          /* green, saturation, lightness, etc. */
                Pfloat      z;          /* blue, value, saturation, etc. */
            } general;
        } val;
} Pgcolr;
```

Constants defined for colour type are:

| | | |
|---|---|---|
| 0 | PINDIRECT | *Indirect* |

| | | |
|---|---|---|
| 1 | PMODEL_RGB | *Red, Green, and Blue* |
| 2 | PMODEL_CIELUV | *CIE* |
| 3 | PMODEL_HSV | *Hue, Saturation, and Value* |
| 4 | PMODEL_HLS | *Hue, Lightness, and Saturation* |

The *ind* member of the val union is used for type PINDIRECT. The *general* member is used for the other types.

Constants defined for polyline shading method are:

| | | |
|---|---|---|
| 1 | PSD_NONE | *No Shading* |
| 2 | PSD_COLOUR | *Colour Interpolation Shading* |
| 3 | PSD_DOT_PRODUCT | |
| 4 | PSD_NORMAL | |

Constants defined for curve approximation type are:

| | |
|---|---|
| 1 | PCURV_WS_DEP |
| 2 | PCURV_CONSTANT_PARAMETRIC_BETWEEN_KNOTS |
| 3 | PCURV_CHORDAL_SIZE_WC |
| 4 | PCURV_CHORDAL_SIZE_NPC |
| 5 | PCURV_CHORDAL_SIZE_DC |
| 6 | PCURV_CHORDAL_DEVIATION_WC |
| 7 | PCURV_CHORDAL_DEVIATION_NPC |
| 8 | PCURV_CHORDAL_DEVIATION_DC |

See SET CURVE APPROXIMATION CRITERIA for a description of the meaning of these constants and their use of the curve approximation value.

**Execution**

SET POLYLINE REPRESENTATION PLUS sets the specified extended polyline bundle table entry in the specified workstation state list. The extended polyline bundle table contains bundled entries of the PHIGS and PHIGS Extension polyline attributes. SET POLYLINE REPRESENTATION PLUS sets the individual entries of this table. During traversal, the attribute values of these bundles are used when the corresponding aspect source flag in the workstation state list is set to BUNDLED.

The PHIGS polyline bundle table is a portion of the extended table. The function SET POLYLINE REPRESENTATION sets the following attributes in the extended table: linetype, linewidth scale factor, and colour index (colour type of *Indirect* is assumed).

During structure traversal the current polyline index refers to an entry in the workstation extended polyline bundle table. Depending on the settings of the aspect source flags, the contents of that bundle table entry are used to determine the attributes associated with output primitives that use the extended polyline bundle table. These primitives are:

POLYLINE
POLYLINE 3
POLYLINE SET 3 WITH DATA
NON-UNIFORM B-SPLINE CURVE
GDP (*Linear Primitives*)
GDP3 (*Linear Primitives*)
CELL ARRAY (*When Performing Minimal Simulation*)
CELL ARRAY 3 (*When Performing Minimal Simulation*)

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT, OUTIN, nor MO)

100 Ignoring function, the bundle index value is less than one

103 Ignoring function, setting this bundle table entry would exceed the maximum number of entries allowed in the workstation bundle table

104 Ignoring function, the specified linetype is not available on the specified workstation

110 Ignoring function, the specified colour model is not available on the workstation

113 Ignoring function, the colour index value is less than zero

122 Ignoring function, the specified polyline shading method is not available on the workstation

136 Ignoring function, one of the components of the colour specification is out of range

127 Ignoring function, the specified approximation criteria type is not available on the specified workstation

128 Ignoring function, the specified parametric surface characteristics type is not available on the specified workstation

138 Ignoring function, one or more of the fields in the specified data record is inconsistent with the specified type

**SEE ALSO**

INQUIRE POLYLINE FACILITIES PLUS (3P+)
SET INDIVIDUAL ASF (3P)
INQUIRE POLYLINE REPRESENTATION PLUS (3P+)
SET CURVE APPROXIMATION CRITERIA (3P+)
SET POLYLINE REPRESENTATION (3P)
SET LINETYPE (3P)
SET LINEWIDTH SCALE FACTOR (3P)
SET POLYLINE COLOUR (3P+)
SET POLYLINE COLOUR INDEX (3P)
INQUIRE POLYLINE FACILITIES (3P)

**NAME**

SET POLYLINE SHADING METHOD – create a structure element that sets the current polyline shading method

**SYNOPSIS**

**C Syntax**

```
void
pset_line_shad_meth ( method )
Pint            method;          shading method
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET POLYLINE SHADING METHOD places a polyline structure element with the value polyline shading method into the open structure. When the current polyline shading method Aspect Source Flag (ASF) is set to INDIVIDUAL, the current value of this attribute applies to the following output primitives:

- POLYLINE
- POLYLINE 3
- POLYLINE SET 3 WITH DATA
- GENERALIZED DRAWING PRIMITIVE *(Polyline Primitives)*
- GENERALIZED DRAWING PRIMITIVE 3 *(Polyline Primitives)*
- NON-UNIFORM B-SPLINE CURVE

**C Input Parameters**

*method* The shading method. The predefined values are:

| | | |
|---|---|---|
| 1 | PSD_NONE | *No Shading* |
| 2 | PSD_COLOUR | *Colour Interpolation Shading* |

**Execution**

Depending on the edit mode, a SET POLYLINE SHADING METHOD element is inserted into the open structure after the element pointer, or replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the newly created SET POLYLINE SHADING METHOD element.

When the structure is traversed, the current polyline shading method entry in the traversal state list is set to the specified value. This value is used with subsequent line output primitives when the current polyline shading method ASF entry in the traversal state list is INDIVIDUAL. During traversal, if the specified shading method is not available on a workstation, method 1 will be used on that workstation.

When the current polyline shading method ASF is set to BUNDLED, the polyline shading method is taken from the workstation extended representation indicated by the current polyline index. In this case, the polyline shading method set with SET POLYLINE SHADING METHOD has no effect.

When the shading method is NONE, each segment of the polyline or curve is drawn in a single colour. If the primitive specifies per-vertex colour, each segment is drawn using the colour associated with the first vertex of the segment.

**ERRORS**

005 Ignoring function, function requires state (PHOP,*,STOP,*)

**SEE ALSO**

POLYLINE SET 3 WITH DATA (3P+)
PHIGS WORKSTATION DESCRIPTION TABLE (7P)
NON-UNIFORM B-SPLINE CURVE (3P+)

**NAME**

SET POLYMARKER COLOUR – create a structure element to set the current polymarker colour

**SYNOPSIS**

**C Syntax**

```
void
pset_marker_colr ( colour )
Pgcolr          *colour;          polymarker colour
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET POLYMARKER COLOUR creates a structure element to set the current polymarker colour.

When the current polymarker colour Aspect Source Flag ASF is set to INDIVIDUAL, the current value of this attribute applies to the following output primitives:

- POLYMARKER
- POLYMARKER 3

**C Input Parameters**

*colour* Polymarker colour. Pgcolr is defined as:

```
typedef struct {
        Pint            type;           /* indirect, RGB, CIE, HSV, HLS */
        union {
          Pint          index;          /* colour table index */
          struct {
                Pfloat          x;      /* red, hue, etc. */
                Pfloat          y;      /* green, saturation, lightness, etc. */
                Pfloat          z;      /* blue, value, saturation, etc. */
          } general;
        } val;
} Pgcolr;
```

**Execution**

Depending on the edit mode, a SET POLYMARKER COLOUR element with the value polymarker colour in the specified colour model is inserted into the open structure after the element pointer or it replaces the element pointed at by the element pointer. The element pointer is then updated to point to this SET POLYMARKER COLOUR element.

When the current polymarker colour ASF is set to INDIVIDUAL, marker output primitives that follow in the structure network are drawn using the current polymarker colour.

When the current polymarker colour ASF is set to BUNDLED, the polymarker colour is taken from the workstation extended representation indicated by the current polymarker index. In this case, the polymarker colour set with SET POLYMARKER COLOUR has no effect.

The standard PHIGS function SET POLYMARKER COLOUR INDEX is equivalent to calling to SET POLYMARKER COLOUR with type INDIRECT.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

COLOUR (7P)
SET INDIVIDUAL ASF (3P)
SET POLYMARKER REPRESENTATION (3P)
SET POLYMARKER REPRESENTATION PLUS(3P+)
SET POLYMARKER COLOUR INDEX (3P)

**NAME**

SET POLYMARKER COLOUR INDEX – create a structure element to set the current polymarker colour index attribute

**SYNOPSIS**

**C Syntax**

```
void
pset_marker_colr_ind ( colour )
Pint            colour;          polymarker colour index
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET POLYMARKER COLOUR INDEX creates a structure element containing a value for the current polymarker colour index attribute.

When the current polymarker colour index Aspect Source Flag (ASF) is set to INDIVIDUAL, this attribute indexes the current colour representation that applies to the following output primitives:

- POLYMARKER
- POLYMARKER 3

**C Input Parameter**

*colour* An integer colour index, which selects a colour value from the workstation's colour table.

**Execution**

If the current edit mode is INSERT, a SET POLYMARKER COLOUR INDEX element is inserted into the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, the new SET POLYMARKER COLOUR INDEX element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

When the SET POLYMARKER COLOUR INDEX element is traversed, the current polymarker colour index entry in the traversal state list is set to the colour index.

When the current polymarker colour index ASF is set to INDIVIDUAL, the polymarker output primitives that follow in the structure network are drawn with the colour representation selected by the current polymarker colour index from the workstation's colour table.

If the colour index specified is not available on the workstation, colour index 1 is used.

When the current polymarker colour index ASF is set to BUNDLED, the polymarker colour index is taken from the workstation's representation indicated by the current polymarker index. In this case, the colour index set with SET POLYMARKER COLOUR INDEX has no effect.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

113 Ignoring function, the colour index value is less than zero

**SEE ALSO**

COLOUR (7P)
SET COLOUR REPRESENTATION (3P)
SET INDIVIDUAL ASF (3P)
SET POLYMARKER REPRESENTATION (3P)
SET POLYMARKER COLOUR (3P+)

**NAME**

SET POLYMARKER INDEX – create a structure element containing a polymarker representation index attribute

**SYNOPSIS**

**C Syntax**

```
void
pset_marker_ind ( index )
Pint            index;          polymarker index
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET POLYMARKER INDEX creates a structure element containing a polymarker index number, which selects an entry from the workstation's polymarker bundle table. This attribute applies to the output primitives:

- POLYMARKER
- POLYMARKER 3

If the current edit mode is INSERT, a SET POLYMARKER INDEX element is inserted into the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, the new SET POLYMARKER INDEX element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

**C Input Parameter**

*index* A *polymarker index* for a polymarker representation on the workstation.

**Execution**

When the SET POLYMARKER INDEX element is traversed, the 'current polymarker index' value is set to the *polymarker index*, which specifies an entry from the workstation's polymarker bundle table. The default *polymarker index* is 1, and if the *polymarker index* specified is not available on the workstation, 1 will be used. When the ASFs are *INDIVIDUAL*, the attributes come from the appropriate SET *attribute* elements.

A polymarker representation contains values for the following attributes:

- marker type
- marker size scale factor
- polymarker colour index

Polymarker output primitives which follow in the structure network will be drawn using the values from the specified representation for those attributes whose Aspect Source Flag (ASF) is set to BUNDLED.

Polymarker representations are defined with SET POLYMARKER REPRESENTATION.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

100 Ignoring function, the bundle index value is less than one

**SEE ALSO**

SET INDIVIDUAL ASF (3P)
SET POLYMARKER REPRESENTATION (3P)
SET POLYMARKER REPRESENTATION PLUS (3P+)

**NAME**

SET POLYMARKER REPRESENTATION – define a *polymarker* attribute bundle on a workstation

**SYNOPSIS**

**C Syntax**

```
void
pset_marker_rep ( ws, index, rep )
Pint            ws;          workstation identifier
Pint            index;       polymarker bundle index
Pmarker_bundle  *rep;        polymarker representation pointer
```

**Required PHIGS Operating States**

(PHOP, WSOP, ∗, ∗)

**DESCRIPTION**

**Purpose**

SET POLYMARKER REPRESENTATION defines a bundle of polymarker attributes for a specified entry in the workstation's polymarker bundle table.

Depending on the ASF for each of the polymarker attributes, the bundled attributes may apply to the following primitives:

- POLYMARKER
- POLYMARKER 3

**C Input Parameters**

*ws* The identifier of the workstation for which the polymarker representation is being defined.

*index* The polymarker index of the entry being defined.

*rep* A pointer to a structure containing the attribute values defining the polymarker representation, defined as follows:

```
typedef struct {
        Pint    type;       /* marker type */
        Pfloat  size;       /* marker size scale factor */
        Pint    colr_ind;   /* colour index */
} Pmarker_bundle;
```

*rep->type* is an integer value specifying a *marker type*; the following marker types are defined:

| | | |
|---|---|---|
| 1 | PMARKER_DOT | *Point* (·) |
| 2 | PMARKER_PLUS | *Plus sign* (+) |
| 3 | PMARKER_ASTERISK | *Asterisk* (∗) |
| 4 | PMARKER_CIRCLE | *Circle* (◯) |
| 5 | PMARKER_CROSS | *X-mark* (×) |

Support for marker types is workstation-dependent. The *Point* (·) marker type is always the smallest dot possible, regardless of the marker size scale factor value applicable.

*rep->size* is the *marker size scale factor*.

*rep->colr_ind* is the *polymarker colour* index, which selects a colour value from the workstation's colour table.

**Execution**

When SET POLYMARKER REPRESENTATION is called, the polymarker index entry in the table of defined polymarker representations on the workstation is set to the marker type, marker size scale factor, and polymarker colour index values.

When polymarker output primitives are displayed, the polymarker representation specified by the current polymarker index entry in the traversal state list provides the polymarker attributes for which the Aspect Source Flag (ASF) is BUNDLED. For example, when the current marker type ASF is set to BUNDLED, the effective marker type is the marker type attribute in the polymarker representation selected by the current polymarker index. The current polymarker index is set by SET POLYMARKER INDEX elements.

The polymarker bundle table is numbered from 1.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT, OUTIN, nor MO)

100 Ignoring function, the bundle index value is less than one

103 Ignoring function, setting this bundle table entry would exceed the maximum number of entries allowed in the workstation bundle table

105 Ignoring function, the specified marker type is not available on the specified workstation

113 Ignoring function, the colour index value is less than zero

**SEE ALSO**

SET POLYMARKER INDEX (3P)
SET INDIVIDUAL ASF (3P)
INQUIRE POLYMARKER FACILITIES (3P)
INQUIRE POLYMARKER REPRESENTATION (3P)
SET MARKERTYPE (3P)
SET MARKER SIZE SCALE FACTOR (3P)
SET POLYMARKER COLOUR INDEX (3P)
SET POLYMARKER COLOUR (3P+)
SET POLYMARKER REPRESENTATION PLUS (3P+)

**NAME**

SET POLYMARKER REPRESENTATION PLUS – use to define an extended polymarker attribute bundle on a workstation

**SYNOPSIS**

**C Syntax**

```
void
pset_marker_rep_plus ( ws, index, rep )
Pint                ws;          workstation identifier
Pint                index;       polymarker bundle index
Pmarker_bundle_plus *rep;        polymarker representation pointer
```

**Required PHIGS Operating States**

(PHOP, WSOP, ∗, ∗)

**DESCRIPTION**

**Purpose**

SET POLYMARKER REPRESENTATION PLUS defines an extended polymarker attribute bundle on a workstation.

**C Input Parameters**

*ws* The identifier of the workstation.

*index* This is an index to the workstation marker bundle table.

*rep* A pointer to a Pmarker_bundle_plus structure. Pmarker_bundle_plus is defined as:

```
typedef struct {
        Pint        type;       /* marker type */
        Pfloat      size;       /* marker size scale factor */
        Pgcolr      colr;       /* marker colour */
} Pmarker_bundle_plus;
```

Pgcolr is defined as:

```
typedef struct {
        Pint                type;       /* indirect, RGB, CIE, HSV, HLS */
        union {
            Pint            ind;        /* index in workstation colour bundle table */
            struct {
                Pfloat      x;          /* red, hue, etc. */
                Pfloat      y;          /* green, saturation, lightness, etc. */
                Pfloat      z;          /* blue, value, saturation, etc. */
            } general;
        } val;
} Pgcolr;
```

*rep->type* is an integer value specifying a *marker type*; the following marker types are defined:

| | | |
|---|---|---|
| 1 | PMARKER_DOT | *Point* (·) |
| 2 | PMARKER_PLUS | *Plus sign* (+) |
| 3 | PMARKER_ASTERISK | *Asterisk* (∗) |
| 4 | PMARKER_CIRCLE | *Circle* (◯) |
| 5 | PMARKER_CROSS | *X-mark* (×) |

Support for marker types is workstation-dependent. The *Point* (·) marker type is always the smallest dot possible, regardless of the marker size scale factor value applicable.

*rep->size* is the *marker size scale factor*.

*rep->colr* is the *polymarker colour*, which selects a colour value from the workstation's colour table.

**Execution**

In the polymarker bundle table of the workstation state list, the given index is associated with the specified parameters.

The marker size scale factor is a scale factor applied to the nominal marker size. During structure traversal, the scale factor is multiplied by the nominal marker size on a workstation. The result is mapped by the workstation to the nearest available marker size.

Polymarker colour is a general colour, either an indirect or a direct colour in a specified colour model. If an indirect colour is specified and the specified colour index is not available during structure traversal, then colour index one is used.

The extended polymarker bundle table in the workstation state list has predefined entries taken from the workstation description table. A number of these entries are predefined for every workstation of category OUTPUT or OUTIN. Any table entry, including the predefined entries, may be redefined with this function.

When polymarkers are displayed during structure traversal, the current polymarker index in the PHIGS traversal state list refers to an entry in the extended polymarker bundle table. If the current polymarker index is not present in the extended polymarker bundle table, then polymarker index one is used. The setting of the corresponding Aspect Source Flags(ASFs) determines which aspects in the entry are used.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT, OUTIN, nor MO)

100 Ignoring function, the bundle index value is less than one

103 Ignoring function, setting this bundle table entry would exceed the maximum number of entries allowed in the workstation bundle table

105 Ignoring function, the specified marker type is not available on the specified workstation

110 Ignoring function, the specified colour model is not available on the workstation

113 Ignoring function, the colour index value is less than zero

136 Ignoring function, one of the components of the colour specification is out of range

**SEE ALSO**

SET INDIVIDUAL ASF (3P)
INQUIRE POLYMARKER REPRESENTATION PLUS (3P+)
SET POLYMARKER REPRESENTATION (3P)
SET MARKER TYPE (3P)
SET MARKER SIZE SCALE FACTOR (3P)
SET POLYMARKER COLOUR INDEX (3P)
SET POLYMARKER COLOUR (3P+)
INQUIRE POLYMARKER FACILITIES (3P)

**NAME**

SET REFLECTANCE EQUATION – create a structure element to set the current interior reflectance equation

**SYNOPSIS**

**C Syntax**

```
void
pset_refl_eqn ( equation )
Pint            equation;          reflectance equation
```

**Required PHIGS Operating States**

(PHOP, ∗, STOP, ∗)

**DESCRIPTION**

**Purpose**

SET REFLECTANCE EQUATION creates a structure element to set the current interior reflectance equation.

When the current interior reflectance equation Aspect Source Flag (ASF) is set to INDIVIDUAL, the current value of this attribute defines the interior reflectance equation to be applied to the following output primitives:

- FILL AREA
- FILL AREA 3
- FILL AREA SET
- FILL AREA SET 3
- FILL AREA SET 3 WITH DATA
- SET OF FILL AREA SET 3 WITH DATA
- NON-UNIFORM B-SPLINE SURFACE
- QUADRILATERAL MESH 3 WITH DATA
- TRIANGLE STRIP 3 WITH DATA

**C Input Parameters**

*equation*

The reflectance equation. The possible values defined in phigs.h are:

| | | |
|---|---|---|
| 1 | PREFL_NONE | *No Reflectance Calculation Performed* |
| 2 | PREFL_AMBIENT | *Use Ambient Term* |
| 3 | PREFL_AMB_DIFF | *Use Ambient and Diffuse Terms* |
| 4 | PREFL_AMB_DIFF_SPEC | *Use Ambient, Diffuse, and Specular Terms* |

**Execution**

Depending on the edit mode, a SET REFLECTANCE EQUATION element is inserted into the open structure after the element pointer or replaces the element pointed at by the element pointer. The element pointer is then updated to point to this SET REFLECTANCE EQUATION element.

At traversal time, if the specified reflectance equation is not defined, then the default is used.

When the current interior reflectance equation ASF is set to INDIVIDUAL, area-defining output primitives which follow in the structure network are filled using the current interior reflectance equation.

When the current interior reflectance equation ASF is set to BUNDLED, the interior reflectance equation is taken from the workstation extended representation indicated by the current interior index. In this case, the interior reflectance equation set with SET REFLECTANCE EQUATION has no effect.

**ERRORS**

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

**SEE ALSO**

SET INDIVIDUAL ASF (3P)
SET INTERIOR REPRESENTATION (3P)
SET FACE DISTINGUISHING MODE (3P+)
SET INDIVIDUAL ASF (3P)
SET INTERIOR REPRESENTATION PLUS (3P+)
SET REFLECTANCE EQUATION (3P+)
FILL AREA (3P)
FILL AREA 3 (3P)
FILL AREA SET (3P)
FILL AREA SET 3 (3P)
FILL AREA SET 3 WITH DATA (3P+)
SET OF FILL AREA SET 3 WITH DATA (3P+)
NON-UNIFORM B-SPLINE SURFACE (3P+)
QUADRILATERAL MESH 3 WITH DATA (3P+)
TRIANGLE STRIP 3 WITH DATA (3P+)

**NAME**

SET REFLECTANCE PROPERTIES – create a structure element to set surface area properties

**SYNOPSIS**

**C Syntax**

```
void
pset_refl_props ( properties )
Prefl_props          *properties;          surface area properties
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET REFLECTANCE PROPERTIES creates a structure element to set surface area properties. When the current area properties Aspect Source Flag (ASF) is set to INDIVIDUAL, the current value of this attribute applies to the following output primitives:

- FILL AREA
- FILL AREA 3
- FILL AREA SET
- FILL AREA SET 3
- FILL AREA SET 3 WITH DATA
- SET OF FILL AREA SET 3 WITH DATA
- QUADRILATERAL MESH 3 WITH DATA
- TRIANGLE STRIP 3 WITH DATA
- NON-UNIFORM B-SPLINE SURFACE

**C Input Parameters**

*properties*

A pointer to a Prefl_props structure that specifies the area properties. Prefl_props is defined in phigs.h as follows:

```
typedef struct {
        Pfloat    ambient_coef;    /* ambient reflectance coefficient */
        Pfloat    diffuse_coef;    /* diffuse reflectance coefficient */
        Pfloat    specular_coef;   /* specular reflectance coefficient */
        Pgcolr    specular_colr;   /* specular colour */
        Pfloat    specular_exp;    /* specular exponent */
        Pfloat    transpar_coef;   /* transparency coefficient */
} Prefl_props;
```

Pgcolr is defined as:

```
typedef struct {
        Pint            type;           /* indirect, RGB, CIE, HSV, HLS */
        union {
          Pint          ind;            /* colour table index */
          struct {
                Pfloat          x;      /* red, hue, etc. */
                Pfloat          y;      /* green, saturation, lightness, etc. */
                Pfloat          z;      /* blue, value, saturation, etc. */
          } general;
```

```
        } val;
} Pgcolr;
```

The color type is one of the following:

| | |
|---|---|
| 0 | PINDIRECT |
| 1 | PMODEL_RGB |
| 2 | PMODEL_CIELUV |
| 3 | PMODEL_HSV |
| 4 | PMODEL_HLS |

**Execution**

Depending upon the edit mode, a SET REFLECTANCE PROPERTIES element is inserted into the open structure after the element pointer or replaces the element pointed at by the element pointer. The element pointer is then updated to point to this SET REFLECTANCE PROPERTIES element.

When the structure is traversed, the current area properties entry in the traversal state list is set to the specified value. This value is used with subsequent filled area output primitives when the current area properties ASF entry in the traversal state list is INDIVIDUAL. During traversal, if the specified area properties are not available on a workstation, method one (*None*) will be used on that workstation.

When the current area properties ASF is set to BUNDLED, the area properties are taken from the workstation extended representation indicated by the current interior index. In this case, the area properties set with SET REFLECTANCE PROPERTIES have no effect.

The area properties supported by a workstation type can be determined with the function INQUIRE INTERIOR FACILITIES PLUS.

**ERRORS**

005 Ignoring function, function requires state (PHOP,*,STOP,*)

**SEE ALSO**

INTRO PHIGS PLUS (3P+)
FILL AREA (3P)
FILL AREA SET (3P)
FILL AREA 3 (3P)
FILL AREA SET 3 (3P)
FILL AREA SET 3 WITH DATA (3P+)
SET OF FILL AREA SET 3 WITH DATA (3P+)
SET INDIVIDUAL ASF (3P)
SET INTERIOR REPRESENTATION (3P)
SET INTERIOR REPRESENTATION PLUS (3P+)
INQUIRE INTERIOR FACILITIES PLUS (3P+)
NON-UNIFORM B-SPLINE SURFACE (3P+)
QUADRILATERAL MESH 3 WITH DATA (3P+)
TRIANGLE STRIP 3 WTIH DATA (3P+)

**NAME**

SET RENDERING COLOUR MODEL – create a structure element to set current rendering colour model

**SYNOPSIS**

**C Syntax**

```
void
pset_rendering_colr_model( colour_model )
Pint            colour_model;        colour model
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET RENDERING COLOUR MODEL specifies the colour model in which lighting calculation and colour interpolation are performed.

**C Input Parameters**

*colour_model*

The defined rendering colour models are:

| | | |
|---|---|---|
| 0 | PRCM_WS_DEP | |
| 1 | PRCM_RGB | *Red, Green, Blue Colour Model* |
| 2 | PRCM_CIE | *Commission Internationale de l'Eclairage Colour Model* |
| 3 | PRCM_HSV | *Hue, Saturation, Value Colour Model* |
| 4 | PRCM_HLS | *Hue, Lightness, Saturation Colour Model* |

**Execution**

Depending on the edit mode, a SET RENDERING COLOUR MODEL element is inserted into the open structure after the element pointer, or replaces the element pointed to by the element pointer. When an element of this type is interpreted, the the value associated with the element. If the specific colour model is not available, colour model 0, PRCM_WS_DEP, is used.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

INQUIRE RENDERING COLOUR MODEL FACILITIES (3P+)

**NAME**

SET STRING MODE – set string device operating mode and echoing state

**SYNOPSIS**

**C Syntax**

**void**
**pset_string_mode ( ws, dev, mode, echo )**

| | | |
|---|---|---|
| **Pint** | **ws;** | *workstation identifier* |
| **Pint** | **dev;** | *string device number* |
| **Pop_mode** | **mode;** | *operating mode* |
| **Pecho_switch** | **echo;** | *echo switch* |

**Required PHIGS Operating States**

(PHOP, WSOP, ∗, ∗)

**DESCRIPTION**

**Purpose**

SET STRING MODE sets the operating mode and echo switch for a specified string device on a specified workstation.

**C Input Parameters**

*ws* The workstation identifier of the workstation associated with the device.

*dev* The device number of the string device to be set. See the *Available Devices* section in INITIALIZE STRING 3 for a description of the available devices.

*mode* Specifies the operating mode for the specified string device. Pop_mode is an enumeration defined in phigs.h as follows:

```
typedef enum {
        POP_REQ,
        POP_SAMPLE,
        POP_EVENT
} Pop_mode;
```

*echo* The echo switch value for the specified string device. Pecho_switch is an enumeration defined in phigs.h as follows:

```
typedef enum {
        PSWITCH_NO_ECHO,
        PSWITCH_ECHO
} Pecho_switch;
```

**Execution**

SET STRING MODE sets the operating mode and echo flag of the specified device. The default operating mode is POP_REQ. The default echo switch is PSWITCH_ECHO.

The operating mode controls how the input from the device is obtained:

- If the operating mode is REQUEST, REQUEST STRING may be used to prompt for and receive input from a string input device. REQUEST STRING will not return until the operator enters input to the device.

- If the operating mode is SAMPLE, SAMPLE STRING may be used to return the current input value of the device without waiting for operator input.

- If the operating mode is EVENT, the input values generated by the operator are added to the PHIGS input event queue. The functions AWAIT EVENT and GET STRING may then be used to retrieve these events from the input queue.

The echo switch may be used by some devices to control whether or not values selected by the operator are echoed to the device.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

**SEE ALSO**

INITIALIZE STRING (3P)
REQUEST STRING (3P)
GET STRING (3P)
SAMPLE STRING (3P)
INQUIRE STRING DEVICE STATE (3P)

**NAME**

SET STROKE MODE – set stroke device operating mode and echoing state

**SYNOPSIS**

**C Syntax**

```
void
pset_stroke_mode ( ws, dev, mode, echo )
Pint            ws;         workstation identifier
Pint            dev;        stroke device number
Pop_mode        mode;       operating mode
Pecho_switch    echo;       echo switch
```

**Required PHIGS Operating States**

(PHOP, WSOP, ∗, ∗)

**DESCRIPTION**

**Purpose**

SET STROKE MODE sets the operating mode and echo switch for a specified stroke device on a specified workstation.

**C Input Parameters**

*ws* The workstation identifier of the workstation associated with the device.

*dev* The device number of the stroke device to be set. See the *Available Devices* section in INITIALIZE STROKE 3 for a description of the available devices.

*mode* Specifies the operating mode for the specified stroke device. Pop_mode is an enumeration defined in phigs.h as follows:

```
typedef enum {
        POP_REQ,
        POP_SAMPLE,
        POP_EVENT
} Pop_mode;
```

*echo* The echo switch value for the specified stroke device. Pecho_switch is an enumeration defined in phigs.h as follows:

```
typedef enum {
        PSWITCH_NO_ECHO,
        PSWITCH_ECHO
} Pecho_switch;
```

**Execution**

SET STROKE MODE sets the operating mode and echo flag of the specified device. The default operating mode is POP_REQ. The default echo switch is PSWITCH_ECHO.

The operating mode controls how the input from the device is obtained:

- If the operating mode is REQUEST, REQUEST STROKE may be used to prompt for and receive input from a stroke input device. REQUEST STROKE will not return until the operator enters input to the device.
- If the operating mode is SAMPLE, SAMPLE STROKE may be used to return the current input value of the device without waiting for operator input.
- If the operating mode is EVENT, the input values generated by the operator are added to the PHIGS input event queue. The functions AWAIT EVENT and GET STROKE may then be used to retrieve these events from the input queue.

The echo switch may be usd by some devices to control whether or not values selected by the operator are echoed to the device.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

**SEE ALSO**

INITIALIZE STROKE (3P)
INITIALIZE STROKE 3 (3P)
REQUEST STROKE 3 (3P)
GET STROKE 3 (3P)
SAMPLE STROKE 3 (3P)
INQUIRE STROKE DEVICE STATE 3 (3P)

**NAME**

SET SURFACE APPROXIMATION CRITERIA – create a structure element to set the surface approximation criteria

**SYNOPSIS**

**C Syntax**

```
void
pset_surf_approx ( type, uvalue, vvalue )
Pint        type;       approximation type
Pfloat      uvalue;     approximation value for u dimension
Pfloat      vvalue;     approximation value for v dimension
```

**Required PHIGS Operating States**

(PHOP, ∗, STOP, ∗)

**DESCRIPTION**

**Purpose**

SET SURFACE APPROXIMATION CRITERIA places into the currently open structure a structure element that specifies the surface approximation type and values. The surface approximation type controls the display precision of non-uniform B-spline surfaces by specifying the method to use to render the surface. The approximation values are used in a type-specific way, as described in the *Execution* section below.

When the current surface approximation criteria Aspect Source Flag (ASF) is set to INDIVIDUAL, the current value of this attribute defines the surface approximation criteria to be applied to the following output primitive:

- NON-UNIFORM B-SPLINE SURFACE

**C Input Parameters**

*type* The surface approximation type. Possible values defined in phigs.h are:

| | |
|---|---|
| 1 | PSURF_WS_DEP |
| 2 | PSURF_CONSTANT_PARAMETRIC_BETWEEN_KNOTS |
| 3 | PSURF_CHORDAL_SIZE_WC |
| 4 | PSURF_CHORDAL_SIZE_NPC |
| 5 | PSURF_CHORDAL_SIZE_DC |
| 6 | PSURF_PLANAR_DEVIATION_WC |
| 7 | PSURF_PLANAR_DEVIATION_NPC |
| 8 | PSURF_PLANAR_DEVIATION_DC |

*uvalue, vvalue*

The values to use when applying the specified approximation method. Valid values depend on the approximation method. The value for the *u* dimension is *uvalue*. The value for the *v* dimension is *vvalue*. See the *Execution* section below for a description of the use of these values by each approximation method.

**Execution**

Depending upon the edit mode, a SET SURFACE APPROXIMATION CRITERIA element is either inserted into the open structure after the element pointer, or replaces the element pointed at by the element pointer. The element pointer is then updated to point to this SET SURFACE APPROXIMATION CRITERIA element.

The approximation methods corresponding to the defined approximation types and the use of the approximation values by each method are:

*Workstation Dependent, Type 1*

Use an implementation dependent approximation method. This method is defined by the server.

*Constant Parametric Subdivision Between Knots, Type 2*

Tessellate using equal parametric increments between the *uknots* and the *vknots*. The integer

portion of the *u* and *v* approximation values are used and interpreted as follows:

≤ 0: Evaluate the surface only at the parameter limits and at the knots that fall within the parameter limits.

> 0: Evaluate the surface at the parameter limits, at the knots that fall within the parameter limits, and at the *uvalue* or *vvalue* number of positions between the knots, but only if the position is within the parameter limits.

*Chordal Size in WC, Type 3*
Tessellate until the maximum length chord measured in World Coordinates (WC) for both the *u* and *v* dimensions is less than the corresponding positive real-number approximation value, *uvalue* and *vvalue*, respectively.

*Chordal Size in NPC, Type 4*
Tessellate until the maximum length chord measured in Normalized Projection Coordinates (NPC) for both the *u* and *v* dimensions is less than the corresponding positive real-number approximation value, *uvalue* and *vvalue*, respectively.

*Chordal Size in DC, Type 5*
Tessellate until the maximum length chord measured in Device Coordinates (DC) for both the *u* and *v* dimensions is less than the corresponding positive real-number approximation value, *uvalue* and *vvalue*, respectively.

*Planar Deviation in WC, Type 6*
Tessellate until the absolute value of the deviation measured in WC between the surface and the generated planar sub-primitives is less than the specified positive real-number approximation value.

*Planar Deviation in NPC, Type 7*
Tessellate until the absolute value of the deviation measured in NPC between the surface and the generated planar sub-primitives is less than the specified positive real-number approximation value.

*Planar Deviation in DC, Type 8*
Tessellate until the absolute value of the deviation measured in DC between the surface and the generated planar sub-primitives is less than the specified positive real-number approximation value.

At traversal time, the default approximation type is used if the specified approximation type is not supported by the workstation. If the approximation values are inconsistent with the specified approximation type, values of 1.0 is used. The approximation types supported by a workstation type can be determined with the function INQUIRE CURVE AND SURFACE FACILITIES, or by consulting the relevant entries in PHIGS WORKSTATION DESCRIPTION TABLE (7P).

When the current surface approximation criteria ASF is set to INDIVIDUAL, area-defining output primitives that follow in the structure network are filled using the current curve approximation criteria style.

When the current surface approximation criteria ASF is set to BUNDLED, the surface approximation criteria is taken from the workstation extended representation indicated by the current interior index. In this case, the surface approximation criteria set with SET SURFACE APPROXIMATION CRITERIA has no effect.

**ERRORS**

005 Ignoring function, function requires state (PHOP, ∗, STOP, ∗)

138 Ignoring function, one or more of the fields in the specified data record are inconsistent with the specified type

**SEE ALSO**

NON-UNIFORM B-SPLINE SURFACE (3P+)
INQUIRE CURVE AND SURFACE FACILITIES (3P+)

**NAME**

SET TEXT ALIGNMENT – create a structure element to set the current text alignment attribute

**SYNOPSIS**

**C Syntax**

```
void
pset_text_align ( text_align )
Ptext_align          *text_align;          text alignment
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET TEXT ALIGNMENT creates a structure element containing a value for the *current text alignment* attribute, which positions the text string in relation to the *text position*. This attribute applies to the output primitives:

- TEXT
- TEXT 3

If the current edit mode is INSERT, a SET TEXT ALIGNMENT element is inserted into the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, then the new SET TEXT ALIGNMENT element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

**C Input Parameters**

*text_align*

A pointer to the *text alignment* structure, defined as follows:

```
typedef struct {
        Phor_text_align     hor;/* horizontal component */
        Pvert_text_align    vert;/* vertical component */
} Ptext_align;
```

*text_align->hor* is the horizontal component. This is an enumerated value, and may be one of:

| | |
|---|---|
| PHOR_NORM | *Normal* |
| PHOR_LEFT | *Left* |
| PHOR_CTR | *Center* |
| PHOR_RIGHT | *Right* |

*text_align->vert* is the vertical component. This is an enumerated value, and may be one of:

| | |
|---|---|
| PVERT_NORM | *Normal* |
| PVERT_TOP | *Top* |
| PVERT_CAP | *Cap* |
| PVERT_HALF | *Half* |
| PVERT_BASE | *Base* |
| PVERT_BOTTOM | *Bottom* |

**Execution**

When the SET TEXT ALIGNMENT element is traversed, the current text alignment entry in the traversal state list is set to text alignment. This attribute is used to position text strings from text output primitives that follow in the structure network, in relation to the text position provided with each text output primitive. The two components of the alignment can be considered individually.

Imagine first rendering the text string using all other text attributes, and then moving the entire text string to place the *text extent parallelogram* that surrounds the character bodies in the correct position in relation to the text position. (The size and shape of the text is entirely specified by the other attributes.) This movement is oriented by the *character up* and *character base* vectors; consider the direction of the character up vector to be vertical, and that of the character base vector to be horizontal.

The horizontal alignment of *Left* or *Right* requires the corresponding side of the parallelogram to pass through the text position. The horizontal alignment of *Center* causes the text position to lie midway between the left and right sides of the parallelogram.

The vertical alignment corresponds to one of the 5 horizontal lines through the definition of a character. (These lines are in the same location for every character in a single font.) The vertical alignment of *Top* or *Bottom* requires the corresponding side of the parallelogram to pass through the text position. The vertical alignment of *Cap* causes the text position to lie on the cap line of the string (when the text path is *Left* or *Right*), or on the cap line of the topmost character in the string (text path is *Up* or *Down*). The vertical alignment of *Base* causes the text position to lie on the base line of the entire string (when the text path is *Left* or *Right*) or on the base line of the bottom character in the string (text path is *Up* or *Down*). The vertical alignment of *Half* causes the text position to lie on the half line of the entire string (when the text path is *Left* or *Right*) or on a line midway between the half lines of the top and bottom characters (text path is *Up* or *Down*).

The Normal value of either text alignment component causes an effect equivalent to one of the other values of the same component. PHIGS defines which other value is used to be the natural alignment for the text path value used:

| *Text Path Value* | *Equivalent (Horizontal, Vertical) Alignment* |
|---|---|
| RIGHT | (*Left, Base*) |
| LEFT | (*Right, Base*) |
| UP | (*Center, Base*) |
| DOWN | (*Center, Top*) |

The default *text alignment* is (*Normal, Normal*); the default text path is *Right*.

**Example**

(*Center, Top*) text alignment might be used with text path *Right* to center a chart's *x-axis* label under the *x-axis*, without calculating the combined size of the characters in the string. (*Right, Center*) might be used with text path *Down* to center the right edge of a chart's *y-axis* label along the *y-axis*. Each character faces normally, but the characters in the string would proceed down the display, to the left of the *y-axis*.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

SET TEXT PATH (3P)
SET ANNOTATION TEXT ALIGNMENT (3P)

**NAME**

SET TEXT COLOUR – create a structure element to set the text colour attribute

**SYNOPSIS**

**C Syntax**

```
void
pset_text_colr ( colour )
Pgcolr          *colour;          text colour
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET TEXT COLOUR creates a structure element to set the text colour attribute.

When the current interior shading method Aspect Source Flag (ASF) is set to INDIVIDUAL, the current value of this attribute applies to the following output primitives:

- TEXT
- TEXT 3
- ANNOTATION TEXT RELATIVE
- ANNOTATION TEXT RELATIVE 3

**C Input Parameters**

*colour* Text colour. Pgcolr is defined as:

```
typedef struct {
        Pint            type;           /* indirect, RGB, CIE, HSV, HLS */
        union {
          Pint          ind;            /* colour table index */
          struct {
                Pfloat          x;      /* red, hue, etc. */
                Pfloat          y;      /* green, saturation, lightness, etc. */
                Pfloat          z;      /* blue, value, saturation, etc. */
          } general;
        } val;
} Pgcolr;
```

**Execution**

Depending on the edit mode, a SET TEXT COLOUR element with the value text colour in the specified colour model is inserted into the open structure after the element pointer, or it replaces the element pointed at by the element pointer. The element pointer is then updated to point to this SET TEXT COLOUR element.

When the current text colour ASF is set to INDIVIDUAL, text output primitives that follow in the structure network are drawn using the current text colour.

When the current text colour ASF is set to BUNDLED, the text colour is taken from the workstation extended representation indicated by the current text index. In this case, the text colour set with SET TEXT COLOUR has no effect.

The standard PHIGS function SET TEXT COLOUR INDEX is equivalent to calling to SET TEXT COLOUR with type INDIRECT.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

COLOUR (7P)
SET INDIVIDUAL ASF (3P)

SET TEXT REPRESENTATION PLUS (3P+)
SET TEXT COLOUR INDEX (3P)

**NAME**

SET TEXT COLOUR INDEX – create a structure element to set the current text colour index attribute

**SYNOPSIS**

**C Syntax**

```
void
pset_text_colr_ind ( colour )
Pint            colour;          text colour index
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET TEXT COLOUR INDEX creates a structure element containing a value for the current text colour index attribute.

When the current text colour index Aspect Source Flag (ASF) is set to INDIVIDUAL, this attribute indexes the current colour representation that applies to the output primitives:

- TEXT
- TEXT 3
- ANNOTATION TEXT RELATIVE
- ANNOTATION TEXT RELATIVE 3

**C Input Parameter**

*colour* An integer colour index, which selects a colour value from the workstation's colour table.

**Execution**

If the current edit mode is INSERT, a SET TEXT COLOUR INDEX element is inserted into the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, the new SET TEXT COLOUR INDEX element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

When the SET TEXT COLOUR INDEX element is traversed, the current text colour index entry in the traversal state list is set to the colour index.

When the current text colour index ASF is set to INDIVIDUAL, the text output primitives that follow in the structure network are drawn with the colour representation selected by the current text colour index from the workstation's colour table.

If the colour index specified is not available on the workstation, colour index 1 is used.

When the current text colour index ASF is set to BUNDLED, the text colour index is taken from the workstation's representation indicated by the current text index. In this case, the colour index set with SET TEXT COLOUR INDEX has no effect.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

113 Ignoring function, the colour index value is less than zero

**SEE ALSO**

COLOUR (7P)
SET COLOUR REPRESENTATION (3P)
SET INDIVIDUAL ASF (3P)
SET TEXT REPRESENTATION (3P)
SET TEXT COLOUR (3P+)
SET TEXT REPRESENTATION PLUS (3P+)

**NAME**

SET TEXT FONT – create a structure element to set the current text font attribute

**SYNOPSIS**

**C Syntax**

```
void
pset_text_font ( font )
Pint            font;          text font
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET TEXT FONT creates a structure element containing a value for the current text font attribute, which selects a font from those available on the workstation.

When the *text font* ASF is set to INDIVIDUAL, this attribute applies to the output primitives:

- TEXT
- TEXT 3
- ANNOTATION TEXT RELATIVE
- ANNOTATION TEXT RELATIVE 3

If the current edit mode is INSERT, a SET TEXT FONT element is inserted in the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, the new SET TEXT FONT element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

**C Input Parameters**

*font* The *text font*, specified as an index to the workstation's non-writable table of available fonts.

All the font indices have named constants defined in phigs.h, as shown below.

The fonts available for PCS_ASCII are:

| Constant | Value |
|---|---|
| PFONT_MONO | 1 |

**Execution**

When the SET TEXT FONT element is traversed, the current text font entry in the traversal state list is set to *text font*. When the current text font ASF (Aspect Source Flag) is set to INDIVIDUAL, the *text font* is applied to text strings in output primitives that follow in the structure network.

The text font attribute selects a font from those available on the workstation. The default *text font* is 1; the default *text precision* is String Precision. The default is used when the font selected is not available in the precision selected on the workstation.

When the text font ASF is set to BUNDLED, the text font is taken from the workstation's representation indicated by the *current text index*. In this case, the font set with SET TEXT FONT has no effect.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

SET TEXT PRECISION (3P)
SET INDIVIDUAL ASF (3P)
SET TEXT REPRESENTATION (3P)
INQUIRE TEXT FACILITIES (3P)
SET TEXT REPRESENTATION PLUS (3P+)

**NAME**

SET TEXT INDEX – create a structure element containing a text representation index attribute

**SYNOPSIS**

**C Syntax**

```
void
pset_text_ind ( index )
Pint            index;          text index
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET TEXT INDEX creates a structure element containing a text index number, which selects an entry from the workstation's text bundle table. This attribute applies to the output primitives:

- TEXT
- TEXT 3
- ANNOTATION TEXT RELATIVE
- ANNOTATION TEXT RELATIVE 3

If the current edit mode is INSERT, a SET TEXT INDEX element is inserted into the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, the new SET TEXT INDEX element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

**C Input Parameter**

*index* A *text index* for a text representation on the workstation.

**Execution**

When the SET TEXT INDEX element is traversed, the 'current text index' value is set to the *text index*, which specifies an entry from the workstation's text bundle table. The default *text index* is 1, and if the *text index* specified is not available on the workstation, 1 will be used.

A text representation contains values for the following attributes:

- text font
- text precision
- character expansion factor
- character spacing
- text colour index

Text output primitives which follow in the structure network will be drawn using the values from the specified representation for those attributes whose Aspect Source Flag (ASF) is set to BUNDLED. When the ASFs are INDIVIDUAL, the attributes come from the appropriate SET *attribute* elements.

Text representations are defined with SET TEXT REPRESENTATION.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

100 Ignoring function, the bundle index value is less than one

**SEE ALSO**

SET INDIVIDUAL ASF (3P)
SET TEXT REPRESENTATION (3P)
SET TEXT REPRESENTATION PLUS (3P+)

**NAME**

SET TEXT PATH – create a structure element to set the *current text path* attribute

**SYNOPSIS**

**C Syntax**

```
void
pset_text_path ( path )
Ptext_path          path;          text path
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET TEXT PATH creates a structure element containing a value for the text path attribute, which controls the direction in which the string is written relative to the *character up* and *character base* vectors. This attribute applies to the following output primitives:

- TEXT
- TEXT 3

If the current edit mode is INSERT, a SET TEXT PATH element is inserted into the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, the new SET TEXT PATH element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

**C Input Parameter**

*path* The text path is one of the following enumerated values:

| | |
|---|---|
| PPATH_RIGHT | *Right* |
| PPATH_LEFT | *Left* |
| PPATH_UP | *Up* |
| PPATH_DOWN | *Down* |

**Execution**

When the SET TEXT PATH element is traversed, the current text path entry in the traversal state list is set to text path. The text path determines the direction of displacement between one character and the next in a string, defined in relation to the *character up* and *character base* vectors.

If text path is set to *Right*, then the text string is written along a base line in the direction of the CHARACTER BASE VECTOR. If text path is set to *Left*, then the text string is written along a base line in the direction opposite to the CHARACTER BASE VECTOR. If text path is set to *Up*, then the text string is written in the direction of the CHARACTER UP VECTOR. If text path is set to *Down*, the text string is written in the direction opposite to the CHARACTER UP VECTOR.

**Note:** Text path controls only the direction in which the string is written. The position of the string in relation to the text position point is controlled by the text alignment.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

SET TEXT ALIGNMENT (3P)
SET CHARACTER UP VECTOR (3P)

**NAME**

SET TEXT PRECISION – create a structure element to set the current text precision attribute

**SYNOPSIS**

**C Syntax**

```
void
pset_text_prec ( precision )
Ptext_prec          precision;          text precision
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET TEXT PRECISION creates a structure element containing a value for the current text precision attribute, which controls how closely the text must be drawn in relation to the font definition and the applicable text attributes.

When the current text precision Aspect Source Flag (ASF) is set to INDIVIDUAL, this attribute defines the text precision applied to the output primitives:

- TEXT
- TEXT 3
- ANNOTATION TEXT RELATIVE
- ANNOTATION TEXT RELATIVE 3

**C Input Parameter**

*precision*

The text precision is an enumerated type with one of the following values:

| | |
|---|---|
| PPREC_STRING | *String Precision* |
| PPREC_CHAR | *Character Precision* |
| PPREC_STROKE | *Stroke Precision* |

**Execution**

If the current edit mode is INSERT, a SET TEXT PRECISION element is inserted into the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, the new SET TEXT PRECISION element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

When the SET TEXT PRECISION element is traversed, the current text precision entry in the traversal state list is set to text precision.

When the current text precision ASF is set to INDIVIDUAL, the text output primitives that follow in the structure network are then drawn with the precision specified.

The text precision attribute allows the tradeoff of quality and performance for text primitives.

*Stroke* Stroke precision demands adherence to all text attributes. Clipping must be performed on portions of characters, but need not involve vector strokes. Stroke precision provides the highest quality available.

*Character*

Character precision allows the implementation to clip on a character-by-character basis. If any portion of a character is outside the clipping limits, the entire character can be clipped.

*String* String precision allows the implementation to ignore the *character up* and *character base* vectors, text path, text alignment, and character spacing, and permits clipping in an implementation-dependent way.

The default text font is 1. The default text precision is *String* precision. The default is used when the font selected is not available in the selected precision on the workstation.

When the current text precision ASF is set to BUNDLED, the text precision is taken from the workstation's representation indicated by the current text index. In this case, the text precision set with SET TEXT PRECISION has no effect.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

SET TEXT FONT (3P)
SET INDIVIDUAL ASF (3P)
SET TEXT REPRESENTATION (3P)
SET TEXT REPRESENTATION PLUS (3P+)

**NAME**

SET TEXT REPRESENTATION – define a text attribute bundle on a workstation

**SYNOPSIS**

**C Syntax**

```
void
pset_text_rep ( ws, index, rep )
Pint            ws;         workstation identifier
Pint            index;      text bundle index
Ptext_bundle    *rep;       text representation pointer
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

SET TEXT REPRESENTATION defines a bundle of text attributes for a specified entry in the workstation's text bundle table. Depending on the ASF for each of the text attributes, the bundled attributes may apply to the following primitives:

- TEXT
- TEXT 3
- ANNOTATION TEXT RELATIVE
- ANNOTATION TEXT RELATIVE 3

**C Input Parameters**

*ws* The identifier of the workstation for which the text representation is being defined.

*index* The *text index* of the entry being defined.

*rep* A pointer to a structure containing the attribute values defining the text representation, defined as follows:

```
typedef struct {
        Pint        font;       /* text font */
        Ptext_prec  prec;       /* text precision */
        Pfloat      char_expan; /* character char_expansion factor */
        Pfloat      char_space; /* character spacing */
        Pint        colr_ind;   /* text colour index */
} Ptext_bundle;
```

*rep->font* is a *text font index*.

*rep->prec* is a *text precision* enumeration value, one of:

| | |
|---|---|
| PPREC_STRING | *String Precision* |
| PPREC_CHAR | *Character Precision* |
| PPREC_STROKE | *Stroke Precision* |

*rep->char_expan* is the *character expansion factor* value.

*rep->char_space* is the *character spacing factor* value.

*rep->colr_ind* is the *text colour*, which selects a colour value from the workstation's colour table.

**Execution**

When SET TEXT REPRESENTATION is called, the text index entry in the table of defined text representations on the workstation is set to the text font index, text precision, character expansion factor, character spacing factor, and text colour values.

When text output primitives are displayed, the text representation specified by the current text index entry in the traversal state list provides the text attributes for which the Aspect Source Flag (ASF) is BUNDLED. For example, when the current text font ASF is set to BUNDLED, the effective text font is the text font attribute in the text representation selected by the current text index. The current text index is set by SET TEXT INDEX structure elements.

The text precision specifies the level of precision used to draw the text characters in relation to the font definition and applicable text attributes.

The character expansion factor is used to scale the standard width of the characters established by the width-to-height ratio specified by the font design. An expansion factor of less than one produces narrower characters; an expansion factor of greater than one produces wider characters, relative to their height.

The character spacing factor is specified as a fraction of the current character height. A positive value inserts additional space between adjacent character bodies in the text string; a negative value causes the adjacent character bodies to overlap. The default value is 0.0, so the character bodies are placed adjacent to one another with no additional space beyond that already in the font's design.

The text bundle table is numbered from 1.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT, OUTIN, nor MO)

100 Ignoring function, the bundle index value is less than one

103 Ignoring function, setting this bundle table entry would exceed the maximum number of entries allowed in the workstation bundle table

106 Ignoring function, the specified font is not available for the requested text precision on the specified workstation

113 Ignoring function, the colour index value is less than zero

**SEE ALSO**

SET TEXT INDEX (3P)
SET INDIVIDUAL ASF (3P)
INQUIRE TEXT REPRESENTATION (3P)
INQUIRE TEXT REPRESENTATION PLUS (3P+)
INQUIRE TEXT FACILITIES (3P)
SET TEXT FONT (3P)
SET TEXT PRECISION (3P)
SET CHARACTER EXPANSION FACTOR (3P)
SET CHARACTER SPACING (3P)
SET TEXT COLOUR (3P+)
SET TEXT COLOUR INDEX (3P)
SET TEXT REPRESENTATION PLUS (3P+)

**NAME**

SET TEXT REPRESENTATION PLUS – use to define an extended text attribute representation bundle on a workstation

**SYNOPSIS**

**C Syntax**

```
void
pset_text_rep_plus ( ws, index, rep )
Pint              ws;          workstation identifier
Pint              index;       text bundle index
Ptext_bundle_plus *rep;        text representation pointer
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

SET TEXT REPRESENTATION PLUS defines an extended text attribute representation bundle on a workstation. Depending on the ASF for each of the text attributes, the bundled attributes may apply to the following primitives:

- TEXT
- TEXT 3
- ANNOTATION TEXT RELATIVE
- ANNOTATION TEXT RELATIVE 3

**C Input Parameters**

*ws* The identifier of the workstation.

*index* This is an index to the workstation text bundle table.

*rep* A pointer to a Ptext_bundle_plus structure. Ptext_bundle_plus is defined as:

```
typedef struct {
        Pint        font;       /* text font */
        Ptext_prec  prec;       /* text precision */
        Pfloat      char_expan; /* character expansion factor */
        Pfloat      char_space; /* character spacing */
        Pgcolr      colr_ind;   /* text colour */
} Ptext_bundle_plus;
```

Ptext_prec is defined as:

```
typedef enum {
        PPREC_STRING,
        PPREC_CHAR,
        PPREC_STROKE
} Ptext_prec;
```

Pgcolr is defined as:

```
typedef struct {
        Pint        type;       /* indirect, RGB, CIE, HSV, HLS */
        union {
            Pint    ind;        /* index in workstation colour bundle table */
            struct {
                Pfloat  x;      /* red, hue, etc. */
                Pfloat  y;      /* green, saturation, lightness, etc. */
```

```
            Pfloat          z;          /* blue, value, saturation, etc. */
        } general;
    } val;
} Pgcolr;
```

**Execution**

In the text bundle table of the workstation state list, the given index is associated with the specified parameters.

The text precision value determines the fidelity with which the other aspects are used. The values of text precision, in order of increasing fidelity, are STRING, CHAR, and STROKE. During structure traversal, if the specified text precision is not available on a workstation, the value STRING is used on that workstation.

The character expansion factor specifies the deviation of the width to height ratio of the characters from the ratio indicated by the font designer. During structure traversal, only the magnitude of the character expansion factor is considered. If the result of taking the absolute value of the character expansion factor value is less than the smallest supported character expansion factor on a workstation, the smallest supported character expansion factor on that workstation is used.

Character spacing specifies how much additional space is inserted between two adjacent character bodies. Character spacing is specified as a fraction of the font-nominal character height.

Text colour is a general colour, either an indirect or a direct colour in a specified colour model. If an indirect colour is specified and the specified colour index is not available during structure traversal, then colour index one is used.

The extended text bundle table in the workstation state list has predefined entries taken from the workstation description table. A number of these entries are predefined for every workstation of category OUTPUT or OUTIN. Any table entry, including the predefined entries, may be redefined with this function.

When either text, or annotation text, is displayed during structure traversal, the current text index in the PHIGS traversal state list refers to an entry in the extended text bundle table. If the current text index is not present in the extended text bundle table, then text index one is used. The setting of the corresponding Aspect Source Flags(ASFs) determines which aspects in the entry are used.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT, OUTIN, nor MO)

100 Ignoring function, the bundle index value is less than one

103 Ignoring function, setting this bundle table entry would exceed the maximum number of entries allowed in the workstation bundle table

106 Ignoring function, the specified font is not available for the requested text precision on the specified workstation

110 Ignoring function, the specified colour model is not available on the workstation

113 Ignoring function, the colour index value is less than zero

136 Ignoring function, one of the components of the color specification is out of range

**SEE ALSO**

SET INDIVIDUAL ASF (3P)
INQUIRE TEXT REPRESENTATION PLUS (3P+)
SET TEXT REPRESENTATION (3P)
SET TEXT INDEX (3P)
SET TEXT FONT (3P)
SET TEXT PRECISION (3P)

SET CHARACTER EXPANSION FACTOR (3P)
SET CHARACTER SPACING (3P)
SET TEXT COLOUR (3P+)
SET TEXT COLOUR INDEX (3P)
INQUIRE TEXT REPRESENTATION (3P)

**NAME**

SET VALUATOR MODE – set valuator device operating mode and echoing state

**SYNOPSIS**

**C Syntax**

```
void
pset_val_mode ( ws, dev, mode, echo )
Pint            ws;         workstation identifier
Pint            dev;        valuator device number
Pop_mode        mode;       operating mode
Pecho_switch    echo;       echo switch
```

**Required PHIGS Operating States**

(PHOP, WSOP, ∗, ∗)

**DESCRIPTION**

**Purpose**

SET VALUATOR MODE sets the operating mode and echo switch for a specified valuator device on a specified workstation.

**C Input Parameters**

*ws* The workstation identifier of the workstation associated with the device.

*dev* The device number of the valuator device to be set. See the *Available Devices* section in INITIALIZE VALUATOR 3 for a description of the available devices.

*mode* Specifies the operating mode for the specified valuator device. Pop_mode is an enumeration defined in phigs.h as follows:

```
typedef enum {
        POP_REQ,
        POP_SAMPLE,
        POP_EVENT
} Pop_mode;
```

*echo* The echo switch value for the specified valuator device. Pecho_switch is an enumeration defined in phigs.h as follows:

```
typedef enum {
        PSWITCH_NO_ECHO,
        PSWITCH_ECHO
} Pecho_switch;
```

**Execution**

SET VALUATOR MODE sets the operating mode and echo flag of the specified device. The default operating mode is POP_REQ. The default echo switch is PSWITCH_ECHO.

The operating mode controls how the input from the device is obtained:

- If the operating mode is REQUEST, REQUEST VALUATOR may be used to prompt for and receive input from a valuator input device. REQUEST VALUATOR will not return until the operator enters input to the device.

- If the operating mode is SAMPLE, SAMPLE VALUATOR may be used to return the current input value of the device without waiting for operator input.

- If the operating mode is EVENT, the input values generated by the operator are added to the PHIGS input event queue. The functions AWAIT EVENT and GET VALUATOR may then be used to retrieve these events from the input queue.

The echo switch may be usd by some devices to control whether or not values selected by the operator are echoed to the device.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

061 Ignoring function, specified workstation is neither of category INPUT nor of category OUTIN

250 Ignoring function, the specified device is not available on the specified workstation

**SEE ALSO**

INITIALIZE VALUATOR (3P)
REQUEST VALUATOR (3P)
GET VALUATOR (3P)
SAMPLE VALUATOR (3P)
INQUIRE VALUATOR DEVICE STATE (3P)

**NAME**

SET VIEW INDEX – create a structure element containing a view index attribute

**SYNOPSIS**

**C Syntax**

```
void
pset_view_ind ( index )
Pint            index;          view index
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

SET VIEW INDEX creates a structure element containing a view index number, which selects an entry from the workstation's view table. The view representation selected maps primitives from World Coordinates to Normalized Projection Coordinates. This attribute applies to all output primitives.

If the current edit mode is INSERT, a SET VIEW INDEX element is inserted into the currently open structure after the element pointed to by the current element pointer. If the edit mode is REPLACE, the new SET VIEW INDEX element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

**C Input Parameter**

*index* A *view index* for a view representation on the workstation.

**Execution**

When the SET VIEW INDEX element is traversed, the 'current view index' value is set to the *view index*, which specifies an entry from the workstation's view table. The default *view index* is 0, and if the *view index* specified is not available on the workstation, view index 0 will be used.

The view representation maps output primitives from World Coordinates to Normalized Projection Coordinates by:

- Applying a view orientation matrix to transform the primitives to the Viewing Reference Coordinate system (VRC),
- Applying a view mapping matrix to transform the primitives to the Normalized Projection Coordinate system (NPC), and
- Optionally clipping the primitives to specified clipping limits in NPC according to separate clipping indicators for the *x*, *y*, and *z* dimensions.

A view representation is set using SET VIEW REPRESENTATION or SET VIEW REPRESENTATION 3. See the descriptions of these functions for more information on the representations and their default values.

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

114 Ignoring function, the view index value is less than zero

**SEE ALSO**

SET VIEW REPRESENTATION (3P)
SET VIEW REPRESENTATION 3 (3P)

**NAME**

SET VIEW REPRESENTATION – define a 2D view representation entry on a workstation

**SYNOPSIS**

**C Syntax**

```
void
pset_view_rep ( ws, index, rep )
Pint            ws;          workstation identifier
Pint            index;       view index
Pview_rep       *rep;        view representation
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

SET VIEW REPRESENTATION defines a view representation entry in the workstation's view table, using a two-dimensional model. A view representation defines how the composited objects are to be represented in relation to the viewer.

A view representation controls the viewing stage of the transformation pipeline, which transforms coordinates in World Coordinates (WC) to Normalized Projection Coordinates (NPC), and optionally clips to the limits of NPC space.

**C Input Parameters**

*ws* The identifier of the workstation for which the view representation is being defined.

*index* The view index of the entry being defined.

*rep* A pointer to a Pview_rep structure containing the values in the view representation, defined in phigs.h as follows:

```
typedef struct {
        Pmatrix     ori_matrix;     /* view orientation matrix */
        Pmatrix     map_matrix;     /* view mapping matrix */
        Plimit      clip_limit;     /* clipping limits */
        Pclip_ind   xy_clip;        /* x-y clipping indicator */
} Pview_rep;
```

Pmatrix is defined in phigs.h as follows:

```
typedef  Pfloat  Pmatrix[3][3];
```

Plimit is defined in phigs.h as follows:

```
typedef struct {
        Pfloat    x_min;   /* x min */
        Pfloat    x_max;   /* x max */
        Pfloat    y_min;   /* y min */
        Pfloat    y_max;   /* y max */
} Plimit;
```

Pclip_ind is defined in phigs.h as follows:

```
typedef enum {
        PIND_NO_CLIP,
        PIND_CLIP
} Pclip_ind;
```

The *ori_matrix* is the $3 \times 3$ homogeneous view orientation matrix, probably from EVALUATE VIEW ORIENTATION MATRIX.

The *map_matrix* is the $3 \times 3$ homogeneous view mapping matrix, probably from EVALUATE VIEW MAPPING MATRIX.

The *clip_limit* contains the 4 Pfloat clipping limits: *x_min, x_max, y_min,* and *y_max*.

The *xy_clip* contains the *x-y* clip indicator for the *x* and *y* axes.

**Execution**

SET VIEW REPRESENTATION defines a view representation entry in the workstation's view table using a two-dimensional model.

The *current view index* attribute applied to each output primitive at traversal time determines which view representation is used to map the primitive from WC space to NPC space, typically with a parallel or perspective projection.

This viewing process for each output primitive is accomplished in three stages:

1. The *view orientation matrix* is applied to orient View Reference Coordinate (VRC) space with respect to WC space. Use the utility function EVALUATE VIEW ORIENTATION MATRIX to generate this matrix from a *view reference point* and *view up vector* specified in WC. The view reference point becomes the origin of VRC and the view up vector orients the VRC *v* axis.

2. The *view mapping matrix* is applied to map the VRC system to the NPC system. Use the utility function EVALUATE VIEW MAPPING MATRIX to generate this matrix. The utility accepts a specified *view area* called a window in VRC space to be mapped onto a specified *projection viewport* in NPC space. The projection is established in relation to the *projection reference point,* also specified in VRC space. The projection may be perspective or parallel, oblique or non-oblique.

3. Clipping limits are optionally applied. The *view clipping limits* define an area of NPC. When a clipping indicator is set to *Clipping,* any portion of an object beyond that limit is not displayed.

   **Note**: These clipping limits are typically identical to the projection viewport provided to EVALUATE VIEW MAPPING MATRIX. If the clipping limits differ from the projection viewport, the object portions mapped by the *view mapping matrix* to be within the clipping limits in NPC are displayed.

In any case, objects are subject to workstation clipping, which may not be disabled. The maximum volume of NPC space that may be displayed is from 0 to 1 in all dimensions.

The immediate visual effect of changing a view representation depends on the workstation's *display update state*. When SET VIEW REPRESENTATION is called, the display update state may prohibit immediate updating (that is, regeneration) of the workstation's display. In this case, the *view index* view table entry's *viewing transformation update state* is set to PENDING, and the following values for the entry are set to the values being *requested* by the function invocation:

- Requested view orientation matrix
- Requested view mapping matrix
- Requested view clipping limits
- Requested *x-y* clipping indicator

The next time the workstation is updated, for any view table entry with the viewing transformation update state set to PENDING, the values stored in the *requested* entries replace the corresponding *current* values for that *view index*, and the view entry's viewing transformation update state is set to NOTPENDING. Then regeneration proceeds, and the new viewing representations take effect.

**Default Representation**

View index 0 is the default for the *current view index* in the traversal state list. View table entry 0 is the default view representation, and cannot be changed. The effect of the default representation is to map the VRC rectangle [0,1] x [0,1] to the NPC rectangle $[0,1] \times [0,1]$ without scaling, rotation, or translation. It contains the following values:

- view orientation matrix *Identity Matrix*
- view mapping matrix *Identity Matrix*
- view clipping limits *Xmin = 0 Xmax = 1*
  *Ymin = 0 Ymax = 1*
- *x-y* clipping indicator *Perform Clipping*

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

057 Ignoring function, specified workstation is of category MI

115 Ignoring function, the view index value is less than one

150 Ignoring function, setting this view table entry would exceed the maximum number of entries allowed in the workstation's view table

153 Ignoring function, invalid view clipping limits; XMIN ≥ XMAX, YMIN ≥ YMAX, or ZMIN > ZMAX

154 Ignoring function, the view clipping limits are not within NPC range

**SEE ALSO**

SET VIEW INDEX (3P)
EVALUATE VIEW ORIENTATION MATRIX (3P)
EVALUATE VIEW MAPPING MATRIX (3P)
SET VIEW TRANSFORMATION INPUT PRIORITY (3P)
INQUIRE VIEW REPRESENTATION (3P)
UPDATE WORKSTATION (3P)
SET VIEW REPRESENTATION 3 (3P)

**NAME**

SET VIEW REPRESENTATION 3 – define 3D view representation entry on a workstation

**SYNOPSIS**

**C Syntax**

```
void
pset_view_rep3 ( ws, index, rep )
Pint            ws;         workstation identifier
Pint            index;      view index
Pview_rep3      *rep;       view representation
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

SET VIEW REPRESENTATION 3 defines a view representation entry in the workstation's view table, using a three-dimensional model. A view representation defines how the composited objects are to be represented in relation to the viewer.

A view representation controls the viewing stage of the transformation pipeline, which transforms coordinates in World Coordinates (WC) to Normalized Projection Coordinates (NPC), and optionally clips to the limits of NPC space.

**C Input Parameters**

*ws* The identifier of the workstation for which the view representation is being defined.

*index* The view index of the entry being defined.

*rep* A pointer to a structure containing the values in the 3D view representation, defined as follows:

```
typedef struct {
        Pmatrix3    ori_matrix;     /* orientation matrix */
        Pmatrix3    map_matrix;     /* mapping matrix */
        Plimit3     clip_limit;     /* clipping limits */
        Pclip_ind   xy_clip;        /* x-y clipping indicator */
        Pclip_ind   back_clip;      /* back clipping indicator */
        Pclip_ind   front_clip;     /* front clipping indicator */
} Pview_rep3;
```

Pmatrix3 is defined in phigs.h as follows:

```
typedef  Pfloat  Pmatrix3[4][4];
```

Plimit3 is defined in phigs.h as follows:

```
typedef struct {
        Pfloat    x_min;   /* x min */
        Pfloat    x_max;   /* x max */
        Pfloat    y_min;   /* y min */
        Pfloat    y_max;   /* y max */
        Pfloat    z_min;   /* z min */
        Pfloat    z_max;   /* z max */
} Plimit3;
```

Pclip_ind is defined in phigs.h as follows:

```
typedef enum {
        PIND_NO_CLIP
        PIND_CLIP
} Pclip_ind;
```

The *ori_matrix* is the $4 \times 4$ homogeneous *view orientation matrix*, probably from EVALUATE VIEW ORIENTATION MATRIX 3.

The *map_matrix* is the $4 \times 4$ homogeneous *view mapping matrix*, probably from EVALUATE VIEW MAPPING MATRIX 3.

The *clip_limit* contains the 6 Pfloat clipping limits: *xmin*, *xmax*, *ymin*, *ymax*, *zmin*, and *zmax*.

The *xy_clip* contains the *x*-*y* clip indicator for the *x* and *y* axes.

The *back_clip* contains the back clipping indicator, which controls clipping against the *back* plane, sometimes called the *yon* plane.

The *front_clip* contains the front clipping indicator, which controls clipping against the *front* plane, sometimes called the *fore* plane.

**Execution**

SET VIEW REPRESENTATION 3 defines a view representation entry in the workstation's view table using a three-dimensional model.

The *current view index* attribute applied to each output primitive at traversal time determines which view representation is used to map the primitive from WC space to NPC space, typically with a parallel or perspective projection.

This viewing process for each output primitive is accomplished in three stages:

1. The *view orientation matrix* is applied to orient View Reference Coordinate (VRC) space with regard to WC space. Use the utility function EVALUATE VIEW ORIENTATION MATRIX 3 to generate this matrix from a *view reference point*, *view plane normal vector*, and *view up vector* specified in WC. The view reference point becomes the origin of VRC; the view plane normal and the view up vector orient the VRC space (VRC *u* and *y* axes) in relation to the WC axes.

2. The *view mapping matrix* is applied to map the VRC system to the NPC system. Use the utility function EVALUATE VIEW MAPPING MATRIX 3 to generate this matrix. The utility accepts a specified *view volume* called a window in VRC space to be mapped onto a specified *projection viewport* in NPC space. The projection is established in relation to the *projection reference point*, also specified in VRC space. The projection may be perspective or parallel, oblique or non-oblique.

3. Clipping limits are optionally applied. The *view clipping limits* define an area of NPC. When a clipping indicator is set to *Clipping*, any portion of an object beyond that limit is not displayed.

   **Note**: These clipping limits are typically identical to the projection viewport provided to EVALUATE VIEW MAPPING MATRIX 3. If the clipping limits differ from the projection viewport, the object portions mapped by the view mapping matrix to be within the clipping limits in NPC are displayed.

In any case, objects are subject to workstation clipping, which may not be disabled. The maximum volume of NPC space that may be displayed is from 0 to 1 in all dimensions.

The immediate visual effect of changing a view representation depends on the workstation's *display update state*. When SET VIEW REPRESENTATION is called, the display update state may prohibit immediate updating (that is, regeneration) of the workstation's display. In this case, the *view index* view table entry's *viewing transformation update state* is set to PENDING, and the following values for the entry are set to the values being *requested* by the function invocation:

- Requested view orientation matrix
- Requested view mapping matrix
- Requested view clipping limits
- Requested *x-y* clipping indicator
- Requested back clipping indicator
- Requested front clipping indicator

The next time the workstation is updated, for any view table entry with the viewing transformation update state set to PENDING, the values stored in the *requested* entries replace the corresponding *current* values for that view index, and the view entry's viewing transformation update state is set to NOTPENDING. Then regeneration proceeds, and the new viewing representations take effect.

**Default Representation**

View index 0 is the default for the *current view index* in the traversal state list. View table entry 0 is the default view representation, and cannot be changed. The effect of the default representation is to map the cube $[0,1] \times [0,1] \times [0,1]$ in VRC space to the cube $[0,1] \times [0,1] \times [0,1]$ in NPC space, without any scaling, rotation, or translation, and to clip the image in NPC space at the boundaries of the same unit cube. It contains the following values:

- view orientation matrix — *Identity Matrix*
- view mapping matrix — *Identity Matrix*
- view clipping limits — *Xmin = 0 Xmax = 1*<br>*Ymin = 0 Ymax = 1*<br>*Zmin = 0 Zmax = 1*
- *x-y* clipping indicator — *Perform Clipping*
- back clipping indicator — *Perform Clipping*
- front clipping indicator — *Perform Clipping*

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

057 Ignoring function, specified workstation is of category MI

115 Ignoring function, the view index value is less than one

150 Ignoring function, setting this view table entry would exceed the maximum number of entries allowed in the workstation's view table

153 Ignoring function, invalid view clipping limits; XMIN ≥ XMAX, YMIN ≥ YMAX, or ZMIN > ZMAX

154 Ignoring function, the view clipping limits are not within NPC range

**SEE ALSO**

SET VIEW INDEX (3P)
EVALUATE VIEW ORIENTATION MATRIX (3P)
EVALUATE VIEW MAPPING MATRIX (3P)
SET VIEW TRANSFORMATION INPUT PRIORITY (3P)
INQUIRE VIEW REPRESENTATION (3P)
UPDATE WORKSTATION (3P)
SET VIEW REPRESENTATION (3P)

**NAME**

SET VIEW TRANSFORMATION INPUT PRIORITY – assign relative priority of a view representation to be used when transforming input values

**SYNOPSIS**

**C Syntax**

```
void
pset_view_tran_in_pri ( ws, index, ref_index, priority )
Pint        ws;          workstation identifier
Pint        index;       view index
Pint        ref_index;   reference view index
Prel_pri    priority;    relative priority
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

SET VIEW TRANSFORMATION INPUT PRIORITY assigns a relative priority to an entry in the workstation's view table, to be used when transforming LOCATOR or STROKE operator input coordinates from Device Coordinates (DC) to World Coordinates (WC).

**C Input Parameters**

*ws* The identifier of the workstation for which the view transformation input priority is being set.

*index* The priority is being set for this *view index*.

*ref_index*
The priority of *index* is set just higher or lower than that of this *reference view index*.

*priority* The *relative priority* for *index*, relative to the *ref_index*; one of the enumerated values:

| | |
|---|---|
| PPRI_HIGHER | *Higher* |
| PPRI_LOWER | *Lower* |

**Execution**

SET VIEW TRANSFORMATION INPUT PRIORITY assigns a relative priority to a view representation entry in the workstation's view table, to be used when transforming LOCATOR or STROKE operator input coordinates from DC to WC. The priority of view index is set just higher or just lower than the priority of the reference view index.

LOCATOR or STROKE operator input data are transformed by the inverse viewing pipeline, from DC to WC. First, the inverse workstation transformation (in effect when the input is generated) maps the DC values to Normalized Projection Coordinates (NPC), always resulting in values in the NPC unit cube. Then, the NPC positions need to be mapped to WC by the inverse of one of the viewing transformations; the relative priority order of the viewing transformations is used to determine which viewing transformation inverse to apply.

Views with lower priority than view 0's will never be used to map input data. View 0 is the identity transformation encompassing the entire NPC space cube and cannot be changed. Therefore, any input position is necessarily in view 0, and lower priority views are effectively disabled. The initial ordering, when a workstation is opened, gives view number 0 the highest priority, view 1 the next highest, and so on. All views except 0 are disabled.

If the view index is the same as the reference view index, the function has no effect.

The relative view transformation input priorities may be obtained using INQUIRE LIST OF VIEW INDICES.

**ERRORS**

| | |
|---|---|
| 003 | Ignoring function, function requires state (PHOP, WSOP, *, *) |
| 054 | Ignoring function, the specified workstation is not open |
| 057 | Ignoring function, specified workstation is of category MI |
| 114 | Ignoring function, the view index value is less than zero |
| 101 | Ignoring function, the specified representation has not been defined |

**SEE ALSO**

SET VIEW REPRESENTATION 3 (3P)
INQUIRE LIST OF VIEW INDICES (3P)

**NAME**

SET WORKSTATION VIEWPORT – set 2D viewport limits for a specified workstation

**SYNOPSIS**

**C Syntax**

```
void
pset_ws_vp ( ws, viewport )
Pint            ws;             workstation id
Plimit          *viewport;      workstation viewport limits
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

SET WORKSTATION VIEWPORT defines a 2D area in Device Coordinate (DC) space, at a constant *z* value. The abstract image within the workstation window, defined in Normalized Projection Coordinate (NPC) space, is mapped to this viewport.

Together, the workstation window and the workstation viewport define the *workstation transformation* that converts the image from NPC to DC of workstation's physical display surface. The workstation window defines what within NPC space is displayed; the workstation viewport defines where the image is displayed in DC space.

Control of the workstation transformation allows you to choose a portion of the abstract image composed in NPC space for display on the workstation, without changing the definition of the image in NPC space.

**C Input Parameters**

*ws* The identifier of the workstation for which the viewport is to be set.

*viewport*

A pointer to a structure containing the DC limits for the 2D workstation viewport, defined as follows:

```
typedef struct  {
        Pfloat  x_min;    /* x minimum */
        Pfloat  x_max;    /* x maximum */
        Pfloat  y_min;    /* y minimum */
        Pfloat  y_max;    /* y maximum */
} Plimit;
```

**Execution**

The current workstation viewport defines the area in Device Coordinates in which the abstract image within the current workstation window is displayed. The workstation window is defined in Normalized Projection Coordinates. Together, the workstation window and the workstation viewport define the *workstation transformation* that converts the image from NPC to DC of workstation's physical display surface.

The *x* minimum must be non-negative and less than *x* maximum and *y* minimum must be non-negative and less than *y* maximum. In addition, all values must be within the workstation's DC range. The default PEX-SI workstation DC maximum is 1, but it is selectable by using WORKSTATION TYPE SET (3P).

SET WORKSTATION VIEWPORT sets the *x* and *y* components of the requested workstation viewport in the specified workstation's state list to the values specified. The *z* component of the requested workstation viewport and current workstation viewport are not changed. The effect of calling SET WORKSTATION VIEWPORT is visible only after the requested workstation viewport replaces the current workstation viewport. The time at which this occurs depends on the workstation's display update state. This action is performed immediately, and the workstation transformation update state is set to NOTPENDING, if any one of the following is true:

- The workstation's display update state allows update.

- The workstation's modification mode is any value other than No Immediate Visual Effect, and the dynamic modification accepted for workstation transformation entry in the workstation description table is set to Immediate.
- The display space empty status in the workstation state list is EMPTY.

Otherwise, the workstation transformation update state is set to PENDING, and the requested workstation viewport will not replace the current workstation viewport until the next time the workstation is updated. The workstation transformation update state will be set to NOTPENDING at that time.

If the current workstation window and viewport do not have the same aspect ratios, the workstation transformation preserves the proportions of the image by mapping the workstation window to the largest possible area of the workstation viewport so that:

- The aspect ratio of the window in *x* and *y* is maintained.
- The lower left hand corner of the window is mapped to the lower left hand corner of the viewport.

If the aspect ratios of the workstation window and viewport are different, there will be unused space along the upper or right-hand edges of the viewport, but not both.

The default workstation transformation maps the entire NPC view plane, $[0,1] \times [0,1] \times [0,1]$, onto the largest square area in the workstation display space including the display's lower left corner.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

057 Ignoring function, specified workstation is of category MI

152 Ignoring function, invalid viewport; XMIN ≥ XMAX or YMIN ≥ YMAX.

157 Ignoring function, the workstation viewport is not within display space

**SEE ALSO**

SET WORKSTATION WINDOW (3P)
SET WORKSTATION VIEWPORT 3 (3P)

**NAME**

SET WORKSTATION VIEWPORT 3 – set 3D viewport limits for a specified workstation

**SYNOPSIS**

**C Syntax**

```
void
pset_ws_vp3 ( ws, viewport )
Pint        ws;          workstation id
Plimit3     *viewport;   workstation viewport limits
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

SET WORKSTATION VIEWPORT 3 defines a 3D volume in Device Coordinate (DC) space. The abstract image within the workstation window, defined in Normalized Projection Coordinate (NPC) space, will be mapped to this viewport.

Together, the workstation window and the workstation viewport define the *workstation transformation* that converts the image from NPC to DC on the workstation's physical display surface. The workstation window defines what within NPC space will be displayed; the workstation viewport defines where the image is displayed in DC space.

Control of the workstation transformation allows you to choose a portion of the abstract image composed in NPC space for display on the workstation, without changing the definition of the image in NPC space.

**C Input Parameters**

*ws* The identifier of the workstation for which the viewport is to be set.

*viewport*

A pointer to a structure containing the six DC *workstation viewport limits*, defined as follows:

```
typedef struct  {
        Pfloat   x_min;    /* x minimum */
        Pfloat   x_max;    /* x maximum */
        Pfloat   y_min;    /* y minimum */
        Pfloat   y_max;    /* y maximum */
        Pfloat   z_min;    /* z minimum */
        Pfloat   z_max;    /* z maximum */
} Plimit3;
```

**Execution**

The current workstation viewport defines the volume in Device Coordinates in which the abstract image within the current workstation window is displayed. The workstation window is defined in Normalized Projection Coordinates. Used together, the workstation window and the workstation viewport define an isotropic *workstation transformation* that converts the image from NPC to DC of workstation's physical display surface.

The *x* minimum must be non-negative and less than *x* maximum, the *y* minimum must be non-negative and less than *y* maximum, and the *z* minimum must be non-negative and less than *z* maximum. In addition, all values must be within the workstation's DC range. The default PEX-SI workstation DC maximum is 1, but it is selectable by using WORKSTATION TYPE SET (3P).

SET WORKSTATION VIEWPORT 3 sets the requested workstation viewport in the specified workstation's state list to the values specified in *viewport*. The effect of calling SET WORKSTATION VIEWPORT 3 is visible only after the requested workstation viewport replaces the current workstation viewport. The time at which this occurs depends on the workstation's display update state. This action is performed immediately, and the *workstation transformation update state* is set to NOTPENDING, if any one of the following is true:

- The workstation's display update state allows update.
- The workstation's modification mode is any value other than *No Immediate Visual Effect*, and the *dynamic modification accepted for workstation transformation* entry in the workstation description table is set to IMMEDIATE.
- The *display space empty* status in the workstation state list is EMPTY.

Otherwise, the workstation transformation update state is set to PENDING, and the requested workstation viewport will not replace the current workstation viewport until the next time the workstation is updated. The workstation transformation update state will be set to NOTPENDING at that time.

If the current workstation window and viewport do not have the same aspect ratios, the workstation transformation will preserve the proportions of the image by mapping the workstation window to the largest parallelopiped within the workstation viewport so that:

- The aspect ratio of the window in *x* and *y* is maintained.
- The lower left hand corner of the window closest to 0 is mapped to the lower left hand corner of the viewport furthest from the observer.
- The *z* extent of the workstation window is mapped to the entire *z* extent of the workstation viewport.

If the aspect ratios of the workstation window and viewport are different, there will be unused space along the upper or right-hand edges of the viewport, but not both.

The default workstation transformation maps the entire NPC view volume, $[0,1] \times [0,1] \times [0,1]$, onto the largest parallelopiped in the workstation display space including the display's lower left corner furthest from the observer.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

057 Ignoring function, specified workstation is of category MI

152 Ignoring function, invalid viewport; XMIN ≥ XMAX, YMIN ≥ YMAX, or ZMIN > ZMAX

157 Ignoring function, the workstation viewport is not within display space

**SEE ALSO**

SET WORKSTATION WINDOW 3 (3P)
SET WORKSTATION VIEWPORT (3P)

**NAME**

SET WORKSTATION WINDOW – specify a 2D window to be displayed on the workstation

**SYNOPSIS**

**C Syntax**

```
void
pset_ws_win ( ws, window )
Pint            ws;             workstation id
Plimit          *window;        workstation window limits
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

SET WORKSTATION WINDOW defines the 2D area in Normalized Projection Coordinate (NPC) space to be displayed on the specified workstation. The workstation window is a rectangular box in NPC space, which is mapped to the workstation viewport, defined in Device Coordinate (DC) space.

Together, the workstation window and the workstation viewport define the *workstation transformation* that converts the image from NPC to DC on the workstation's physical display surface. The workstation window defines what within NPC space is displayed; the workstation viewport defines where the image is displayed in DC space.

Control of the workstation window allows you to choose a portion of the abstract image composed in NPC space for display on the workstation, without changing the definition of the image in NPC space.

**C Input Parameters**

*ws* The identifier of the workstation for which the window is to be set.

*window* A pointer to a structure containing the four 2D NPC *workstation window limits*, defined as follows:

```
typedef struct  {
        Pfloat  x_min;    /* x minimum */
        Pfloat  x_max;    /* x maximum */
        Pfloat  y_min;    /* y minimum */
        Pfloat  y_max;    /* y maximum */
} Plimit;
```

**Execution**

The current workstation window defines the rectangular box in Normalized Projection Coordinates (NPC) containing the portion of the abstract image to be displayed within the current workstation viewport. Together, the workstation window and the workstation viewport define an isotropic *workstation transformation* that converts the image from NPC to DC on the workstation's physical display surface.

The range for each of the workstation window limits is [0,1]. In addition, *x minimum* must be less than *x maximum*, and *y minimum* must be less than *y maximum*. Output is automatically clipped at the workstation window limits, and this clipping cannot be disabled.

SET WORKSTATION WINDOW sets the *x* and *y* components of the requested workstation window in the specified workstation's state list to the values specified. The *z* component of the requested workstation window and current workstation window is not changed. The effect of calling SET WORKSTATION WINDOW is visible only after the requested workstation window replaces the current workstation window. The time at which this occurs depends on the workstation's display update state. This action is performed immediately, and the *workstation transformation update state* is set to NOTPENDING, if any one of the following is true:

- The workstation's display update state allows update.
- The workstation's modification mode is any value other than *No Immediate Visual Effect*, and the *dynamic modification accepted for workstation transformation* entry in the workstation description table is set to Immediate.

- The *display space empty* status in the workstation state list is EMPTY.

Otherwise, the workstation transformation update state is set to PENDING, and the requested workstation window will not replace the current workstation window until the next time the workstation is updated. The workstation transformation update state will be set to NOTPENDING at that time.

If the current workstation window and viewport do not have the same aspect ratios, the workstation transformation will preserve the proportions of the image by mapping the workstation window to the largest possible area within of workstation viewport so that:

- The aspect ratio of the window in *x* and *y* is maintained.
- The lower left hand corner of the window is mapped to the lower left hand corner of the viewport.

If the aspect ratios of the workstation window and viewport are different, there will be unused space along the upper or right-hand edges of the viewport, but not both.

The default workstation transformation maps the entire NPC view plane, $[0,1] \times [0,1]$, onto the largest square in the workstation display space including the display's lower left corner.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

057 Ignoring function, specified workstation is of category MI

151 Ignoring function, invalid window; XMIN ≥ XMAX or YMIN ≥ YMAX

156 Ignoring function, the workstation window limits are not within NPC range

**SEE ALSO**

SET WORKSTATION VIEWPORT (3P)
SET WORKSTATION WINDOW 3 (3P)

**NAME**

SET WORKSTATION WINDOW 3 – specify a 3D window to be displayed on the specified workstation

**SYNOPSIS**

**C Syntax**

```
void
pset_ws_win3 ( ws, window )
Pint        ws;        workstation id
Plimit3     *window;   workstation window limits
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

SET WORKSTATION WINDOW 3 defines the 3D volume in Normalized Projection Coordinate (NPC) space to be displayed on the specified workstation. The workstation window is a rectangular box in NPC space, which is mapped to the workstation viewport, defined in Device Coordinate (DC) space.

Together, the workstation window and the workstation viewport define the *workstation transformation* that converts the image from NPC to DC on the workstation's physical display surface. The workstation window defines what within NPC space is displayed; the workstation viewport defines where the image is displayed in DC space.

Control of the workstation window allows you to choose a portion of the abstract image composed in NPC space for display on the workstation, without changing the definition of the image in NPC space.

**C Input Parameters**

*ws* The identifier of the workstation for which the window is to be set.

*window* A pointer to a structure containing the six NPC *workstation window limits,* defined as follows:

```
typedef struct  {
        Pfloat    x_min;    /* x minimum */
        Pfloat    x_max;    /* x maximum */
        Pfloat    y_min;    /* y minimum */
        Pfloat    y_max;    /* y maximum */
        Pfloat    z_min;    /* z minimum */
        Pfloat    z_max;    /* z maximum */
} Plimit3;
```

**Execution**

The current workstation window defines the rectangular box in Normalized Projection Coordinates (NPC) containing the portion of the abstract image to be displayed within the current workstation viewport. Together, the workstation window and the workstation viewport define an isotropic *workstation transformation* that converts the image from NPC to DC on the workstation's physical display surface.

The range for each of the *workstation window limits* is [0,1]. In addition, *x minimum* must be less than *x maximum, y minimum* must be less than *y maximum,* and *z minimum* must be less than or equal to *z maximum*. Output is automatically clipped at the workstation window limits, and this clipping cannot be disabled.

SET WORKSTATION WINDOW 3 sets the requested workstation window in the specified workstation's state list to the values specified in *window*. The effect of calling SET WORKSTATION WINDOW 3 is visible only after the requested workstation window replaces the current workstation window. The time at which this occurs depends on the workstation's display update state. This action is performed immediately, and the *workstation transformation update state* is set to NOTPENDING, if any one of the following is true:

- The workstation's display update state allows update.
- The workstation's modification mode is any value other than *No Immediate Visual Effect,* and the

*dynamic modification accepted for workstation transformation* entry in the workstation description table is set to Immediate.

- The *display space empty* status in the workstation state list is EMPTY.

Otherwise, the workstation transformation update state is set to PENDING, and the requested workstation window will not replace the current workstation window until the next time the workstation is updated. The workstation transformation update state will be set to NOTPENDING at that time.

If the current workstation window and viewport do not have the same aspect ratios, the workstation transformation will preserve the proportions of the image by mapping the workstation window to the largest parallelopiped within the workstation viewport so that:

- The aspect ratio of the window in *x* and *y* is maintained.
- The lower left hand corner of the window closest to 0 is mapped to the lower left hand corner of the viewport furthest from the observer.
- The *z* extent of the workstation window is mapped to the entire *z* extent of the workstation viewport.

If the aspect ratios of the workstation window and viewport are different, there will be unused space along the upper or right-hand edges of the viewport, but not both.

The default workstation transformation maps the entire NPC view volume, $[0,1] \times [0,1] \times [0,1]$, onto the largest parallelopiped in the workstation display space including the display's lower left corner furthest from the observer.

**ERRORS**

003 Ignoring function, function requires state ( PHOP, WSOP, *, * )

054 Ignoring function, the specified workstation is not open

057 Ignoring function, specified workstation is of category MI

151 Ignoring function, invalid window; XMIN ≥ XMAX, YMIN ≥ YMAX, or ZMIN > ZMAX

156 Ignoring function, the workstation window limits are not within NPC range

**SEE ALSO**

SET WORKSTATION VIEWPORT 3 (3P)
SET WORKSTATION WINDOW (3P)

**NAME**

TEXT – create structure element specifying 2D text primitive

**SYNOPSIS**

**C Syntax**

```
void
ptext ( text_pt, text )
Ppoint          *text_pt;          text point
char            *text;             text string
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

The TEXT subroutine puts a structure element containing the full specification of a two-dimensional TEXT primitive into the currently open structure.

The TEXT primitive is a character string. The location of the string in the display is controlled by the *text point* subroutine parameter, specified in Modelling Coordinates. The *z* coordinate is assumed to be 0. Aspects of the text display, such as the font, colour, spacing, height, and alignment, are controlled by the current values of the primitive attributes listed below.

If the current edit mode is INSERT, the structure element created by the TEXT subroutine is inserted into the open structure after the element pointed to by the current *element pointer*. If the current edit mode is REPLACE, the new TEXT element replaces the element in the structure pointed to by the *element pointer*. In either case, the *element pointer* is updated to point to the new TEXT element.

**C Input Parameters**

*text_pt* A pointer to a Ppoint structure containing the *x* and *y* coordinates that locate the text string. The Ppoint structure is defined in phigs.h as follows:

```
typedef struct  {
        Pfloat    x;    /* x coordinate */
        Pfloat    y;    /* y coordinate */
} Ppoint;
```

*text* A pointer to the character string to be written into the display.

**Execution**

When the structure is traversed, the TEXT element draws the specified character string. The position of the text is defined in relation to *text point* by the current values of the text primitive attributes CHARACTER UP VECTOR, TEXT PATH, and TEXT ALIGNMENT.

Other aspects of the appearance of the text are controlled by the attributes TEXT FONT, TEXT PRECISION, CHARACTER HEIGHT, CHARACTER WIDTH, CHARACTER EXPANSION FACTOR, CHARACTER SPACING, and TEXT COLOUR INDEX.

The text point is specified in Modelling Coordinates. The text primitive is subject to the current transformations in the transformation pipeline from the Modelling Coordinate System to the workstation display.

**Attributes Applied**

The attributes listed below are used to display the TEXT primitive when the structure is traversed. The Aspect Source Flags (ASFs) tell where to access the output display attributes. These attributes can come directly from the traversal state list, or they can be accessed indirectly, using the appropriate index in the traversal state list and the corresponding bundled representation in the PHIGS workstation state list.

| | |
|---|---|
| text font | text font ASF |
| text precision | text precision ASF |
| character expansion factor | character expansion factor ASF |

| | |
|---|---|
| character spacing | character spacing ASF |
| text colour | text colour index ASF |
| character height | |
| character up vector | |
| text path | |
| text alignment | |
| text index | |
| depth cue index | |
| name set | |

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

INQUIRE TEXT FACILITIES (3P)
INQUIRE TEXT EXTENT (3P)
TEXT 3 (3P)

**NAME**

TEXT 3 – create structure element specifying 3D text primitive

**SYNOPSIS**

**C Syntax**

```
void
ptext3 ( text_pt, dir, text )
Ppoint3          *text_pt;          text point
Pvec3            dir[2];            direction vectors
char             *text;             text string
```

**Required PHIGS Operating States**

(PHOP, *, STOP, *)

**DESCRIPTION**

**Purpose**

The TEXT 3 subroutine puts a structure element containing the full specification of a three-dimensional TEXT 3 primitive into the currently open structure.

The TEXT 3 primitive is a character string. The location and orientation of the string in the display is controlled by the *text point* and *direction vectors* subroutine parameters, specified in Modelling Coordinates (MC). Other aspects of the text display, such as the font, colour, spacing, height, and alignment, are controlled by the current values of the primitive attributes listed below.

If the current edit mode is INSERT, the structure element created by the TEXT 3 subroutine is inserted into the open structure after the element pointed to by the current element pointer. If the current edit mode is REPLACE, the TEXT 3 element replaces the element in the structure pointed to by the element pointer. In either case, the element pointer is updated to point to the new TEXT 3 element.

**C Input Parameters**

*text_pt* A pointer to a Ppoint3 structure that specifies the *x*, *y*, and *z* coordinates that locate the text string. The Ppoint3 structure is defined in phigs.h as follows:

```
typedef struct  {
        Pfloat    x;    /* x coordinate */
        Pfloat    y;    /* y coordinate */
        Pfloat    z;    /* z coordinate */
} Ppoint3;
```

*dir* An array of two Pvec3 structures containing direction vectors. The Pvec3 structure is defined in phigs.h as follows:

```
typedef struct  {
        Pfloat    delta_x;        /* x magnitude */
        Pfloat    delta_y;        /* y magnitude */
        Pfloat    delta_z;        /* z magnitude */
} Pvec3;
```

*text* A pointer to the character string to be written into the display.

**Execution**

When the structure is traversed, the TEXT 3 element draws the specified character string on the plane in the MC system defined by *text point* and the two *direction vectors*. These parameters define a Text Local Coordinate (TLC) system in the MC system. The *text point* parameter defines the origin of this TLC system, the first direction vector defines the positive *x* axis, and the second direction vector defines the positive *y* axis. Only the directions, not the lengths, of these vectors are relevant.

The precise position of the text is defined in relation to this plane by the current values of the text primitive attributes CHARACTER UP VECTOR, TEXT PATH, and TEXT ALIGNMENT. The text primitive is subject to the current transformations in the transformation pipeline from the MC system to the workstation display.

Other aspects of the appearance of the text are controlled by the attributes TEXT FONT, TEXT PRECISION, CHARACTER HEIGHT, CHARACTER WIDTH, CHARACTER EXPANSION FACTOR, CHARACTER SPACING, and TEXT COLOUR INDEX.

**Attributes Applied**

The attributes listed below are used to display the TEXT 3 primitive when the structure is traversed. The Aspect Source Flags (ASFs) tell where to access the output display attributes. These attributes can come directly from the traversal state list, or they can be accessed indirectly, using the appropriate index in the traversal state list and the corresponding bundled representation in the PHIGS workstation state list.

| | |
|---|---|
| text font | text font ASF |
| text precision | text precision ASF |
| character expansion factor | character expansion factor ASF |
| character spacing | character spacing ASF |
| text colour | text colour index ASF |
| character height | |
| character up vector | |
| text path | |
| text alignment | |
| text index | |
| depth cue index | |
| name set | |

**ERRORS**

005 Ignoring function, function requires state (PHOP, *, STOP, *)

**SEE ALSO**

INQUIRE TEXT FACILITIES (3P)
INQUIRE TEXT EXTENT (3P)
TEXT (3P)

**NAME**

TRANSFORM POINT – apply a 2D transformation matrix to a specified 2D point

**SYNOPSIS**

**C Syntax**

```
void
ptran_point ( p, m, error_ind, r)
Ppoint          *p;             point
Pmatrix         m;              transformation matrix
Pint            *error_ind;     OUT error indicator
Ppoint          *r;             OUT transformed point
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use TRANSFORM POINT to calculate the transformed coordinates of a 2D point.

**C Input Parameters**

*p* A pointer to a Ppoint data structure containing the coordinates of the point to be transformed. Ppoint is defined in phigs.h as follows:

```
typedef struct  {
        Pfloat  x;   /* x coordinate */
        Pfloat  y;   /* y coordinate */
} Ppoint;
```

*m* The $3 \times 3$ homogeneous transformation matrix to apply to *p*. Pmatrix is defined in phigs.h as follows:

```
typedef  Pfloat  Pmatrix[3][3];
```

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*r* A pointer to a Ppoint structure in which to store the coordinates of the transformed point.

**Execution**

TRANSFORM POINT applies the $3 \times 3$ homogeneous *transformation matrix* to the 2D *point*, and returns the coordinates of the resulting point:

*transformed point = matrix* X *point*

**ERRORS**

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

**SEE ALSO**

SET LOCAL TRANSFORMATION 3 (3P)
ROTATE (3P)
SCALE (3P)
TRANSLATE (3P)
BUILD TRANSFORMATION MATRIX (3P)
COMPOSE TRANSFORMATION MATRIX (3P)
COMPOSE MATRIX (3P)
TRANSFORM POINT 3 (3P)

**NAME**

TRANSFORM POINT 3 – apply a 3D transformation matrix to a specified 3D point

**SYNOPSIS**

**C Syntax**

```
void
ptran_point3 ( p, m, error_ind, r)
Ppoint3          *p;          point
Pmatrix3         m;           transformation matrix
Pint             *error_ind;  OUT error indicator
Ppoint3          *r;          OUT transformed point
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use TRANSFORM POINT 3 to calculate the transformed coordinates of a 3D point.

**C Input Parameters**

*p* A pointer to a Ppoint3 data structure containing the coordinates of the point to be transformed. Ppoint3 is defined in phigs.h as follows:

```
typedef struct  {
        Pfloat  x;   /* x coordinate */
        Pfloat  y;   /* y coordinate */
        Pfloat  z;   /* z coordinate */
} Ppoint3;
```

*m* The $4 \times 4$ homogeneous transformation matrix to apply to *p*. Pmatrix3 is defined in phigs.h as follows:

```
typedef Pfloat     Pmatrix3[4][4];
```

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*r* A pointer to a Ppoint3 structure in which to store the coordinates of the transformed point.

**Execution**

TRANSFORM POINT applies the $4 \times 4$ homogeneous *transformation matrix* to the 3D *point*, and returns the coordinates of the resulting point:

*transformed point = matrix* X *point*

**ERRORS**

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

**SEE ALSO**

SET LOCAL TRANSFORMATION 3 (3P)
ROTATE X (3P)
ROTATE Y (3P)
ROTATE Z (3P)
SCALE 3 (3P)
TRANSLATE 3 (3P)
BUILD TRANSFORMATION MATRIX 3 (3P)
COMPOSE TRANSFORMATION MATRIX 3 (3P)
COMPOSE MATRIX 3 (3P)

TRANSFORM POINT (3P)

**NAME**

TRANSLATE – calculate a 2D transformation matrix to perform a specified translation

**SYNOPSIS**

**C Syntax**

```
void
ptranslate ( trans_vector, error_ind, m)
Pvec        *trans_vector;      translation vector
Pint        *error_ind;         OUT error indicator
Pmatrix     m;                  OUT transformation matrix
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use TRANSLATE to generate a 2D homogeneous ($3 \times 3$) transformation matrix that performs a 2D translation.

The returned matrix can be passed as an argument to SET LOCAL TRANSFORMATION or SET GLOBAL TRANSFORMATION to modify the modelling transformation that is applied to output primitives during traversal.

**C Input Parameter**

*trans_vector*

A pointer to a Pvec structure containing the Modelling Coordinate translation values to be applied in the *x* and *y* dimensions. Pvec is defined in phigs.h as follows:

```
typedef struct  {
        Pfloat      delta_x;    /* the x axis translation value */
        Pfloat      delta_y;    /* the y axis translation value */
} Pvec;
```

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*m* The $3 \times 3$ homogeneous transformation matrix that performs the specified translation. Pmatrix is defined in phigs.h as follows:

```
typedef  Pfloat  Pmatrix[3][3];
```

**Execution**

TRANSLATE returns a 2D homogeneous ($3 \times 3$) transformation matrix that performs the translation specified by translation vector.

The translation vector specifies the translation distance in the *x* and *y* directions.

**ERRORS**

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

**SEE ALSO**

SET LOCAL TRANSFORMATION (3P)
SET GLOBAL TRANSFORMATION (3P)
SET VIEW REPRESENTATION (3P)
BUILD TRANSFORMATION MATRIX (3P)
COMPOSE TRANSFORMATION MATRIX (3P)
COMPOSE MATRIX (3P)
TRANSFORM POINT (3P)
TRANSLATE 3 (3P)

**NAME**

TRANSLATE 3 – calculate a 3D transformation matrix to perform a specified translation

**SYNOPSIS**

**C Syntax**

```
void
ptranslate3 ( trans_vector, error_ind, m)
Pvec3          *trans_vector;     translation vector
Pint           *error_ind;        OUT error indicator
Pmatrix3       m;                 OUT transformation matrix
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

Use TRANSLATE 3 to generate a 3D homogeneous $(4 \times 4)$ transformation matrix that performs a 3D translation.

The returned matrix may be passed as an argument to SET LOCAL TRANSFORMATION 3 or SET GLOBAL TRANSFORMATION 3 to modify the modelling transformation that is applied to output primitives during traversal.

**C Input Parameter**

*trans_vector*

A pointer to a Pvec3 structure containing the Modelling Coordinate translation values to be applied in the *x*, *y*, and *z* dimensions. Pvec3 is defined in phigs.h as follows:

```
typedef struct  {
        Pfloat      delta_x;    /* the x axis translation value */
        Pfloat      delta_y;    /* the y axis translation value */
        Pfloat      delta_z;    /* the z axis translation value */
} Pvec3;
```

**C Output Parameters**

*error_ind*

A pointer to the location to store the error number of any error detected by this function.

*m* The $4 \times 4$ homogeneous transformation matrix that performs the specified translation. Pmatrix is defined in phigs.h as follows:

```
typedef Pfloat  Pmatrix3[4][4];
```

**Execution**

TRANSLATE 3 returns a 3D homogeneous $(4 \times 4)$ transformation matrix that performs the translation specified by translation vector.

The translation vector specifies the translation distance in the *x*, *y*, and *z* directions.

**ERRORS**

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

**SEE ALSO**

SET LOCAL TRANSFORMATION 3 (3P)
SET GLOBAL TRANSFORMATION 3 (3P)
SET VIEW REPRESENTATION 3 (3P)
BUILD TRANSFORMATION MATRIX 3 (3P)
COMPOSE TRANSFORMATION MATRIX 3 (3P)
COMPOSE MATRIX 3 (3P)
TRANSFORM POINT 3 (3P)
TRANSLATE (3P)

**NAME**

UNPOST ALL STRUCTURES – unpost all structures posted to the specified workstation

**SYNOPSIS**

**C Syntax**

```
void
punpost_all_structs ( ws_id )
Pint            ws_id;          workstation identifier
```

**Required PHIGS Operating States**

(PHOP, WSOP, ∗, ∗)

**DESCRIPTION**

**Purpose**

UNPOST ALL STRUCTURES unposts all of the structures posted to a specified workstation. Following UNPOST ALL STRUCTURES, no structure networks remain eligible for display on the workstation. The immediate effect of the function on the display surface will depend on the workstation's display update state.

**C Input Parameter**

*ws_id* All structures are to be unposted from the workstation with this identifier.

**Execution**

UNPOST ALL STRUCTURES removes all the structures from the *table of posted structures* on the specified workstation.

The unposted structures are not deleted from the Central Structure Store, and may again be posted to this or another workstation. No other workstations are affected.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, ∗, ∗)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (i.e., the workstation category is neither OUTPUT, OUTIN, nor MO)

**SEE ALSO**

POST STRUCTURE (3P)
INQUIRE POSTED STRUCTURES (3P)
CLOSE WORKSTATION (3P)
UNPOST STRUCTURE (3P)

**NAME**

UNPOST STRUCTURE – unpost a structure from the specified workstation

**SYNOPSIS**

**C Syntax**

```
void
punpost_struct ( ws_id, struct_id )
Pint        ws_id;          workstation identifier
Pint        struct_id;      structure identifier
```

**Required PHIGS Operating States**

(PHOP, WSOP, *, *)

**DESCRIPTION**

**Purpose**

UNPOST STRUCTURE unposts a single structure from a workstation. The structure is not deleted from the Central Structure Store (CSS), but it is no longer eligible for display on the workstation.

The immediate effect of the function on the display surface will depend on the workstation's display update state.

**C Input Parameters**

*ws_id* The structure is to be unposted from the workstation with this identifier.

*struct_id*

The identifier of the structure to be unposted.

**Execution**

UNPOST STRUCTURE removes the structure specified from the table of posted structures on the workstation. Unless the structure is part of a structure network posted to the workstation, the structure is no longer eligible for display on the workstation, and changes to the structure will no longer affect the workstation.

The structure is not deleted from the CSS and may again be posted to the same or another workstation. Other workstations are unaffected.

If structure identifier is not listed in the table of posted structures or does not exist in the CSS when UNPOST STRUCTURE is called, the function takes no action.

The effect of unposting the structure may be immediately visible on the workstation's display, depending on the workstation's display update state.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (i.e., the workstation category is neither OUTPUT, OUTIN, nor MO)

**SEE ALSO**

POST STRUCTURE (3P)
INQUIRE POSTED STRUCTURES (3P)
INQUIRE SET OF WORKSTATIONS TO WHICH POSTED (3P)
UNPOST ALL STRUCTURES (3P)

**NAME**

UPDATE WORKSTATION – execute deferred workstation actions and optionally correct the display

**SYNOPSIS**

**C Syntax**

```
void
pupd_ws ( ws, regen_flag )
Pint            ws;            workstation identifier
Pregen_flag     regen_flag;    when to do the regeneration
```

**Required PHIGS Operating States**

(PHOP, WSOP, ∗, ∗ )

**DESCRIPTION**

**Purpose**

UPDATE WORKSTATION completes any actions in progress on the specified workstation. Then, depending on the value of the *regeneration flag* and the correctness of the workstation's display, the display is regenerated to make it PVISUAL_ST_CORRECT.

**C Input Parameters**

*ws* The identifier of the workstation to be updated.

*regen_flag*

The regeneration flag controls whether the display should be made entirely correct. Valid values are:

| | |
|---|---|
| PFLAG_PERFORM | *Make picture PVISUAL_ST_CORRECT if it is not* |
| PFLAG_POSTPONE | *Postpone regeneration; only complete deferred workstation actions* |

**Execution**

UPDATE WORKSTATION always initiates transmission of any data buffered from previous actions on the specified workstation, without first clearing the display surface.

Then UPDATE WORKSTATION may regenerate the display, depending on the value of the *regeneration flag* parameter and the correctness of the workstation's display. If *regeneration flag* is set to PFLAG_PERFORM, and the workstation's display is not entirely PVISUAL_ST_CORRECT, UPDATE WORKSTATION regenerates the display, performing the following actions:

1. If the workstation's display is not empty, then the display surface is cleared.

2. For every view representation in the View workstation state list, if the *view transformation update state* is PUPD_PEND, the current view representation is loaded from the *requested* view representation, and the update state is set to PUPD_NOT_PEND.

3. If the *workstation transformation update state* is PUPD_PEND, the *current workstation window*, and *current workstation viewport* are loaded with the *requested* values for each, and the update state is set to PUPD_NOT_PEND.

4. If the workstation's *HLHSR update state* is PUPD_PEND, the *current HLHSR mode* is updated to the value of the *requested HLHSR mode*; and the update state is set to PUPD_NOT_PEND.

5. All structure networks posted for this workstation are redisplayed in their priority order.

6. The workstation's *state of visual representation* is set to PVISUAL_ST_CORRECT.

The workstation's state of visual representation indicates if the display is PVISUAL_ST_CORRECT, PVISUAL_ST_SIMULATED, or PVISUAL_ST_DEFER. INQUIRE DISPLAY UPDATE STATE returns the workstation's *display surface empty* and state of visual representation state list entries. Normally, the traversal in step 5 causes the display surface empty workstation state to become PSTAT_NOTEMPTY. (If all the posted

structures are empty, or contain elements but no output primitive elements, the display surface empty may be PSTRUCT_STATUS_EMPTY or PSTAT_NOTEMPTY.)

UPDATE WORKSTATION performs the same sequence of actions as REDRAW ALL STRUCTURES when:

- *regeneration flag* is PFLAG_PERFORM.
- The workstation's state of visual representation is PVISUAL_ST_DEFER or PVISUAL_ST_SIMULATED.

**ERRORS**

003 Ignoring function, function requires state (PHOP, WSOP, *, *)

054 Ignoring function, the specified workstation is not open

059 Ignoring function, the specified workstation does not have output capability (that is, the workstation category is neither OUTPUT, OUTIN, nor MO)

**SEE ALSO**

INQUIRE DISPLAY UPDATE STATE (3P)
SET DISPLAY UPDATE STATE (3P)
REDRAW ALL STRUCTURES (3P)

**NAME**

WORKSTATION TYPE CREATE – create a new workstation type — PEX-SI extension

**SYNOPSIS**

**C Syntax**

```
Pint
phigs_ws_type_create ( base, attributes)
Pint                base;                    base workstation type
Phigs_ws_type_attr  attributes;
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

WORKSTATION TYPE CREATE creates a new workstation type. The new type is a copy of an existing type, possibly changed by any attribute-value pairs specified by the second parameter.

A new workstation type is created by an application to obtain a type with different workstation description table values from those of an existing workstation type. The workstation description table values can be changed by specifying the desired values to this function or by a subsequent call to WORKSTATION TYPE SET, passing it the workstation type created by WORKSTATION TYPE CREATE.

The workstation type returned from this function can be used anywhere a workstation type parameter is specified. When created, it is added to the list of available workstation types in the PHIGS description table.

This is a PEX-SI addition, and is not part of the PHIGS standard.

**C Input Parameters**

*base* An existing workstation type from which to build the new type.

*attributes*

A 0 terminated list of workstation type attributes and corresponding values. As described in WORKSTATION TYPE SET, the workstation description table fields of the new workstation type are modified according to the contents of this list.

**C Output Parameters**

If this function is successful, the new workstation type is returned as the value of this function.

**Execution**

The third argument to OPEN WORKSTATION is a *workstation type*. Each workstation type has a *workstation description table* associated with it, which contains both the PHIGS-specified data describing workstation capabilities and some PEX-SI-specific data describing workstation characteristics. The PEX-SI-specific data, as well as some of the PHIGS data, can be changed by the application prior to opening a workstation. WORKSTATION TYPE SET describes the data that can be changed and how to change it.

PEX-SI supplies a number of predefined workstation types. These are described in OPEN WORKSTATION. The predefined types cannot be operated on by WORKSTATION TYPE SET, their description table values can not be changed. Instead, it is necessary to create a new workstation type, based on one of the predefined types, which can be modified by WORKSTATION TYPE SET. Once created, the new workstation type can be used anywhere that PHIGS calls for a workstation type, including the *base type* parameter of this function. The new type can be repeatedly edited by WORKSTATION TYPE SET and passed to OPEN WORKSTATION This could be done, for instance, to open workstations with only slightly different description table values. When the type is no longer needed it should be destroyed with WORKSTATION TYPE DESTROY.

If a call to this function is successful the new workstation type will be added to the *list of available workstation types* in the PHIGS description table.

**ERRORS**

No Error

**SEE ALSO**

WORKSTATION TYPE DESTROY (3P)
WORKSTATION TYPE GET (3P)
WORKSTATION TYPE SET (3P)
OPEN WORKSTATION (3P)
INQUIRE LIST OF AVAILABLE WORKSTATION TYPES (3P)
INQUIRE WORKSTATION CONNECTION AND TYPE (3P)
PHIGS WORKSTATION DESCRIPTION TABLE (7P)

**NAME**

WORKSTATION TYPE DESTROY – destroy a workstation type — PEX-SI extension

**SYNOPSIS**

**C Syntax**

```
void
phigs_ws_type_destroy ( wst )
Pint            wst;            workstation type
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

WORKSTATION TYPE DESTROY destroys a workstation type created by the application. Any memory used by the workstation type is released and the workstation type is removed from the *list of available workstation types* in the PHIGS description table. After a workstation type is destroyed it can no longer be used. See WORKSTATION TYPE CREATE for a description of how and why to create a new workstation type.

**C Input Parameter**

*wst* The workstation type to destroy.

**ERRORS**

No Error

**SEE ALSO**

WORKSTATION TYPE CREATE (3P)
WORKSTATION TYPE GET (3P)
WORKSTATION TYPE SET (3P)
OPEN WORKSTATION (3P)
INQUIRE LIST OF AVAILABLE WORKSTATION TYPES (3P)

**NAME**

WORKSTATION TYPE GET – return the value of a specified workstation type attribute – PEX-SI Extension

**SYNOPSIS**

**C Syntax**

```
caddr_t
phigs_ws_type_get ( wst, attr, arg )
Pint                wst;          workstation type
Phigs_ws_type_attr  attr;
char                *arg
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

WORKSTATION TYPE GET retrieves the values of certain workstation type attributes. The attribute value to return is specified by the *attribute* parameter. See WORKSTATION TYPE CREATE and WORKSTATION TYPE SET for more information.

This is a PEX-SI addition, and is not part of the PHIGS standard.

**C Input Parameters**

*wst* The workstation type in question.

*attr* The attribute indicating the value to return. See WORKSTATION TYPE SET for a list of attributes defined, their descriptions, and the data types of their values.

*arg* Needed for som attributes to specify additional information.

**C Output Parameters**

This function returns the requested information as the value of the function. The return value must be cast to the appropriate type when accessing the returned information.

**ERRORS**

No Error

**SEE ALSO**

WORKSTATION TYPE SET (3P)
PHIGS WORKSTATION DESCRIPTION TABLE (7P)
INQUIRE LIST OF AVAILABLE WORKSTATION TYPES (3P)
INQUIRE WORKSTATION CONNECTION AND TYPE (3P)

**NAME**

WORKSTATION TYPE SET – set workstation type attributes — PEX-SI Extension

**SYNOPSIS**

**C Syntax**

```
caddr_t
phigs_ws_type_set ( wst, <attribute-value list> )
Pint                wst;                    workstation type
Phigs_ws_type_attr  attributes;             attributes
```

**Required PHIGS Operating States**

(PHOP, *, *, *)

**DESCRIPTION**

**Purpose**

WORKSTATION TYPE SET changes the specified attribute values in the workstation description table associated with the specified workstation type.

The workstation description table associated with each workstation type contains both PHIGS-defined fields and PEX-SI-specific fields. All of the PEX-SI-specific fields and some of the PHIGS-specific fields can be changed with WORKSTATION TYPE SET. The fields that can be modified are listed below in the *Workstation Type Attributes* section.

WORKSTATION TYPE SET operates only on workstation types created by WORKSTATION TYPE CREATE. The predefined types cannot be modified, nor can types returned by INQUIRE WORKSTATION CONNECTION AND TYPE. See OPEN WORKSTATION for a description of the predefined workstation types.

This is a PEX-SI addition, and is not part of the PHIGS standard.

**C Input Parameters**

*wst* The workstation type to modify. This cannot be a specific workstation type (one associated with an open workstation) or one of the predefined types.

*<attribute-value list>*

A zero terminated list of workstation type attributes and corresponding values. The attributes are members of the Phigs_ws_type_attr enumeration defined in phigs.h. The data type of the values varies according to the attribute. The list of attributes below specifies the value type required for each attribute. The attribute fields of the workstation type are modified according to the contents of this list.

Phigs_ws_type_attr is defined as follows:

```
typedef enum {
        /* generic attributes start at 1 */
        PHG_BASE_NAME = PHIGS_ATTR( PHG_ATTR_ENUM, PHIGS_FIRST_ATTR + 5),
#define  PHIGS_BASE_NAME  (char *)PHG_BASE_NAME
        /* generic X attributes start at 25 */
        /* PHIGS_X_DISPLAY can only be set.  Only the name is remembered. */
        PHG_X_DISPLAY = PHIGS_ATTR( PHG_ATTR_STRING, PHIGS_FIRST_ATTR + 25),
#define PHIGS_X_DISPLAY (char *)PHG_X_DISPLAY
        PHG_X_DISPLAY_NAME = PHIGS_ATTR( PHG_ATTR_STRING, PHIGS_FIRST_ATTR + 26),
#define PHIGS_X_DISPLAY_NAME (char *)PHG_X_DISPLAY_NAME
        PHG_X_DISPLAY_WINDOW = PHIGS_ATTR( PHG_ATTR_INT_PAIR, PHIGS_FIRST_ATTR + 27),
#define PHIGS_X_DISPLAY_WINDOW (char *)PHG_X_DISPLAY_WINDOW
        /* PHIGS_X_RMDB can only be set. */
        PHG_X_RMDB = PHIGS_ATTR( PHG_ATTR_INT_TRIPLE, PHIGS_FIRST_ATTR + 28),
#define PHIGS_X_RMDB (char *)PHG_X_RMDB
        PHG_X_BUF_MODE = PHIGS_ATTR( PHG_ATTR_INT, PHIGS_FIRST_ATTR + 29),
#define PHIGS_X_BUF_MODE (char *)PHG_X_BUF_MODE
```

```
        PHG_X_HANDLE_EXPOSE = PHIGS_ATTR( PHG_ATTR_BOOLEAN, PHIGS_FIRST_ATTR + 31),
#define PHIGS_X_HANDLE_EXPOSE (char *)PHG_X_HANDLE_EXPOSE
        PHG_X_HANDLE_DESTROY = PHIGS_ATTR( PHG_ATTR_BOOLEAN, PHIGS_FIRST_ATTR + 32),
#define PHIGS_X_HANDLE_DESTROY (char *)PHG_X_HANDLE_DESTROY
        PHG_DC_MODEL = PHIGS_ATTR( PHG_ATTR_ENUM, PHIGS_FIRST_ATTR + 33),
#define PHIGS_DC_MODEL  (char *)PHG_DC_MODEL
        PHG_X_CMAP_PROP_ATOM = PHIGS_ATTR( PHG_ATTR_INT, PHIGS_FIRST_ATTR + 34),
#define PHIGS_X_CMAP_PROP_ATOM (char *)PHG_X_CMAP_PROP_ATOM

        /* tool attributes start at 50 */
        PHG_TOOL_WIDTH = PHIGS_ATTR( PHG_ATTR_INT, PHIGS_FIRST_ATTR + 50),
#define PHIGS_TOOL_WIDTH (char *)PHG_TOOL_WIDTH
        PHG_TOOL_HEIGHT = PHIGS_ATTR( PHG_ATTR_INT, PHIGS_FIRST_ATTR + 51),
#define PHIGS_TOOL_HEIGHT (char *)PHG_TOOL_HEIGHT
        PHG_TOOL_X = PHIGS_ATTR( PHG_ATTR_INT, PHIGS_FIRST_ATTR + 52),
#define PHIGS_TOOL_X (char *)PHG_TOOL_X
        PHG_TOOL_Y = PHIGS_ATTR( PHG_ATTR_INT, PHIGS_FIRST_ATTR + 53),
#define PHIGS_TOOL_Y (char *)PHG_TOOL_Y
        PHG_TOOL_LABEL = PHIGS_ATTR( PHG_ATTR_STRING, PHIGS_FIRST_ATTR + 54),
#define PHIGS_TOOL_LABEL (char *)PHG_TOOL_LABEL
        PHG_TOOL_ICON_NAME = PHIGS_ATTR( PHG_ATTR_STRING, PHIGS_FIRST_ATTR + 55),
#define PHIGS_TOOL_ICON_NAME (char *)PHG_TOOL_ICON_NAME
        PHG_TOOL_BORDER_WIDTH = PHIGS_ATTR( PHG_ATTR_INT, PHIGS_FIRST_ATTR + 56),
#define PHIGS_TOOL_BORDER_WIDTH (char *)PHG_TOOL_BORDER_WIDTH

        /* Phigs attributes start at 150 */
        PHG_WS_CATEGORY = PHIGS_ATTR( PHG_ATTR_ENUM, PHIGS_FIRST_ATTR + 150),
#define PHIGS_WS_CATEGORY (char *)PHG_WS_CATEGORY
} Phigs_ws_type_attr;
```

The following is an example of an attribute list:

```
Pint            wst;

phigs_ws_type_set( wst,
                PHIGS_TOOL_X, (char*)700,
                PHIGS_TOOL_Y, (char*)100,
                PHIGS_TOOL_WIDTH, (char*)200,
                PHIGS_TOOL_HEIGHT, (char*)200,
                PHIGS_TOOL_LABEL, "views",
                (char*)0 );
```

**Execution**

WORKSTATION TYPE SET modifies the fields of the workstation description table associated with the specified workstation type. The modifiable fields are listed below. The changes are specified with *attribute-value* pairs. The *attribute* specifies the field to change, the *value* is the value to assign to the field. The PEX-SI-specific fields are different for different workstation types, so the attributes are listed according to the workstation type to which they apply.

**Workstation Type Attributes**

The workstation description table attributes are described here. The C constants are listed along with a description of each attribute, the data type required, and the default value. The C constants are members of the Phigs_ws_type_attr enumerated type defined in phigs.h.

**Common x_tool and x_drawable Attributes**

*PHIGS_BASE_NAME* (C)

This argument to WORKSTATION TYPE GET retrieves the base type of the workstation type. It can not be specified as an argument to WORKSTATION TYPE SET or WORKSTATION TYPE CREATE.

C enumeration: *PHIGS_X_TOOL, PHIGS_X_DRAWABLE*

*PHIGS_WS_CATEGORY* (C)

The category of the workstation. This attribute can be used to change the workstation category from OUTIN (output and input) to OUTPUT (output only).

Data type: *Pwscat (PCAT_OUT or PCAT_OUTIN only)* (C)

Default: OUTIN

C enumeration: *PCAT_OUT, PCAT_IN, PCAT_OUTIN, PCAT_MO, PCAT_MI*

*PHIGS_X_BUF_MODE* (C)

Specifies the method of double buffering to use.

Data type: *Phigs_dbl_buff* (C)

Default: PHIGS_BUF_SINGLE

C enumeration: *PHIGS_BUF_SINGLE, PHIGS_BUF_DOUBLE*

*PHIGS_X_DISPLAY* (C)

Modify the workstation type to reflect the capabilities of the specified display. The display is specified by a display pointer. Some of the workstation type attributes and workstation description table values are dependent on the capabilities of the PEX extension in the X server. This attribute causes such values to be modified to reflect those capabilities. This attribute can only be specified to WORKSTATION TYPE SET or WORKSTATION TYPE CREATE. It cannot be specified to WORKSTATION TYPE GET.

Data type: *Display** (C)

Default: NULL

*PHIGS_X_DISPLAY_NAME* (C)

Modify the workstation type to reflect the capabilities of the specified display. The display is specified by a display name string. Some of the workstation type attributes and workstation description table values are dependent on the capabilities of the PEX extension in the X server. This attribute causes such values to be modified to reflect those capabilities.

Data type: *char** (C)

Default: NULL

*PHIGS_X_DISPLAY_WINDOW* (C)

Modify the workstation type to reflect the capabilities of the specified window. Some of the workstation type attributes and workstation description table values are dependent on the capabilities of the X window and PEX extension the PHIGS workstation will use. This attribute causes such values to be modified to reflect those capabilities. This attribute can only be specified to WORKSTATION TYPE SET of WORKSTATION TYPE CREATE. It cannot be specified to WORKSTATION TYPE GET.

Data type: *Display**, *Window* (C)

Default: NULL, 0

*PHIGS_HANDLE_EXPOSE* (C)

Enable or disable PEX-SI handling of window exposures. A non-zero value specifies that PHIGS will redraw the PHIGS output when the window or portions of the window are exposed. A zero value specifies that PHIGS will not redraw when the window or portions of the window are exposed.

Data type: *Pint* (C)

Default: 1

*PHIGS_HANDLE_DESTROY* (C)

Enable or disable handling of DestroyNotify events from the workstation window. A non-zero value specifies that the PEX-SI API will exit when a DestroyNotify event is received on the window of an open workstation. A zero value specifies that the PEX-SI API will do nothing when a DestroyNotify event is received on the window of an open workstation.

Data type: *Pint* (C)

Default: 1

*PHIGS_DC_MODEL* (C)

Set the effect of window resizing on the workstation's DC limits. Valid values are defined in phigs.h:

```
typedef enum {
        PHIGS_DC_LIMITS_FIXED,
        PHIGS_DC_LIMITS_ADJUST_TO_WINDOW,
} Phigs_DC_model;
```

DC_LIMITS_FIXED causes the DC limits to be left as they are, even though the new window size is possibly smaller or larger than the limits.

DC_LIMITS_ADJUST_TO_WINDOW, the default, causes the DC limits to be adjusted to the new window size.

Data type: *Phigs_DC_model* (C)

Default: PHIGS_DC_LIMITS_FIXED

**x_tool Attributes**

*PHIGS_TOOL_WIDTH* (C)

*PHIGS_TOOL_HEIGHT* (C)

The initial size of the graphics window in screen coordinates.

Data type: *Pint* (C)

Default: 600, 600

*PHIGS_TOOL_X* (C)

*PHIGS_TOOL_Y* (C)

The initial position of the workstation in screen coordinates.

Data type: *Pint* (C)

Default: 50, 50

*PHIGS_TOOL_LABEL* (C)

The label to put in the workstation frame banner.

Data type: *char** (C)

Default: PHIGS Workstation

*PHIGS_TOOL_ICON_LABEL* (C)

The label for the icon when the workstation window is closed (iconified).

Data type: *char** (C)

Default:

*PHIGS_TOOL_BORDER_WIDTH* (C)

The border width of the window.

Data type: *Pint* (C)

Default: 0

**ERRORS**

No Error

**SEE ALSO**

WORKSTATION TYPE CREATE (3P)
WORKSTATION TYPE DESTROY (3P)
WORKSTATION TYPE GET (3P)
OPEN WORKSTATION (3P)
INQUIRE LIST OF AVAILABLE WORKSTATION TYPES (3P)
INQUIRE WORKSTATION CONNECTION AND TYPE (3P)

ESCAPE -3 (3P)
PHIGS WORKSTATION DESCRIPTION TABLE (7P)

**NAME**

COLOUR – introduction to using colour in the PEX-SI

**DESCRIPTION**

This page provides an overview of using the capabilities of a colour system from the PHIGS API to PEX. The topics covered are:

- Standard PHIGS: Indexed Colour
- PHIGS+: General Colour
- Setting a Workstation's Colour Table
- How Colour Maps Are Set Up

**Standard PHIGS: Indexed Colour**

Standard PHIGS supports only *indexed colour*. Structure elements select a colour for a primitive by using an integer colour index, such as that selected by SET POLYLINE COLOUR INDEX(3P). When a primitive is displayed on a workstation, the colour index is used to select a colour value (called a colour representation) from the workstation's colour table. The colour representation is what is actually seen by the observer. The default colour index for all primitives is 1.

**PHIGS PLUS: General Colour**

PHIGS PLUS allows the specification of colour either as indexed colours, or as RGB colours directly. When a colour is specified as an index, it is called *indirect colour*, since the colours are indirectly specified through the workstation colour table. When a colour is specified as an RGB triple, it is called *direct colour*. General colour is an abstraction that allows colour or direct colour to be specified as a single aspect. A general colour consists of a type and a value. If the type is indirect, then the value is interpreted as an index into the workstation colour table. If the type RGB, then the value is interpreted as an RGB value. In addition to RGB type, PHIGS+ allows specification of direct color in other types, for example HLS, HSV and CIELUV. In the PEX-SI, however, only the type RGB is allowed.

Using the PEX-SI primitives and attributes, the colour for a primitive can be set using either attribute elements, such as SET POLYLINE COLOUR (3P+), or primitives with one or more colours as part of the primitive, such as POLYLINE SET 3 WITH DATA (3P+). In either case, the colour is specified as a general colour, so that an index or direct colour can be used. This functionality supercedes the standard PHIGS functionality. Standard PHIGS primitives will be rendered using the PHIGS+ attributes, and standard PHIGS attributes affect PHIGS+ primitives. For example, SET POLYLINE COLOUR INDEX(3P) is equivalent to calling SET POLYLINE COLOUR(3P+) with the type set to indirect, both elements set the same attribute. The default colour for all primitives is indirect colour, index 1. If a primitive or attribute specifies direct colour on an indexed colour workstation, then *indirect colour, index 1* is used instead.

**Setting a Workstation's Colour Table**

In PHIGS, an application can use the function SET COLOUR REPRESENTATION to specify a colour representation by a triple of real values whose meaning depends on the current colour model. PEX-SI supports only the RGB colour model. The colour coordinate space is [0,1] in all three dimensions. No intensity is 0.0 and is mapped to X11 colour value 0. Maximum intensity is 1.0 and is mapped to X11 colour value 65535. The RGB values for the default colour definitions are given above.

A change to a colour representation will show up during the next regeneration.

A PHIGS application can inquire for any colour index, either the value it sets into the workstation's colour table or the values that will be *Realized* on the display, after mapping to available X11 colour values. See INQUIRE COLOUR REPRESENTATION (3P).

Other colour models PHIGS+ defines are CIE (*Commission Internationale de l'Eclairage's* universal colour definition system), HSV (Hue, Saturation, and Value), and HLS (Hue, Lightness, and Saturation).

Colour representations 0 and 1 can be set on a monochrome workstation, to produce either normal or inverse video.

**How Colormaps Are Set Up**

The API tries to manage PEX colour and colour approximation lookup tables and X colormaps when implementing the PHIGS colourmap. Ideally, you should take the following steps before creating a window:

1. Find the most appropriate visual.
2. See if there is a StandardColormap property defined on that visual.
3. Set up the colormap on that visual or use an existing colormap.

However, X doesn't provide a mechanism by which StandardColormap properties can be queried on an individual visual class if a window on that visual doesn't exist. Therefore, heursitics are used heavily to determine this. The following process takes place.

1. If the workstation is of type *x_tool*, determine the ''best'' visual for PEX and open the window on that visual. The best visual is the avaiable visual with the largest colormap size and the highest numbered class (DirectColor > TrueColor > PseudoColor > StaticColor). As an attempt to cut down on colormap flashing, the default visual is used if it has a StandardColormap with at least the same number of entries as the best visual.

   If the workstation is of type *x_drawable*, it is the application's responsibility to determine the most appropriate visual and open up the window on that visual.

   An error is returned if a read-only visual with no StandardColormaps defined is available.

2. The StandardColormap's colorcube is used if it is large enough to hold a 5/9/5 colorcube for an 8-bit display. The characteristics of the StandardColormap are set in the workstation's ColourApprox lookup table.

   If there is no StandardColormap defined, an attempt is made to define a contiguous colormap in the existing colormap using *XallocColorCells()*. If this fails, than a new colormap is created. In either case, the colormap is filled with the appropriate entries and the ColorApprox lookup table is set. The colorcube is set up to use about the same percentage of the colormap that a 5/9/5 colorcube uses of a 256 entry colormap (about 85%). For example, a 7/14/7 colorcube is used in a colormap which has 1024 entries.

**SEE ALSO**

INQUIRE COLOUR FACILITIES (3P)
INQUIRE COLOUR MODEL FACILITIES (3P)
INQUIRE DIRECT COLOUR MODEL FACILITIES (3P+)
SET COLOUR MODEL (3P)
SET COLOUR REPRESENTATION (3P)
INQUIRE COLOUR REPRESENTATION (3P)
INQUIRE PREDEFINED COLOUR REPRESENTATION (3P)
SET POLYLINE COLOUR INDEX (3P)
SET POLYLINE COLOUR (3P+)

**BUGS**

The PHIGS standard uses the "international" (in other words, British) spelling of "colour." X11 and American usage spell "colormap" without the "u."

**NAME**

PHIGS DESCRIPTION TABLE – contains information describing the static capabilities of PEX-SI

**DESCRIPTION**

The PHIGS Description Table is created when PHIGS is opened and exists until PHIGS is closed. The PHIGS standard includes the default values for the pipeline context in the logical PHIGS Description Table.

The remaining values are limits and capabilities. These may be inquired.

*Key to Data Type Abbreviations*

| | | | |
|---|---|---|---|
| I | Integer | W | Workstation Type |
| E | Enumeration Type | L(type) | List of Values of a Given Type |

*Key Denoting the Source of Information*

API determined by API, hardcoded

ETI determined by server, API queries with Get Enumerated Type Information

IDC determined by server,API queries with Get Implementation-Dependent Constants

**PHIGS Description Table**

| *Description Table Entry* | *Data Type* | *Initial Value* |
|---|---|---|
| number of available workstation types | I | 2 |
| list of available workstation types | L(W) | see Table 1 |
| max number of simultaneously open workstations | I | API(20) |
| max number of simultaneously open archive files | I | API(25) |
| number of available names for name sets | I | IDC |
| available character sets | I | API(1,ASCII) |
| max length of normal filter list for ISS | I | IDC |
| max length of inverted filter list for ISS | I | IDC |
| available generalized structure elements | I | ETI |
| max number of distinct planes in modelling clip volumes | I | IDC |
| number of available modelling clip operators | I | ETI |

**Table 1 — Workstation Type and Category**

| *Type* | *Category* |
|---|---|
| x_tool | OUTIN |
| x_drawable | OUTIN |

**SEE ALSO**

PHIGS WORKSTATION DESCRIPTION TABLE (7P)

**NAME**

PHIGS TRAVERSAL STATE LIST – contains all transient information PEX-SI uses during traversal

**DESCRIPTION**

The PHIGS Traversal State List is created whenever a traversal is initiated, and only exists during the traversal. Hence, its values cannot be inquired. Traversal is the execution of a structure network. Traversal begins with a structure posted to a workstation. Structure elements are traversed sequentially. The traversal state list is modified by executing attribute elements.

The initial values in the traversal state list are either defined by the PHIGS standard, or are implementation dependent. These values are inherited by a posted structure. Whenever an EXECUTE STRUCTURE element is traversed, the executed structure, or *child* structure, *inherits* the attributes of the *parent* structure; that is, the structure invoking (executing or referencing) it. Different parent structures can assign different attribute values.

The *child* structure's attributes are then modified by attribute elements as they are sequentially executed. The child structure may also execute subordinate structures, at which point the calling structure is considered the parent of this generation. At the end of the child structure (i.e., after its last element), traversal returns to the parent structure, restoring the saved state of the parent structure, and continues until the end of the parent structure. This continues until the entire structure network is traversed (i.e., until traversal of the posted structure is complete). The modifications to the child's structure do not affect the parent structure's attributes. Traversal state list values may be said to *expire* at the end of the structure in which they are set, as traversal of the parent structure resumes.

The values in the traversal state list are *bound* to primitives as the primitive elements are sequentially executed.

**Data Type Abbreviations**

| | | | |
|---|---|---|---|
| I | Integer | GCOLR | General Colour† |
| R | Real | V2/V3 | 2D/3D Vector |
| E | Enumeration Type | P3 | 3D Point |
| L(type) | List of values of a given type | SET(NM) | Set of Eligible Names |
| MCV | Modelling Clipping Volume | (nys) | Not Yet Supported |

**PEX-SI Traversal State List**

| *Traversal State List Entry* | *Data Type* | *Default or Initial Value* |
|---|---|---|
| polyline index | I | 1 |
| linetype | I | 1 |
| linewidth scale factor | R | 1.0 |
| polyline colour index | I | 1 |
| polyline colour† | GCOLR | (INDIRECT,1) |
| linetype ASF | E | INDIVIDUAL |
| linewidth scale factor ASF | E | INDIVIDUAL |
| polyline colour index ASF | E | INDIVIDUAL |
| | | |
| polyline shading method† | I | 1 (NONE) |
| curve approximation criteria type† | I | 1 (Workstation Dependent) |
| curve approximation criteria value† | R | 1.0 |
| polyline shading method ASF† | E | INDIVIDUAL |
| curve approximation criteria ASF† | E | INDIVIDUAL |

† This is a PHIGS Extension ***based on PHIGS PLUS and is not a part of the PHIGS standard.***

**PHIGS Traversal State List (continued)**

| *Traversal State List Entry* | *Data Type* | *Default or Initial Value* |
|---|---|---|
| polymarker index | I | 1 |
| marker type | I | 3 |
| marker size scale factor | R | 1.0 |
| polymarker colour index | I | 1 |
| polymarker colour† | GCOLR | (INDIRECT,1) |
| marker type ASF | E | INDIVIDUAL |
| marker scale factor ASF | E | INDIVIDUAL |
| polymarker colour index ASF | E | INDIVIDUAL |
| | | |
| text index | I | 1 |
| text font | I | 1 (Monospaced Roman Simplex) |
| text precision | E | STRING |
| character expansion factor | R | 1.0 |
| character spacing | R | 0.0 |
| text colour index | I | 1 |
| text colour† | GCOLR | (INDIRECT,1) |
| text font ASF | E | INDIVIDUAL |
| text precision ASF | E | INDIVIDUAL |
| character expansion factor ASF | E | INDIVIDUAL |
| character spacing ASF | E | INDIVIDUAL |
| text colour index ASF | E | INDIVIDUAL |
| | | |
| character height | R | 0.01 |
| character up vector | V2 | (0.0,1.0) |
| character width | R | 0.01 |
| character base vector | V2 | (1.0,0.0) |
| text path | E | RIGHT |
| text alignment (horz & vert) | 2×E | (NORMAL,NORMAL) |
| | | |
| annotation text character height | R | 0.01 |
| annotation text character up vector | V2 | (0.0,1.0) |
| annotation text character width | R | 0.01 |
| annotation text character base vector | V2 | (1.0,0.0) |
| annotation text path | E | RIGHT |
| annotation text alignment (horz & vert) | 2×E | (NORMAL,NORMAL) |
| annotation style | I | 1 (unconnected) |

† This is a *PHIGS Extension* based on PHIGS PLUS and is not a part of the PHIGS standard.

**PHIGS Traversal State List (continued)**

| *Traversal State List Entry* | *Data Type* | *Default or Initial Value* |
|---|---|---|
| interior index | I | 1 |
| interior style | E | HOLLOW |
| back interior style† | E | HOLLOW |
| interior style index | I | 1 |
| back interior style index† | I | 1 |
| interior colour index | I | 1 |
| interior colour† | GCOLR | (INDIRECT,1) |
| back interior colour† | GCOLR | (INDIRECT,1) |
| interior style ASF | E | INDIVIDUAL |
| back interior style ASF† | E | INDIVIDUAL |
| interior style index ASF | E | INDIVIDUAL |
| back interior style index ASF† | E | INDIVIDUAL |
| interior colour index ASF | E | INDIVIDUAL |
| back interior colour ASF† | E | INDIVIDUAL |
| | | |
| ambient reflection coefficient† | R | 1.0 |
| back ambient reflection coefficient† | R | 1.0 |
| diffuse reflection coefficient† | R | 1.0 |
| back diffuse reflection coefficient† | R | 1.0 |
| specular reflection coefficient† | R | 1.0 |
| back specular reflection coefficient† | R | 1.0 |
| specular colour† | GCOLR | (INDIRECT,1) |
| back specular colour† | GCOLR | (INDIRECT,1) |
| specular exponent† | R | 0.0 |
| back specular exponent† | R | 0.0 |
| interior shading method† | I | 1 (NONE) |
| back interior shading method† | I | 1 (NONE) |
| interior reflectance equation† | I | 1 (NONE) |
| back interior reflectance equation† | I | 1 (NONE) |
| area properties ASF† | E | INDIVIDUAL |
| back area properties ASF† | E | INDIVIDUAL |
| interior shading method ASF† | E | INDIVIDUAL |
| back interior shading method ASF† | E | INDIVIDUAL |
| interior reflectance equation ASF† | E | INDIVIDUAL |
| back interior reflectance equation ASF† | E | INDIVIDUAL |
| | | |
| surface approximation criteria type† | I | 1 (Workstation Dependent) |
| surface approximation criteria value† | 2×R | (1.0,1.0) |
| surface approximation criteria ASF† | E | INDIVIDUAL |
| | | |
| face distinguishing mode† | E | NO |
| face culling mode† | E | NONE |

† This is a *PHIGS Extension* based on PHIGS PLUS and is not a part of the PHIGS standard.

**PHIGS Traversal State List (continued)**

| *Traversal State List Entry* | *Data Type* | *Default or Initial Value* |
|---|---|---|
| depth cue index† | I | 0 |
| colour mapping index† | I | 0 |
| rendering colour model† | I | 0 |
| light source state† | L(I) | empty |
| edge index | I | 1 |
| edge flag | E | OFF |
| edgetype | I | 1 |
| edgewidth scale factor | R | 1.0 |
| edge colour index | I | 1 |
| edge colour† | GCOLR | (INDIRECT,1) |
| edge flag ASF | E | INDIVIDUAL |
| edgetype ASF | E | INDIVIDUAL |
| edgewidth scale factor ASF | E | INDIVIDUAL |
| edge colour index ASF | E | INDIVIDUAL |
| pattern size | 2×R | (1.0,1.0) |
| pattern reference point | P3 | (0.0,0.0,0.0) |
| pattern reference vectors | 2×V3 | (1.0,0.0,0.0) (0.0,1.0,0.0) |
| pick identifier | I | 0 |
| view index | I | 0 |
| HLHSR identifier | I | 0 |
| name set | SET(NM) | no classes (empty set) |
| global modelling transformation | 4×4×R | Identity |
| local modelling transformation | 4×4×R | Identity |
| modelling clipping volume | MCV | all of WC |
| modelling clipping indicator | E | NOCLIP |

† This is a *PHIGS Extension* based on PHIGS PLUS and is not a part of the PHIGS standard.

‡ This is a PHIGS-specific item of traversal state. See the appropriate GSE man page.

∗ Zero is workstation dependent.

**SEE ALSO**

PHIGS DESCRIPTION TABLE (7P)
EXECUTE STRUCTURE (3P)
INQUIRE ANCESTORS OF STRUCTURE (3P)
INQUIRE DESCENDANTS OF STRUCTURE (3P)
ELEMENT SEARCH (3P)

**NAME**

PHIGS WORKSTATION DESCRIPTION TABLE – contains information describing the capabilities of a PEX-SI PHIGS workstation

**Key to Data Type Abbreviations**

| | | | |
|---|---|---|---|
| I | Integer | P3 | 3D Point |
| R | Real | FP | Font/Precision Pair |
| E | Enumeration Type | CC | Chromaticity Coefficient |
| B | Bounding Range | D | Data Record |
| L(type) | List of values of a given type | (nys) | Not Yet Supported |

*Key Denoting the Source of Information:*

| | |
|---|---|
| API | determined by API, hardcoded |
| API+ | determined by API, using X or other information |
| ETI | determined by server, API queries the PEX extension with Get Enumerated Type Information |
| IDC | determined by server, API queries the PEX extension with Get Implementation-Dependent Constants |
| GTI | determined by server, API queries the PEX extension with Get Table Information |
| GPE | determined by server, API queries the PEX extension with Get Predefined Entries |
| GDY | determined by server, API queries the PEX extension with Get Dynamics |
| GWI | determined by server, API queries the PEX extension with Get PHIGS Workstation Information |
| LFI | List Fonts with Information |
| * | Partially-controlled by application |

**PEX-SI PHIGS Workstation Description Table**

| *Workstation Description Table Entry* | *Data Type* | *Initial Value* |
|---|---|---|
| workstation type | I | API* |
| workstation category | E | API*(PCAT_OUTIN) |
| device coordinate units | E | API(PDC_OTHER) |
| maximum display space size: in device coordinates | 3×R | API+* |
| number of available HLHSR identifiers | I | API (2) |
| list of available HLHSR identifiers | L(I) | API (OFF, ON) |
| number of available HLHSR modes | I | ETI |
| list of available HLHSR modes | L(I) | ETI |
| number of predefined view representations | I | 6 |
| table of predefined view representations | | See Table 1 |

*PHIGS Workstation Description Table — Continued*

| ***Workstation Description Table Entry*** | ***Data Type*** | ***Initial Value*** |
|---|---|---|
| workstation classification | E | API (RASTER) |
| dynamic modification flags: | | |
| view representation | E | GDY |
| polyline bundle representation | E | GDY |
| polymarker bundle representation | E | GDY |
| text bundle representation | E | GDY |
| interior bundle representation | E | GDY |
| edge bundle representation | E | GDY |
| pattern representation | E | GDY |
| colour representation | E | GDY |
| workstation transformation | E | GDY |
| highlighting filter | E | GDY |
| invisibility filter | E | GDY |
| HLHSR mode | E | GDY |
| structure content modifications | E | GDY |
| post structure | E | GDY |
| unpost structure | E | GDY |
| delete structure | E | GDY |
| reference modifications | E | GDY |
| default value for deferral state: | | |
| deferral mode | E | API (ASAP) |
| modification mode | E | API (UQUM) |
| number of available linetypes | I | ETI |
| list of available linetypes | L(I) | ETI |
| number of available linewidths | I | IDC |
| nominal linewidth | R | IDC |
| minimum linewidth | R | IDC |
| maximum linewidth | R | IDC |
| number of predefined polyline bundles | I | 5 |
| table of predefined polyline bundles | | See Table 2 |
| number of polyline shading methods supported | I | ETI |
| list of polyline shading methods supported | L(I) | ETI |
| number of curve approximation criteria types supported | I | ETI |
| list of curve approximation criteria supported | L(I) | ETI |
| number of available marker types | I | ETI |
| list of available marker types | L(I) | ETI |
| number of available marker sizes | I | IDC |
| nominal marker size | R | IDC |
| minimum marker size | R | IDC |
| maximum marker size | R | IDC |
| number of predefined polymarker bundles | I | 5 |
| table of predefined polymarker bundles | | See Table 3 |

*PHIGS Workstation Description Table — Continued*

| ***Workstation Description Table Entry*** | ***Data Type*** | ***Initial Value*** |
|---|---|---|
| number of text font and precision pairs | I | LFI |
| list of text font and precision pairs | L(FP) | LFI |
| number of available character expansion factors | I | API (0: continuous range) |
| minimum character expansion factor | R | API (0) |
| maximum character expansion factor | R | API (MAXFLOAT) |
| number of available character heights | I | API (0: continuous range) |
| minimum character height | R | API (0) |
| maximum character height | R | API (MAXFLOAT) |
| number of predefined text bundles | I | 6 |
| table of predefined text bundles | | See Table 4 |
| number of available annotation styles | I | ETI |
| list of available annotation styles | L(I) | ETI |
| number of available interior styles | I | ETI |
| list of available interior styles | L(E) | ETI |
| number of available hatch styles | I | ETI |
| list of available hatch styles | L(I) | ETI |
| number of predefined interior bundles | I | 5 |
| table of predefined interior bundles | | See Table 5 |
| number of interior shading methods | I | ETI |
| list of interior shading methods | L(I) | ETI |
| number of reflectance equations supported | I | ETI |
| list of reflectance equations supported | L(I) | ETI |
| number of surface approximation criteria supported | I | ETI |
| list of surface approximation criteria supported | L(I) | ETI |
| number of trimming curve approximation criteria types supported | I | ETI |
| list of trimming curve approximation criteria supported | L(I) | ETI |
| maximum trimming curve order supported | I | IDC |
| number of parametric surface characteristics supported | I | ETI |
| list of parametric surface characteristics | L(I) | ETI |
| number of predefined pattern representations | I | GTI |
| table of predefined pattern representations | | GPE |
| number of available colour models | I | ETI |
| list of available colour models | L(I) | ETI |
| number of available edgetypes | I | ETI |
| list of available edgetypes | L(I) | ETI |
| number of available edgewidths | I | IDC |
| nominal edgewidth | R | IDC |
| minimum edgewidth | R | IDC |
| maximum edgewidth | R | IDC |
| number of predefined edge bundles | I | 5 |
| table of predefined edge bundles | | See Table 6 |

*PHIGS Workstation Description Table — Continued*

| ***Workstation Description Table Entry*** | ***Data Type*** | ***Initial Value*** |
|---|---|---|
| default colour model | I | API (RGB) |
| chromaticity coefficients of display primaries | 3×CC | IDC |
| number of available colours or intensities | I | GTI |
| colour available | E | API+ |
| number of predefined colour representations | I | GTI |
| table of predefined colour representations | | GTI |
| number of available generalized drawing primitives 3 | I | ETI |
| list of available generalized drawing primitives 3 | L(I) | ETI |
| number of available generalized drawing primitives | I | ETI |
| list of available generalized drawing primitives | L(I) | ETI |
| number of available generalized structure elements | I | ETI |
| list of available generalized structure elements | L(I) | ETI |
| number of display priorities supported | I | GWI |
| maximum number of polyline bundle table entries | I | GTI |
| maximum number of polymarker bundle table entries | I | GTI |
| maximum number of text bundle table entries | I | GTI |
| maximum number of interior bundle table entries | I | GTI |
| maximum number of edge bundle table entries | I | GTI |
| maximum number of pattern table entries | I | GTI |
| maximum number of colour indices | I | GTI |
| maximum number of view indices | I | GTI |
| number of predefined depth cue indices | I | 2 |
| table of predefined depth cue operations | | See Table 7 |
| maximum number of depth cue bundle table entries | I | GTI |
| number of predefined colour mapping table entries | I | 1 |
| table of predefined colour mappings | | See Table 8 |
| maximum number of colour mapping table entries | I | GTI |
| number of colour mapping methods supported | I | ETI |
| list of colour mapping methods supported | | ETI |
| number of available true colours | I | API+ |
| number of available pseudo colours | I | API+ |
| number of directly specifiable colour models supported | I | ETI |
| number of rendering colour models supported | I | ETI |
| number of light source types supported | I | ETI |
| list of light source types supported | L(I) | ETI |
| number of predefined light source table indices | I | 2 |
| table of predefined light sources | | See Table 9 |
| maximum number of light source table entries | I | GTI |
| maximum number of simultaneously active non-ambient light sources | I | IDC |

*PHIGS Workstation Description Table — Continued*

| ***Workstation Description Table Entry*** | ***Data Type*** | ***Initial Value*** |
|---|---|---|
| number of logical devices of class LOCATOR | I | API (3) |
| locator device number | I | (1,2,3) |
| default initial locator position | P3 | (0,0,0) |
| number of available prompt and echo types | I | (1) |
| list of available prompt and echo types | L(I) | (1) |
| default echo volume | 3×B | (full window) |
| default locator data record | D | (empty) |
| number of logical input devices of class STROKE | I | API (1) |
| stroke device number | I | (1) |
| maximum input buffer size | I | (200) |
| number of available prompt and echo types | I | (1) |
| list of available prompt and echo types | L(I) | (1) |
| default echo volume | 3×B | (full window) |
| default stroke data record containing at least: | D | |
| input buffer size | I | (100) |
| editing position | I | (1) |
| x, y, z interval | I | (0) |
| time interval | I | (0) |
| number of logical input devices of class VALUATOR | I | API (1) |
| valuator device number | I | (1) |
| default initial value | R | (0) |
| number of available prompt and echo types | I | (2) |
| list of available prompt and echo types | L(I) | (1, −1) |
| default echo volume | 3×B | (full window) |
| default valuator data record containing at least: | D | |
| low value | R | (0) |
| high value | R | (1) |
| number of logical input devices of class CHOICE | I | API (1) |
| choice device number | I | (1) |
| maximum number of choice alternatives | I | (100) |
| number of available prompt and echo types | I | (2) |
| list of available prompt and echo types | L(I) | (1, 3) |
| default echo volume | 3×B | (full window) |
| default choice data record | D | (empty) |
| number of logical input devices of class PICK | I | API (1) |
| pick device number | I | (1) |
| number of available prompt and echo types | I | ETI |
| list of available prompt and echo types | L(I) | ETI |
| default echo volume | 3×B | (full window) |
| default pick data record | D | (empty) |

*PHIGS Workstation Description Table — Continued*

| *Workstation Description Table Entry* | *Data Type* | *Initial Value* |
|---|---|---|
| number of logical input devices of class STRING | I | API (1) |
| string device number | I | (1) |
| maximum input buffer size | I | (1024) |
| number of available prompt and echo types | I | (1) |
| list of available prompt and echo types | L(I) | (1) |
| default echo volume | 3×B | (full window) |
| default string data record containing at least: | D | |
| input buffer size (characters) | I | (1024) |
| initial cursor position | I | (1) |

**Table 1 — Predefined View Representations**

| *View* | *Orientation Matrix* | | | | *Mapping Matrix* | | | | *Clip Limits* |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0, 1.0 |
| full view | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0, 1.0 |
| | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0, 1.0 |
| | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 | |
| 1 | 1.0 | 0.0 | 0.0 | 0.0 | 0.3 | 0.0 | 0.0 | 0.1 | 0.0, 1.0 |
| parallel front | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.3 | 0.0 | 0.1 | 0.0, 1.0 |
| view in lower | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0, 1.0 |
| left corner | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 | |
| 2 | 1.0 | 0.0 | 0.0 | 0.0 | 0.3 | 0.0 | 0.0 | 0.1 | 0.0, 1.0 |
| top view in | 0.0 | 0.0 | -1.0 | 0.0 | 0.0 | 0.3 | 0.0 | 0.9 | 0.0, 1.0 |
| upper left | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0, 1.0 |
| corner | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 | |
| 3 | 0.0 | 0.0 | -1.0 | 0.0 | 0.3 | 0.0 | 0.0 | 0.9 | 0.0, 1.0 |
| right side | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.3 | 0.0 | 0.1 | 0.0, 1.0 |
| view in lower | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0, 1.0 |
| right corner | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 | |
| 4 | 0.7071 | 0.0 | -0.7071 | 0.0 | 0.2121 | 0.0 | 0.0 | 0.75 | 0.0, 1.0 |
| off-axis view | 0.4082 | 1.0 | -0.4082 | 0.0 | 0.0 | 0.2121 | 0.0 | 0.75 | 0.0, 1.0 |
| in upper right | 0.5774 | 0.5774 | 0.5774 | 0.0 | 0.0 | 0.0 | 0.5774 | 0.0 | 0.0, 1.0 |
| corner | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 | |
| 5 | 1.7071 | 0.0 | -0.7071 | 0.0 | 0.3536 | 0.0 | -0.025 | 0.5 | 0.0, 1.0 |
| off-axis per- | 0.4082 | 1.0 | -0.4082 | 0.0 | 0.0 | 0.3536 | -0.025 | 0.5 | 0.0, 1.0 |
| spective view | 0.5774 | 0.5774 | 0.5774 | 0.0 | 0.0 | 0.0 | 0.5274 | 0.0 | 0.0, 1.0 |
| in whole window | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | -0.05 | 1.0 | |

| *View* | *View Reference Point* | *View Up Vector* | *View Plane Normal* | *Projection Reference Point* | *View Mapping Window X Limits* | *View Mapping Window Y Limits* |
|---|---|---|---|---|---|---|
| 1 | (0,0,0) | <0,1,0> | <0,0,1> | (0.5,0.5,5.0) | [0,1] | [0,1] |
| 2 | (0,0,0) | <0,0,-1> | <0,1,0> | (0.5,-0.5,5.0) | [0,1] | [-1,0] |
| 3 | (0,0,0) | <0,1,0> | <1,0,0> | (-0.5,0.5,5.0) | [-1,0] | [0,1] |
| 4 | (0,0,0) | <0,1,0> | <1,1,1> | (0,0,5) | [-0.7071,0.7071] | [-0.7071,0.7071] |
| 5 | (0,0,0) | <0,1,0> | <1,1,1> | (0,0,20) | [-0.7071,0.7071] | [-0.7071,0.7071] |

| *View* | *View Plane* | *Front Plane* | *Back Plane* | *View Mapping Viewport X Limits* | *View Mapping Viewport Y Limits* | *View Mapping Viewport Z Limits* |
|---|---|---|---|---|---|---|
| 1 | 0.0 | 1.0 | 0.0 | [0.1,0.4] | [0.1,0.4] | [0,1] |
| 2 | 0.0 | 1.0 | 0.0 | [0.1,0.4] | [0.6,0.9] | [0,1] |
| 3 | 0.0 | 1.0 | 0.0 | [0.6,0.9] | [0.1,0.4] | [0,1] |
| 4 | 0.0 | 1.7321 | 0.0 | [0.6,0.9] | [0.6,0.9] | [0,1] |
| 5 | 10.0 | 1.7321 | 0.0 | [0,1] | [0,1] | [0,1] |

**Table 2 — Predefined Polyline Bundle Table**

| *Bundle Index* | *Linetype* | *Line Width Scale Factor* | *Colour* | *Shading Method* | *Approximation Type* | *Approx. Value* |
|---|---|---|---|---|---|---|
| 1 | PLINE_SOLID | 1.0 | PINDIRECT, 1 | PSD_NONE | PCURV_WS_DEP | 1.0 |
| 2 | PLINE_DASH | 1.0 | PINDIRECT, 1 | PSD_NONE | PCURV_WS_DEP | 1.0 |
| 3 | PLINE_DOT | 1.0 | PINDIRECT, 1 | PSD_NONE | PCURV_WS_DEP | 1.0 |
| 4 | PLINE_DOT_DASH | 1.0 | PINDIRECT, 1 | PSD_NONE | PCURV_WS_DEP | 1.0 |
| 5 | PLINE_SOLID | 2.0 | PINDIRECT, 1 | PSD_NONE | PCURV_WS_DEP | 1.0 |

**Table 3 — Predefined Polymarker Bundle Table**

| *Bundle Index* | *Marker Type* | *Marker Size Scale Factor* | *Colour* |
|---|---|---|---|
| 1 | PMARKER_DOT | 1.0 | (PINDIRECT, 1) |
| 2 | PMARKER_PLUS | 1.0 | (PINDIRECT, 1) |
| 3 | PMARKER_ASTERISK | 1.0 | (PINDIRECT, 1) |
| 4 | PMARKER_CIRCLE | 1.0 | (PINDIRECT, 1) |
| 5 | PMARKER_CROSS | 1.0 | (PINDIRECT, 1) |

**Table 4 — Predefined Text Bundle Table**

| *Bundle Index* | *Font* | *Text Precision* | *Expansion Factor* | *Character Spacing* | *Colour* |
|---|---|---|---|---|---|
| 1 | PFONT_MONO | PPREC_STROKE | 1.0 | 0.0 | PINDIRECT, 1 |
| 2 | PFONT_MONO | PPREC_STROKE | 1.5 | 0.0 | PINDIRECT, 1 |
| 3 | PFONT_MONO | PPREC_STROKE | 2.0 | 0.0 | PINDIRECT, 1 |
| 4 | PFONT_MONO | PPREC_STROKE | 2.5 | 0.0 | PINDIRECT, 1 |
| 5 | PFONT_MONO | PPREC_STROKE | 3.0 | 0.0 | PINDIRECT, 1 |
| 2 | PFONT_MONO | PPREC_STROKE | 3.5 | 0.0 | PINDIRECT, 1 |

**Table 5 — Predefined Interior Bundle Table**

| *Bundle Index* | *Interior Style* | *Back Interior Style* |
|---|---|---|
| 1 | PSTYLE_SOLID | PSTYLE_SOLID |
| 2 | PSTYLE_HOLLOW | PSTYLE_HOLLOW |
| 3 | PSTYLE_EMPTY | PSTYLE_EMPTY |
| 4 | PSTYLE_SOLID | PSTYLE_HOLLOW |
| 5 | PSTYLE_SOLID | PSTYLE_EMPTY |

All other attributes have the same pre-defined values for all of the Bundle Indices, as shown in the following tables:

| *Interior Style Index* | *Colour* | *Reflectance Equation* | *Shading Method* | *Ambient Coefficient* | *Diffuse Coefficient* | *Specular Coefficient* |
|---|---|---|---|---|---|---|
| 1 | PINDIRECT, 1 | PREFL_NONE | PSD_NONE | 1.0 | 1.0 | 0.0 |

| *Specular Colour* | *Specular Exponent* | *Back Facing Interior Style Index* | *Back Facing Specular Colour* |
|---|---|---|---|
| PINDIRECT, 1 | 0.0 | 1 | PINDIRECT, 1 |

| *Back Facing Reflectance Equation* | *Back Facing Shading Method* | *Back Facing Ambient Coefficient* | *Back Facing Diffuse Coefficient* | *Back Facing Specular Coefficient* |
|---|---|---|---|---|
| PREFL_NONE | PSD_NONE | 1.0 | 1.0 | 0.0 |

| *Back Facing Specular Exponent* | *Surface Approximation Type* | *Surface Approximation Values* |
|---|---|---|
| 0.0 | PSURF_WS_DEP | 1.0 |

**Table 6 — Predefined Edge Bundle Table**

| *Bundle Index* | *Edge Flag* | *Edgetype* | *Edge Width Scale Factor* | *Colour* |
|---|---|---|---|---|
| 1 | PEDGE_ON | PLINE_SOLID | 1.0 | PINDIRECT, 1 |
| 2 | PEDGE_ON | PLINE_DASH | 1.0 | PINDIRECT, 1 |
| 3 | PEDGE_ON | PLINE_DOT | 1.0 | PINDIRECT, 1 |
| 4 | PEDGE_ON | PLINE_DOT_DASH | 1.0 | PINDIRECT, 1 |
| 5 | PEDGE_ON | PLINE_SOLID | 2.0 | PINDIRECT, 1 |

**Table 7 — Predefined Depth Cue Table**

| *Depth Cue Index* | *Depth Cue Mode* | *Depth Cue Reference Planes* | *Depth Cue Scale Factors* | *Depth Cue Colour* |
|---|---|---|---|---|
| 0 | SUPPRESSED | (0.0,1.0) | (0.0,1.0) | (PINDIRECT,0) |
| 1 | ALLOWED | (0.0,1.0) | (0.0,1.0) | (PINDIRECT,0) |

**Table 8 — Predefined Colour Mapping Table**

| *Index* | *Method* |
|---|---|
| 0 | PCOLR_MAP_TRUE |

**Table 9 — Predefined Light Sources**

| *Index* | *Type* | *Data Record* |
|---|---|---|
| 1 | PLIGHT_AMBIENT | (INDIRECT,1) |
| 2 | PLIGHT_AMBIENT | (INDIRECT,1) |

**SEE ALSO**

INQUIRE WORKSTATION CONNECTION AND TYPE (3P)
WORKSTATION TYPE CREATE (3P)
WORKSTATION TYPE GET (3P)
WORKSTATION TYPE SET (3P)