PHIGS TOOLKIT

Department of Computer Science
University of Manchester

Computer Graphics Unit
Manchester Computing Centre
University of Manchester

PTK/101 – PHIGS Toolkit C Reference Manual

Author: Toby Howard, W.T. Hewitt, J.G. Williams, S. Larkin

Date: 2nd September, 1992

Version: 3.1

Preface

Read this first!

The PHIGS Toolkit contains many tools to help application programmers. Some, such as those in the PHIGS Utility Library are single, self-contained procedures. These are referred to as **Programming Tools**. Others, such as the PHIGS Debugger and the PHIGS View Editor, are complex interactive tools, and are called **High-Level Tools**. The PHIGS Toolkit documentation consists of three separate manuals. The first is a **User Guide**, and provides an overview of the functionality of the PHIGS Toolkit. The second manual is a **C Reference Manual**, which gives detailed specifications of each C function in the PHIGS Toolkit. The third and final PHIGS Toolkit manual is a **Fortran Reference Manual**.

Because it is simply not practicable to give detailed examples of the use of every function in the PHIGS Toolkit, to get the most out the Toolkit we suggest you read this preface, the introductory chapter (Chapter 1), and then Chapters 4 to 11 of the User Guide, which describe the functionality of the tools. Next, browse through the function descriptions for each module in the Toolkit in the relevant language reference manual, and have a look at some of the demonstration programs described in Chapter 3 of the User Guide.

History

Ideas for a PHIGS application programmers' Toolkit began during the KRT^3 research project [**?**]. The motivation to create the prototype Toolkit arose out of our experiences trying to program with PHIGS, when it gradually became clear that in many instances the functionality of PHIGS was at too low a level to make programming productive and reliable. The first module to be written was the PHIGS Utility Library, which effectively provided the programmer with a set of functions which were 'PHIGS macros', bundling up commonly used sequences of PHIGS functions into single procedures. Also included in this library were useful functions which PHIGS itself did not provide, such as 'inquire number of elements in structure', 'copy element' and so on. The next major tool to be developed was the Topology Drawer which provided a mechanism for automatically drawing neat diagrams of structure network hierarchies, with the information inquired directly from PHIGS. The difficulties of debugging structure networks (due to the 'black-box' nature of the traversal process) led to the development of the PHIGS Debugger, an interactive tool for tracking the traversal of structure networks.

The prototype PHIGS Toolkit was written in Pascal, and ran only on VAX/VMS, using DEC PHIGS. Subsequently, the Toolkit has been completely revised and rewritten in C by Gareth Williams, and made operable with SunPHIGS (under SunOS), HP PHIGS (under HP-UX) and DEC PHIGS (under VAX/VMS). A Fortran binding has also been provided for both SunOS and VAX/VMS.

Acknowledgements

The PHIGS Toolkit arose from work on the KRT^3 project (1985–88), supported by the Science and Engineering Research Council, under grant GR/D/20533. We are grateful to Digital Equipment Corporation for their support during the initial development of the Toolkit. The development of the production version of the Toolkit was made possible with a grant from AGOCG/Computer Board, and for their hard work administering the PHIGS Toolkit project, we are grateful to Eric Thomas and Julian Gallop. For practical help in commissioning the software, we must thank our official Beta-testers: Dr K.W. Brodlie and Colin Pinks (School of Computer Studies, University of Leeds), Dr I. Haneef and Dr Howard Parish (Department of Biochemistry and Molecular Biology, University of Leeds), and David Johnston (Rutherford Appleton Laboratory). We are also very grateful to the following undergraduate and postgraduate students from the Department of Computer Science at the University of Manchester for their 'no holds barred' feedback about the Toolkit: Richard Lee, Emmanuel Lafouillade, Guy Cuthbert, and Percy Tam. Last, but not least, we are indebted to our colleagues, who over the years have generously shared with us their advice and support; in particular, Tony Arnold, David Carlisle, Graham Gough, Andy Grant, Roger Hubbold, Rachel Lau, Chris Lilley, Fenqiang Lin, Manjula Patel, Adel Nasser, Karen Wyrwas, David Yip and Fuyan Zhang. This manual was prepared using GNU Emacs, PS-LaTeX and GhostScript.

Trademark notice

Disclaimer

The PHIGS Toolkit is produced by Department of Computer Science, and the Computer Graphics Unit, Manchester Computing Centre, both of the Victoria University of Manchester. The PHIGS Toolkit has been tested with care but is not guaranteed for any particular purpose. Neither the authors, nor the University of Manchester offer any warranties or representations, nor do they accept any liabilities with respect to the software and documentation. The PHIGS Toolkit may be used freely within the academic community, but it may not be used for commercial gain without the written permission of the authors.

Contents

Chapter 1

Transformation functions

1.1 Datatypes

```
#define ptkcpceps 1.0e-7

#define ptkcpcok  0

typedef enum
   {
    PTKEXAXIS = 1,
    PTKEYAXIS,
    PTKEZAXIS,
   } ptkeaxistype;

typedef struct
   {
    Pfloat x;
    Pfloat y;
    Pfloat z;
    Pfloat w;
   } Ppoint4;
```

1.2 Functions

ptk_0to3pt

void **ptk_0to3pt** (Ppoint3 *origin, Ppoint3 *xdirn, Ppoint3 *ydirn, Pcompose_type operation, Pmatrix3 matrix)

IN	Ppoint3 *	origin	origin of axes.
IN	Ppoint3 *	xdirn	x direction.
IN	Ppoint3 *	y dirn	y direction.
IN	Pcompose_type	operation	concatenation operation.
OUT	Pmatrix3	matrix	4x4 matrix.

This function computes an object transformation which maps unit vectors along the *x*, *y* and *z* axes onto unit vectors along the corresponding axes of the new coordinate system.

ptk_3ptto3pt

void **ptk_3ptto3pt** (Ppoint3 *p1, Ppoint3 *p2, Ppoint3 *p3, Ppoint3 *q1, Ppoint3 *q2, Ppoint3 *q3, Pcompose_type operation, Pmatrix3 matrix, Pint *error)

IN	Ppoint3 *	p1	3D point.
IN	Ppoint3 *	p2	3D point.
IN	Ppoint3 *	p3	3D point.
IN	Ppoint3 *	q1	3D point.
IN	Ppoint3 *	q2	3D point.
IN	Ppoint3 *	q3	3D point.
IN	Pcompose_type	operation	concatenation operation.
OUT	Pmatrix3	matrix	4x4 matrix.
OUT	Pint *	error	error code.

This function returns the 3 point to 3 point transformation as described in [**?**], pages 353–355. The transformation has the following properties: `p1` is transformed onto `q1`; the vector `(p2-p1)` is transformed to be parallel to the vector `(q2-q1)`; the plane containing the three points `p1, p2, p3` is transformed into the plane containing `q1, q2, q3`. The transformation is concatenated with the 4×4 matrix `matrix` on the basis of `operation`. If the parameters are invalid, `error` is set to -1. Otherwise, its value is `ptkcpcok`.

ptk_accumulatetran

void **ptk_accumulatetran** (Ppoint *fixed, Ppoint *shift, Pfloat rot, Ppoint *scale, Pcompose_type operation, Pmatrix matrix)

IN	Ppoint *	fixed	origin.
IN	Ppoint *	shift	shift factor.
IN	Pfloat	rotx	x rotation.
IN	Ppoint *	scale	scale factor.
IN	Pcompose_type	operation	concatenation operation.
OUT	Pmatrix	matrix	3x3 matrix.

This function computes the specified 2D shift, scale and rotate transformations, in the order scale, rotate, shift, and then concatenates the resulting transformation with the specified matrix on the basis of `operation`.

ptk_accumulatetran3

void **ptk_accumulatetran3** (Ppoint3 *fixed, Ppoint3 *shift, Pfloat rotx, Pfloat roty, Pfloat rotz, Ppoint3 *scale, Pcompose_type operation, Pmatrix3 matrix)

IN	Ppoint3 *	fixed	origin.
IN	Ppoint3 *	shift	shift factor.
IN	Pfloat	rotx	x rotation.
IN	Pfloat	rotx	y rotation.
IN	Pfloat	rotx	z rotation.
IN	Ppoint3 *	scale	scale factor.
IN	Pcompose_type	operation	concatenation operation.
OUT	Pmatrix3	matrix	4x4 matrix.

This function computes the specified 3D shift, scale and rotate transformations, in the order scale, rotate, shift, and then concatenates the resulting transformation with the specified matrix on the basis of `operation`.

ptk_addv

Ppoint **ptk_addv** (Ppoint *`p1`, Ppoint *`p2`)

IN	Ppoint *	p1	2D vector.
IN	Ppoint *	p2	2D vector.

This function adds the two 2D vectors `v1` and `v2`, and returns the result.

ptk_addv3

Ppoint3 **ptk_addv3** (Ppoint3 *`p1`, Ppoint3 *`p2`)

IN	Ppoint3 *	p1	3D vector.
IN	Ppoint3 *	p2	3D vector.

This function adds the two 3D vectors `v1` and `v2`, and returns the result.

ptk_box3tobox3

void **ptk_box3tobox3** (Plimit3 *box1, Plimit3 *box2, ptkboolean preserve, Pcompose_type operation, Pmatrix3 matrix, Pint *error)

IN	Plimit3 *	box1	3D volume.
IN	Plimit3 *	box2	3D volume.
IN	ptkboolean	preserve	preserve aspect ratio.
IN	Pcompose_type	operation	concatenation operation.
OUT	Pmatrix3	matrix	4x4 matrix.
OUT	Pint *	error	error code.

This function computes a mapping from one 3D box to another – a 3D window to 3D viewport transformation – and concatenates this transformation with `matrix` on the basis of `operation`.If the parameters are invalid, `error` is set to -1. Otherwise, its value is `ptkcpcok`.

ptk_boxtobox

void **ptk_boxtobox** (Plimit *box1, Plimit *box2, ptkboolean preserve, Pcompose_type operation, Pmatrix matrix, Pint *error)

IN	Plimit *	box1	2D box.
IN	Plimit *	box2	2D box.
IN	ptkboolean	preserve	preserve aspect ratio.
IN	Pcompose_type	operation	concatenation operation.
OUT	Pmatrix	matrix	3x3 matrix.
OUT	Pint *	error	error code.

This function computes a mapping from one 2D box to another – a 2D window to 2D viewport transformation —and concatenates this transformation it with `matrix` on the basis of `operation`. If the parameters are invalid, `error` is set to -1. Otherwise, its value is `ptkcpcok`.

ptk_concatenatematrix

void **ptk_concatenatematrix** (Pcompose_type operation, Pmatrix matrix1, Pmatrix matrix2, Pmatrix result)

IN	Pcompose_type	operation	concatenation operation.
IN	Pmatrix	matrix1	3x3 matrix.
IN	Pmatrix	matrix2	3x3 matrix.
OUT	Pmatrix	result	3x3 matrix.

This function concatenates the 3×3 matrices `matrix1` and `matrix2` on the basis of `operation`. The result is stored in `result`. Note that `result` can also be `matrix1` or `matrix2` since a copy is made. When `operation` is `preconcatenate`, then `result` ← `matrix2 * matrix1`. When `operation` is `postconcatenate`, `result` ← `matrix1 * matrix2`.

ptk_concatenatematrix3

void **ptk_concatenatematrix3** (Pcompose_type operation, Pmatrix3 matrix1, Pmatrix3 matrix2, Pmatrix3 result)

IN	Pcompose_type	operation	concatenation operation.
IN	Pmatrix3	matrix1	4x4 matrix.
IN	Pmatrix3	matrix2	4x4 matrix.
OUT	Pmatrix3	result	4x4 matrix.

This function concatenates the 4×4 matrices `matrix1` and `matrix2` on the basis of `operation`. The result is stored in `result`. Note that `result` can also be `matrix1` or `matrix2` since a copy is made. When `operation` is `preconcatenate`, then `result` ← `matrix2 * matrix1`. When `operation` is `postconcatenate`, `result` ← `matrix1 * matrix2`.

ptk_crossv3

Ppoint3 **ptk_crossv3** (Ppoint3 *v1, Ppoint3 *v2)

IN	Ppoint3 *	v1	3D vector.
IN	Ppoint3 *	v2	3D vector.

This function evaluates the cross product of the two 3D vectors `v1` and `v2`, returning the new vector as the function result. Since a local copy is made statements such as `v2 == ptk_crossv(v1, v2)` will produce the correct answer.

ptk_dotv

Pfloat **ptk_dotv** (Ppoint *v1, Ppoint *v2)

IN	Ppoint *	v1	2D vector.
IN	Ppoint *	v2	2D vector.

Evaluates the dot product of the two 2D vectors `v1` and `v2`, returning it as the value of the function.

ptk_dotv3

Pfloat **ptk_dotv3** (Ppoint3 *v1, Ppoint3 *v2)

IN	Ppoint3 *	v1	3D vector.
IN	Ppoint3 *	v2	3D vector.

This function evaluates the dot product of the two 3D vectors `v1` and `v2`, returning it as the value of the function.

ptk_equal

ptkboolean **ptk_equal** (Pfloat one, Pfloat two)

IN	Pfloat	one	floating point number.
IN	Pfloat	two	floating point number.

This function returns TRUE if `one` and `two` are equal, or their difference is less than the global constant tolerance `ptkveps`. This is a global variable of type `Pfloat` which may be changed by the application. The default value of `ptkveps` is 1.0e-7.

ptk_evalviewmapping

void **ptk_evalviewmapping** (Plimit *wlimits, Plimit *vlimits, Pcompose_type operation, Pmatrix matrix, Pint *error)

IN	Plimit *	wlimits	window limits.
IN	Plimit *	vlimits	viewport limits.
IN	Pcompose_type	operation	concatenation operation.
OUT	Pmatrix	matrix	3x3 matrix.
OUT	Pint *	error	error code.

This function evaluates a 2d PHIGS view mapping matrix. If the function succeeds, `error` is set to `ptkcpcok`. Otherwise, `error` is 329 if the window limits are not valid, and 330 if the viewport limits are not valid.

ptk_evalviewmapping3

void **ptk_evalviewmapping3** (Plimit3 *wlimits, Plimit3 *vlimits, Pproj_type viewtype, Ppoint3 *refpoint, Pfloat vplanedist, Pcompose_type operation, Pmatrix3 matrix, Pint *error)

IN	Plimit3 *	wlimits	window limits.
IN	Plimit3 *	vlimits	viewport limits.
IN	Pproj_type	viewtype	projection type.
IN	Ppoint3 *	refpoint	projection reference point.
IN	Pfloat	vplanedist	view plane distance.
IN	Pcompose_type	operation	concatenation operation.
OUT	Pmatrix3	matrix	4x4 matrix.
OUT	Pint *	error	error code.

This function evaluates a 3D PHIGS view mapping matrix. If the function succeeds, `error` is set to `ptkcpcok`. Otherwise, `error` is 329 if the window limits are not valid, 336 if the back plane is in front of front plane, 330 if the viewport limits are not valid, 335 if the projection reference point is on the view plane, and 340 if the projection reference point is between front and back planes.

ptk_evalvieworientation

void **ptk_evalvieworientation** (Ppoint *viewrefpoint, Ppoint *viewupvector, Pcompose_type operation, Pmatrix matrix, Pint *error)

IN	Ppoint *	viewrefpoint	view reference point.
IN	Ppoint *	viewupvector	view up vector.
IN	Pcompose_type	operation	concatenation operation.
OUT	Pmatrix	matrix	3x3 matrix.
OUT	Pint *	error	error code.

This function computes a 2D PHIGS view orientation matrix on the basis of a specified view reference point (`viewrefpoint`) and a view up vector (`viewupvector`). If the function succeeds, `error` is set to `ptkcpcok`. Otherwise, `error` is 63 if the view up vector is null.

ptk_evalvieworientation3

void **ptk_evalvieworientation3** (Ppoint3 *viewrefpoint, Ppoint3 *viewplanenormal, Ppoint3 *viewupvector, Pcompose_type operation, Pmatrix3 matrix, Pint *error)

IN	Ppoint3 *	viewrefpoint	view reference point.
IN	Ppoint3 *	viewplanenormal	view plane normal.
IN	Ppoint3 *	viewupvector	view up vector.
IN	Pcompose_type	operation	concatenation operation.
OUT	Pmatrix3	matrix	4x4 matrix.
OUT	Pint *	error	error code.

This function computes a 3D PHIGS view orientation matrix on the basis of a specified view reference point (`viewrefpoint`), a view plane normal (`viewplanenormal`) and a view up vector (`viewupvector`). If the function succeeds, `error` is set to `ptkcpcok`. Otherwise, `error` is 61 if the view plane normal is null, 63 if the view up vector is null, and 58 if the cross product of the view up vector and the view plane normal is null.

ptk_examinestackmatrix

void **ptk_examinestackmatrix** (Pmatrix matrix)

IN	Pmatrix	matrix	3x3 matrix.

This function returns the top entry on the transformation stack. The stack is not disturbed.

ptk_examinestackmatrix3

void **ptk_examinestackmatrix3** (Pmatrix3 matrix)

IN	Pmatrix3	matrix	4x4 matrix.

This function returns the top entry on the transformation stack. The stack is not disturbed.

ptk_invertmatrix

void **ptk_invertmatrix** (Pmatrix a, Pmatrix ainverse, Pint *error)

IN	Pmatrix	a	3x3 matrix.
OUT	Pmatrix	ainverse	3x3 matrix.
OUT	Pint *	error	error code.

This function computes the inverse of the 3×3 matrix `a`, returning the result in `ainverse`. If matrix `a` is singular, then `error` is set to -1, and `ainverse` is undefined, otherwise `error` is set to `ptkcpcok`. This function uses Crout's method, a modification of Gaussian elimination.

ptk_invertmatrix3

void **ptk_invertmatrix3** (Pmatrix3 a, Pmatrix3 ainverse, Pint *error)

IN	Pmatrix3	a	4x4 matrix.
OUT	Pmatrix3	ainverse	4x4 matrix.
OUT	Pint *	error	error code.

This function computes the inverse of the 4×4 matrix `a`, returning the result in `ainverse`. If matrix `a` is singular, then `error` is set to -1, and `ainverse` is undefined, otherwise `error` is set to `ptkcpcok`. This function uses Crout's method, a modification of Gaussian elimination.

ptk_limit

Plimit **ptk_limit** (Pfloat xmin, Pfloat xmax, Pfloat ymin, Pfloat ymax)

IN	Pfloat	xmin	minimum x coordinate.
IN	Pfloat	xmax	maximum x coordinate.
IN	Pfloat	ymin	minimum y coordinate.
IN	Pfloat	ymax	maximum y coordinate.

This function returns a structure representing the 2D rectangle whose corners are defined by `xmin, xmax, ymin, ymax`.

ptk_limit3

Plimit3 **ptk_limit3** (Pfloat xmin, Pfloat xmax, Pfloat ymin, Pfloat ymax, Pfloat zmin, Pfloat zmax)

IN	Pfloat	xmin	minimum x coordinate.
IN	Pfloat	xmax	maximum x coordinate.
IN	Pfloat	ymin	minimum y coordinate.
IN	Pfloat	ymax	maximum y coordinate.
IN	Pfloat	zmin	minimum z coordinate.
IN	Pfloat	zmax	maximum z coordinate.

This function returns a structure representing the 3D volume defined by `xmin, xmax, ymin, ymax, zmin, zmax`.

ptk_matrixtomatrix3

void **ptk_matrixtomatrix3** (Pmatrix mat, Pmatrix3 mat3)

IN	Pmatrix	mat	3x3 matrix.
OUT	Pmatrix3	mat3	4x4 matrix.

This function converts the 3×3 matrix `mat` to the 4×4 matrix `mat3`, as follows:

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & j \end{pmatrix} \rightarrow \begin{pmatrix} a & b & 0 & c \\ d & e & 0 & f \\ 0 & 0 & 1 & 0 \\ g & h & 0 & j \end{pmatrix}$$

ptk_modv

Pfloat **ptk_modv** (Ppoint *vec)

IN	Ppoint *	vec	2D vector.

This function returns the modulus of the 2D vector `vec`.

ptk_modv3

Pfloat **ptk_modv3** (Ppoint3 *vec)

IN	Ppoint3 *	vec	3D vector.

Returns the modulus of the vector `vec`.

ptk_multiplymatrix

void **ptk_multiplymatrix** (Pmatrix matrix1, Pmatrix matrix2, Pmatrix result)

IN	Pmatrix	matrix1	3x3 matrix.
IN	Pmatrix	matrix2	3x3 matrix.
OUT	Pmatrix	result	3x3 matrix.

This function makes `result` the product of the 3×3 matrices `matrix1` and `matrix2`, with `result` ← `matrix1 * matrix2`. Note that `result` can also be `matrix1` or `matrix2` since a copy is made.

ptk_multiplymatrix3

void **ptk_multiplymatrix3** (Pmatrix3 matrix1, Pmatrix3 matrix2, Pmatrix3 result)

IN	Pmatrix3	matrix1	4x4 matrix.
IN	Pmatrix3	matrix2	4x4 matrix.
OUT	Pmatrix3	result	4x4 matrix.

This function makes `result` the product of the 4×4 matrices `matrix1` and `matrix2`, with `result` ← `matrix1 * matrix2`. Note that `result` can also be `matrix1` or `matrix2` since a copy is made. This function uses Winograd's method, see "Handbook of Algorithms and Data Structures", G.H. Gonnet, Addison Wesley.

ptk_nullv

ptkboolean **ptk_nullv** (Ppoint *vec)

IN	Ppoint *	vec	2D vector.

This function returns `TRUE` if the modulus of the 2D vector `vec` is less than the global tolerance `ptkpceps`, otherwise `FALSE`.

ptk_nullv3

ptkboolean **ptk_nullv3** (Ppoint3 *vec)

IN	Ppoint3 *	vec	3D vector.

This function returns `TRUE` if the modulus of the 3D vector`vec` is less than the global tolerance `ptkpceps`, otherwise `FALSE`.

ptk_oto3pt

void **ptk_oto3pt** (Ppoint3 *origin, Ppoint3 *xdirn, Ppoint3 *ydirn, Pcompose_type operation, Pmatrix3 matrix)

IN	Ppoint3 *	origin	origin of axes.
IN	Ppoint3 *	xdirn	x direction.
IN	Ppoint3 *	y dirn	y direction.
IN	Pcompose_type	operation	concatenation operation.
OUT	Pmatrix3	matrix	4x4 matrix.

This function performs the same operation as `ptk_0to3pt`, except the name has an `o` (oh) instead of `0` (zero). This function is provided for members of the Fumbly Fingers Club.

ptk_outputmatrix3

void **ptk_outputmatrix3** (FILE *fileptr, Pmatrix3 matrix, char *string)

OUT	FILE *	fileptr	file pointer.
IN	Pmatrix3	matrix	4x4 matrix.
IN	char *	string	character string.

This function outputs the 3×3 matrix `matrix` and the message `string` to the file specified by `fileptr`.

ptk_point

Ppoint **ptk_point** (Pfloat x, Pfloat y)

IN	Pfloat	x	x coordinate.
IN	Pfloat	y	y coordinate.

This function returns a structure representing the 2D point `(x,y)`.

ptk_point3

Ppoint3 **ptk_point3** (Pfloat x, Pfloat y, Pfloat z)

IN	Pfloat	x	x coordinate.
IN	Pfloat	y	y coordinate.
IN	Pfloat	z	z coordinate.

This function returns a structure representing the 3D point `(x,y,z)`.

ptk_pt3topt4

Ppoint4 **ptk_pt3topt4** (Ppoint3 *point)

IN	Ppoint3 *	point	3D point.

This function converts the 3D point `point` to a 4D point, which is returned as the result of the function. The *w* coordinate of the 4D point is set to 1.0.

ptk_pt3tovec3

Pvec3 **ptk_pt3tovec3** (Ppoint3 *pt)

IN	Ppoint3 *	pt	3D point.

This function converts the 3D point `pt` to a 3D vector.

ptk_pt4topt3

Ppoint3 **ptk_pt4topt3** (Ppoint4 *point)

IN	Ppoint4 *	point	4D point.

This function converts the 4D point `point` to a 3D point, by dividing by *w*. The 3D point is returned as the result of the function.

ptk_rotate

void **ptk_rotate** (Pfloat rotation, ptkeaxistype axis, Pcompose_type operation, Pmatrix matrix)

IN	Pfloat	rotation	angle in degrees.
IN	ptkeaxistype	axis	x, y or z axis.
IN	Pcompose_type	operation	concatenation operation.
OUT	Pmatrix	matrix	3x3 matrix.

This function computes a matrix to perform the specified 2D rotation and concatenates this matrix with `matrix` on the basis of `operation`. `rotation` is expressed in degrees.

ptk_rotate3

void **ptk_rotate3** (Pfloat rotation, ptkeaxistype axis, Pcompose_type operation, Pmatrix3 matrix)

IN	Pfloat	rotation	angle in degrees.
IN	ptkeaxistype	axis	x, y or z axis.
IN	Pcompose_type	operation	concatenation operation.
OUT	Pmatrix3	matrix	4x4 matrix.

This function computes a matrix to perform the specified 3D rotation and concatenates this matrix with `matrix` on the basis of `operation`. `rotation` is expressed in degrees.

ptk_rotatecs

void **ptk_rotatecs** (Pfloat costheta, Pfloat sinetheta, ptkeaxistype axis, Pcompose_type operation, Pmatrix matrix)

IN	Pfloat	costheta	cosine of angle.
IN	Pfloat	sinetheta	sine of angle.
IN	ptkeaxistype	axis	x, y or z axis.
IN	Pcompose_type	operation	concatenation operation.
OUT	Pmatrix	matrix	3x3 matrix.

This function computes a matrix to perform the specified 2D rotation and concatenates this matrix with `matrix` on the basis of `operation`. This form assumes that the rotation is specified using the cos(*theta*) and sin(*theta*) terms. Note that no check is made to ensure that the sum of the squares of these terms is 1.

ptk_rotatecs3

void **ptk_rotatecs3** (Pfloat costheta, Pfloat sinetheta, ptkeaxistype axis, Pcompose_type operation, Pmatrix3 matrix)

IN	Pfloat	costheta	cosine of angle.
IN	Pfloat	sinetheta	sine of angle.
IN	ptkeaxistype	axis	x, y or z axis.
IN	Pcompose_type	operation	concatenation operation.
OUT	Pmatrix3	matrix	4x4 matrix.

This function computes a matrix to perform the specified 3D rotation and concatenates this matrix with `matrix` on the basis of `operation`. This form assumes that the rotation is specified using the cos(*theta*) and sin(*theta*) terms. Note that no check is made to ensure that the sum of the squares of these terms is 1.

ptk_rotateline

void **ptk_rotateline** (Ppoint *p1, Ppoint *p2, Pfloat theta, Pcompose_type operation, Pmatrix matrix, Pint *error)

IN	Ppoint *	p1	2D point.
IN	Ppoint *	p2	2D point.
IN	Pfloat	theta	angle in degrees.
IN	Pcompose_type	operation	concatenation operation.
OUT	Pmatrix	matrix	3x3 matrix.
OUT	Pint *	error	error code.

This function computes a matrix to perform a 2D rotation of `theta` degrees about the line connecting `p1` to `p2`, and concatenates this matrix with `matrix` on the basis of `operation`. If the parameters are invalid, `error` is set to -1. Otherwise, its value is `ptkcpcok`.

ptk_rotateline3

void **ptk_rotateline3** (Ppoint3 *p1, Ppoint3 *p2, Pfloat theta, Pcompose_type operation, Pmatrix3 matrix, Pint *error)

IN	Ppoint3 *	p1	3D point.
IN	Ppoint3 *	p2	3D point.
IN	Pfloat	theta	angle in degrees.
IN	Pcompose_type	operation	concatenation operation.
OUT	Pmatrix3	matrix	4x4 matrix.
OUT	Pint *	error	error code.

This function computes a matrix to perform a 3D rotation of `theta` degrees about the line connecting `p1` to `p2`, and concatenates this matrix with `matrix` on the basis of `operation` ([**?**], pages 55–59). If the parameters are invalid, `error` is set to -1. Otherwise, its value is `ptkcpcok`.

ptk_rotatevv

void **ptk_rotatevv** (Ppoint *v1, Ppoint *v2, Pcompose_type operation, Pmatrix matrix, Pint *error)

IN	Ppoint *	v1	2D vector.
IN	Ppoint *	v2	2D vector.
IN	Pcompose_type	operation	concatenation operation.
OUT	Pmatrix	matrix	3x3 matrix.
OUT	Pint *	error	error code.

This function computes a matrix to perform the rotation (about the origin) of the 3D vector `v1` to the 3D vector `v2`, and concatenates this matrix with `matrix` on the basis of `operation` ([**?**], pages 55–59). If the parameters are invalid, `error` is set to -1. Otherwise, its value is `ptkcpcok`.

ptk_rotatevv3

void **ptk_rotatevv3** (Ppoint3 *v1, Ppoint3 *v2, Pcompose_type operation, Pmatrix3 matrix, Pint *error)

IN	Ppoint3 *	v1	3D vector.
IN	Ppoint3 *	v2	3D vector.
IN	Pcompose_type	operation	concatenation operation.
OUT	Pmatrix3	matrix	4x4 matrix.
OUT	Pint *	error	error code.

This function computes a matrix to perform the rotation (about the origin) of the 3D vector `v1` to the 3D vector `v2`, and concatenates this matrix with `matrix` on the basis of `operation` ([**?**], pages 55–59). If the parameters are invalid, `error` is set to -1. Otherwise, its value is `ptkcpcok`.

ptk_scale

void **ptk_scale** (Ppoint *scale, Pcompose_type operation, Pmatrix matrix)

IN	Ppoint *	shift	shift factor.
IN	Pcompose_type	operation	concatenation operation.
OUT	Pmatrix	matrix	3x3 matrix.

This function computes a matrix to perform the specified 2D scale and concatenates this matrix with `matrix` on the basis of `operation`.

ptk_scale3

void **ptk_scale3** (Ppoint3 *scale, Pcompose_type operation, Pmatrix3 matrix)

IN	Ppoint3 *	shift	shift factor.
IN	Pcompose_type	operation	concatenation operation.
OUT	Pmatrix3	matrix	4x4 matrix.

This function computes a matrix to perform the specified 3D scale and concatenates this with `matrix` on the basis of `operation`.

ptk_scalev

Ppoint **ptk_scalev** (Ppoint *vec, Pfloat scale)

IN	Ppoint *	vec	2D vector.
IN	Pfloat	scale	scale factor.

This function multiplies the 2D vector `v` by the scalar `s` and returns the result.

ptk_scalev3

Ppoint3 **ptk_scalev3** (Ppoint3 *vec, Pfloat scale)

IN	Ppoint3 *	vec	3D vector.
IN	Pfloat	scale	scale factor.

This function multiplies the 3D vector `v` by the scalar `s` and returns the result.

ptk_shear

void **ptk_shear** (ptkeaxistype shearaxis, ptkeaxistype sheardir, Pfloat shearfactor, Pcompose_type operation, Pmatrix matrix)

IN	ptkeaxistype	shearaxis	x or y axis.
IN	ptkeaxistype	sheardir	x or y direction.
IN	Pfloat	shearfactor	shear factor.
IN	Pcompose_type	operation	concatenation operation.
OUT	Pmatrix	matrix	3x3 matrix.

This function computes a matrix to perform the specified 2D shear and concatenates this matrix with `matrix` on the basis of `operation`. The shear is specified as an amount `f` about axis `i` in direction `j`.

ptk_shear3

void **ptk_shear3** (ptkeaxistype shearaxis, ptkeaxistype sheardir, Pfloat shearfactor, Pcompose_type operation, Pmatrix3 matrix)

IN	ptkeaxistype	shearaxis	x, y or z axis.
IN	ptkeaxistype	sheardir	x, y or z direction.
IN	Pfloat	shearfactor	shear factor.
IN	Pcompose_type	operation	concatenation operation.
OUT	Pmatrix3	matrix	4x4 matrix.

This function computes a matrix to perform the specified 3D shear and concatenates this matrix with `matrix` on the basis of `operation`. The shear is specified as an amount `f` about axis `i` in direction `j`.

ptk_shift

void **ptk_shift** (Ppoint *shift, Pcompose_type operation, Pmatrix matrix)

IN	Ppoint *	shift	shift factor.
IN	Pcompose_type	operation	concatenation operation.
OUT	Pmatrix	matrix	3x3 matrix.

This function computes a matrix to perform the specified 2D shift and concatenates this matrix with `matrix` on the basis of `operation`.

ptk_shift3

void **ptk_shift3** (Ppoint3 *shift, Pcompose_type operation, Pmatrix3 matrix)

IN	Ppoint3 *	shift	shift factor.
IN	Pcompose_type	operation	concatenation operation.
OUT	Pmatrix3	matrix	4x4 matrix.

This function computes a matrix to perform the specified 3D shift and concatenates this matrix with `matrix` on the basis of `operation`.

ptk_stackmatrix

void **ptk_stackmatrix** (Pmatrix matrix)

IN	Pmatrix	matrix	3x3 matrix.

This function pushes the 3×3 matrix `matrix` onto the transformation stack.

ptk_stackmatrix3

void **ptk_stackmatrix3** (Pmatrix3 matrix)

IN	Pmatrix3	matrix	4x4 matrix.

This function pushes the 4×4 matrix `matrix` onto the transformation stack.

ptk_subv

Ppoint **ptk_subv** (Ppoint *p1, Ppoint *p2)

IN	Ppoint *	p1	2D vector.
IN	Ppoint *	p2	2D vector.

This function evaluates the 2D vector `v1-v2`.

ptk_subv3

Ppoint3 **ptk_subv3** (Ppoint3 *p1, Ppoint3 *p2)

IN	Ppoint3 *	p1	3D vector.
IN	Ppoint3 *	p2	3D vector.

This function evaluates the 3D vector `v1-v2`.

ptk_transform

Ppoint **ptk_transform** (Pmatrix matrix, Ppoint *point)

IN	Pmatrix	matrix	3x3 matrix.
IN	Ppoint *	point	2D point.

This function transforms the 2D point `point` by the 3×3 matrix `matrix`.

ptk_transform3

Ppoint3 **ptk_transform3** (Pmatrix3 matrix, Ppoint3 *point)

IN	Pmatrix3	matrix	4x4 matrix.
IN	Ppoint3 *	point	3D point.

This function transforms the 3D point `point` by the 4×4 matrix `matrix`, including homogeneous division.

ptk_transform4

Ppoint4 **ptk_transform4** (Pmatrix3 matrix, Ppoint4 *point)

IN	Pmatrix3	matrix	4x4 matrix.
IN	Ppoint4 *	point	4D point.

This function performs the 4D transformation (that is, with no homogeneous division) of the point `pt` by the 4×4 matrix `matrix`, and returns the result as the value of the function.

ptk_transposematrix

void **ptk_transposematrix** (Pmatrix matrix, Pmatrix result)

IN	Pmatrix	matrix	3x3 matrix.
OUT	Pmatrix	result	3x3 matrix.

This function transposes `matrix`, and returns the resul in `result`. Note that `restut` can be the same variable as `matrix` since a copy is made first.

ptk_transposematrix3

void **ptk_transposematrix3** (Pmatrix3 matrix, Pmatrix3 result)

IN	Pmatrix3	matrix	4x4 matrix.
OUT	Pmatrix3	result	4x4 matrix.

This function transposes `matrix`, and returns the resul in `result`. Note that `restut` can be the same variable as `matrix` since a copy is made first.

ptk_unitmatrix

void **ptk_unitmatrix** (Pmatrix matrix)

IN	Pmatrix	matrix	3x3 matrix.

This procedure creates a unit 3×3 matrix, and stores it in `matrix`.

ptk_unitmatrix3

void **ptk_unitmatrix3** (Pmatrix3 matrix)

IN	Pmatrix3	matrix	4x4 matrix.

This procedure creates a unit 4×4 matrix, and stores it in `matrix`.

ptk_unitv

Ppoint **ptk_unitv** (Ppoint *vec)

IN	Ppoint *	vec	2D vector.

This function generates and returns a unit vector from the supplied 2D vector `vec`.

ptk_unitv3

Ppoint3 **ptk_unitv3** (Ppoint3 *vec)

IN	Ppoint3 *	vec	3D vector.

This function generates and returns a unit vector from the supplied 3D vector `vec`.

ptk_unstackmatrix

void **ptk_unstackmatrix** (Pmatrix matrix)

OUT	Pmatrix	matrix	3x3 matrix.

This function pops a 3×3 matrix from the transformation stack and returns it in `matrix`.

ptk_unstackmatrix3

void **ptk_unstackmatrix3** (Pmatrix3 matrix)

OUT	Pmatrix3	matrix	4x4 matrix.

This function pops a 4×4 matrix from the transformation stack and returns it in `matrix`.

ptk_vec3topt3

Ppoint3 **ptk_vec3topt3** (Pvec3 *vec)

IN	Pvec3 *	vec	3D vector.

This function convert the 3D vector `vec` to a 3D point.

ptk_vector

Pvec **ptk_vector** (Pfloat x, Pfloat y)

IN	Pfloat	x	x coordinate.
IN	Pfloat	y	y coordinate.

This function returns a structure representing the 2D vector `(x,y)`.

ptk_vector3

Pvec3 **ptk_vector3** (Pfloat x, Pfloat y, Pfloat z)

IN	Pfloat	x	x coordinate.
IN	Pfloat	y	y coordinate.
IN	Pfloat	z	z coordinate.

This function returns a structure representing the 3D vector `(x,y,z)`.

Chapter 2

Hashstrings functions

2.1 Data types

```
typedef struct
   {
    Pint listlen;
    Pint *stidlist;
    Pint *strlenlist;
    Pchar **strlist;
    } ptksstringtable;

 typedef struct
   {
    Pint listlen;
    Pint *namelenlist;
    Pchar **tablenames;
   } ptkstablelist;
```

2.2 Functions

ptk_createhashtable

void **ptk_createhashtable** (char *tablestr, Pint minint, Pint maxint)

IN	char *	tablestr	hashtable name.
IN	Pint	minint	lower limit of string-integer range.
IN	Pint	maxint	upper limit of string-integer range.

This function creates a new hashtable, with the name `tablestr`. `minint` and `maxint` respectively specify the lower and upper limits of the range of integers to which strings hashed into the hashtable will be mapped.

ptk_delhashtable

ptkboolean **ptk_delhashtable** (char *tablestr)

IN	char *	tablestr	name of table to be deleted.

This function deletes hastable `tablestr` from the table of hashtables, returning TRUE if the table was deleted, otherwise FALSE.

ptk_delstring

ptkboolean **ptk_delstring** (char *tablestr, char *delstr)

IN	char *	tablestr	name of hashtable.
IN	char *	delstr	string to be deleted from string table.

This function deletes the string `delstr` from hashtable `tablestr`. The result of the function is TRUE if the string was, otherwise FALSE.

ptk_hashtableused

ptkboolean **ptk_hashtableused** (char *str)

IN	char *	str	name of hashtable.

This function checks if a hashtable named `str` already exists in the table of hashtables, returning TRUE if the hashtable exists, otherwise FALSE.

ptk_inithashtables

void **ptk_inithashtables** (void)

This function initialises the table in which the details of hashtables created by the application are kept. This function must be called before any other hashtable functions.

ptk_inqhashtables

void **ptk_inqhashtables** (Pint size, Pint *totalsize, char *strbuffer, ptkstablelist *tablerec)

IN	Pint	size	number of bytes allocated for the hashtable data..
OUT	Pint *	totalsize	actual number of bytes required for hashtable data..
OUT	char *	strbuffer	pointer to buffer for hashtable data.
OUT	ptkstablelist *	tablerec	hash table data..

This function returns in the user specified buffer `strbuffer` the names of each hashtable contained in the table of hashtables.

ptk_inqstrings

void **ptk_inqstrings** (char *tablestr, Pint size, Pint *totalsize, char *strbuffer, ptksstringtable *strtable)

IN	char *	tablestr	name of hashtable.
IN	Pint	size	number of bytes allocated by the user for string table data..
OUT	Pint *	totalsize	actual size of buffer required for string table data..
OUT	char *	strbuffer	application supplied buffer for string table data.
OUT	ptksstringtable *	strtable	struct containing length of arrays and pointer to data in strbuffer..

This function is used to interogate hashtable `tablestr`. A list of the `size` strings contained in the hashtable are returned in the user specified buffer `strbuffer`, together with their corresponding string lengths and integer hash values.

ptk_inttostring

void **ptk_inttostring** (char *tablestr, Pint stint, Pint size, char *strbuffer, Pint *buffersize)

IN	char *	tablestr	hashtable name.
IN	Pint	stint	string identifier to search hashtable for.
IN	Pint	size	number of bytes allocated by the user for the string.
OUT	char *	strbuffer	pointer to space allocated by the user for the string.
OUT	Pint *	buffersize	actual size of buffer required.

This function returns the string in hashtable `tablestr` which has been allocated the integer `stint`. The string is returned in the buffer `strbuffer`, which must be allocated by the application. The number of bytes actually used in the buffer is returned in `buffersize`, as the length of string + 1 (for '
0' character), or 0 if no string was returned.

ptk_restoreallhashtables

void **ptk_restoreallhashtables** (FILE *fileptr)

IN	FILE *	fileptr	pointer to a file.

This function restores all hashtables from a file. Effectively, `ptk_restorehashtable` (q.v.) is called for each hashtable in the file. The file should be open and readable when this function is called.

ptk_restorehashtable

void **ptk_restorehashtable** (FILE *fileptr, char *tablestr)

IN	FILE *	fileptr	pointer to a file.
IN	char *	tablestr	hashtable to insert data from file.

This function reads a hashtable from the file `fileptr`, and creates it with the name `tablestr`. If the hashtable already exists, it is deleted, and then recreated from the file. The file should be open and readable when this function is called.

ptk_storeallhashtables

void **ptk_storeallhashtables** (FILE *fileptr)

IN	FILE *	fileptr	pointer to a file.

The function writes all the hashtables in the table of hashtables to the file `fileptr`, which should be opened and writeable.

ptk_storehashtable

void **ptk_storehashtable** (FILE *fileptr, char *tablestr)

IN	FILE *	fileptr	pointer to a file.
IN	char *	table	hashtable to store.

The function writes the hashtable `tablestr` to the file `fileptr`, which should be opened and writeable.

ptk_stringtoint

Pint **ptk_stringtoint** (char *tablestr, char *str)

IN	char *	tablestr	name of hashtable.
IN	char *	str	string to be searched for in string table.

This function returns the integer allocated for the string `str` in hashtable `tablestr`. If the string has not already been allocated an integer, then it is allocated one and this value is returned.

ptk_stringused

ptkboolean **ptk_stringused** (char *tablestr, char *str)

IN	char *	tablestr	name of hashtable.
IN	char *	str	string to search for in string table.

This function checks if the string `str` has already been used in hashtable `tablestr`, Returning TRUE if string was used in the hashtable, otherwise FALSE.

Chapter 3

PHIGS Utilities Library functions

3.1 Data types

```
typedef struct
   {
    Peltype eltype;
    Pint size;
    Peldata eldata;
   } ptkselcontent;
```

3.2 Functions

ptk_arrow

void **ptk_arrow** (Pfloat length, Pfloat width, Ppoint3 *centre, Pfloat angle
)

IN	Pfloat	length	length of arrow.
IN	Pfloat	width	width of arrow.
IN	Ppoint3 *	centre	centre of arrow.
IN	Pfloat	angle	rotation of arrow in degrees anti-clockwise about arrow pointing along x-axis..

This function draws an arrow with the specified length `length` and `width`, rotated through `angle`, centred at `centre`.

ptk_changepostpriority

void **ptk_changepostpriority** (Pint ws, Pint structid, Prel_pri relpriority, Pint relstruct,
Pint *error)

IN	Pint	ws	workstation identifier.
IN	Pint	structid	structure identifier.
IN	Prel_pri	relpriority	relative priority.
IN	Pint	relstruct	relative structure identifier.
OUT	Pint *	error	error code.

This function changes the priority of structure `structid` to immediately higher or lower than that of `relstruct`, according to `relpriority`. If `structid` does not exist, `error` is set to 1. Otherwise, its value is 0.

ptk_closestruct

void **ptk_closestruct** (void)

This function closes the currently open structure, and restores the open structure and element pointer from the structure stack.

ptk_computecharheight

void **ptk_computecharheight** (Pint wsid, char *str, Ppoint *box, Pint font, Pfloat *charht)

IN	Pint	wsid	workstation identifier.
IN	char *	str	string.
IN	Ppoint *	box	width and height of box.
IN	Pint	font	text font.
OUT	Pfloat *	charht	character height.

This function computes the character height for string `str`, using font `font`, in order for it to fit into the rectangle specifed by `box`.

ptk_computecharsize

void **ptk_computecharsize** (Pint wsid, char *str, Ppoint *box, Pint font, Pfloat *charht, Pfloat *charexp)

IN	Pint	wsid	workstation identifier.
IN	char *	str	string.
IN	Ppoint *	box	width and height of box.
IN	Pint	font	text font.
OUT	Pfloat *	charht	character height.
OUT	Pfloat *	charexp	character expansion.

This function computes the character height and expansion for string `str`, using font `font`, in order for it to fit into the rectangle specifed by `box`.

ptk_copycolourtable

void **ptk_copycolourtable** (Pint sourcewsid, Pint destwsid)

IN	Pint	sourcewsid	source workstation identifier.
IN	Pint	destwsid	destination workstation identifier.

This function copies the colour table from workstation `sourcewsid` to workstation `destwsid`.

ptk_copyedgetable

void **ptk_copyedgetable** (Pint sourcewsid, Pint destwsid)

IN	Pint	sourcewsid	source workstation identifier.
IN	Pint	destwsid	destination workstation identifier.

This function copies the edge bundle table from workstation `sourcewsid` to workstation `destwsid`.

ptk_copyelem

void **ptk_copyelem** (Pint structid, Pint elemid)

IN	Pint	structid	structure identifier.
IN	Pint	elemid	element number.

This function copies the element at position `elemid` in structuyre `structid`, into the currently open structure.

ptk_copyelemrange

void **ptk_copyelemrange** (Pint stid, Pint elem1, Pint elem2)

IN	Pint	stid	structure identifier.
IN	Pint	elem1	element pointer.
IN	Pint	elem2	element pointer.

This function copies the element range `elem1` to `elem2` in structure `stid` into the currently open structure.

ptk_copyhilightfilter

void **ptk_copyhilightfilter** (Pint sourcewsid, Pint destwsid)

IN	Pint	sourcewsid	source workstation identifier.
IN	Pint	destwsid	destination workstation identifier.

This function copies the highlighting filter from workstation `sourcewsid` to workstation `destwsid`.

ptk_copyhlhsrmode

void **ptk_copyhlhsrmode** (Pint sourcewsid, Pint destwsid)

IN	Pint	sourcewsid	source workstation identifier.
IN	Pint	destwsid	destination workstation identifier.

This function copies the HLHSR mode from workstation `sourcewsid` to workstation `destwsid`.

ptk_copyinttable

void **ptk_copyinttable** (Pint sourcewsid, Pint destwsid)

IN	Pint	sourcewsid	source workstation identifier.
IN	Pint	destwsid	destination workstation identifier.

This function copies the interior bundle table from workstation `sourcewsid` to workstation `destwsid`.

ptk_copyinvisfilter

void **ptk_copyinvisfilter** (Pint sourcewsid, Pint destwsid)

IN	Pint	sourcewsid	source workstation identifier.
IN	Pint	destwsid	destination workstation identifier.

This function copies the invisibilty filter from workstation `sourcewsid` to workstation `destwsid`.

ptk_copylinetable

void **ptk_copylinetable** (Pint sourcewsid, Pint destwsid)

IN	Pint	sourcewsid	source workstation identifier.
IN	Pint	destwsid	destination workstation identifier.

This function copies the polyline bundle table from workstation `sourcewsid` to workstation `destwsid`.

ptk_copymarkertable

void **ptk_copymarkertable** (Pint sourcewsid, Pint destwsid)

IN	Pint	sourcewsid	source workstation identifier.
IN	Pint	destwsid	destination workstation identifier.

This function copies the polymarker bundle table from workstation `sourcewsid` to workstation `destwsid`.

ptk_copypattable

void **ptk_copypattable** (Pint sourcewsid, Pint destwsid)

IN	Pint	sourcewsid	source workstation identifier.
IN	Pint	destwsid	destination workstation identifier.

This function copies the pattern bundle table from workstation `sourcewsid` to workstation `destwsid`.

ptk_copypostedstruct

void **ptk_copypostedstruct** (Pint sourcewsid, Pint destwsid)

IN	Pint	sourcewsid	source workstation identifier.
IN	Pint	destwsid	destination workstation identifier.

This function posts all the structures already posted to workstation `sourcewsid` to workstation `destwsid`.

ptk_copytexttable

void **ptk_copytexttable** (Pint sourcewsid, Pint destwsid)

IN	Pint	sourcewsid	source workstation identifier.
IN	Pint	destwsid	destination workstation identifier.

This function copies the text bundle table from workstation `sourcewsid` to workstation `destwsid`.

ptk_copyviewtable

void **ptk_copyviewtable** (Pint sourcewsid, Pint destwsid)

IN	Pint	sourcewsid	source workstation identifier.
IN	Pint	destwsid	destination workstation identifier.

This function copies the view table from workstation `sourcewsid` to workstation `destwsid`.

ptk_copywssttable

void **ptk_copywssttable** (Pint sourcewsid, Pint destwsid)

IN	Pint	sourcewsid	source workstation identifier.
IN	Pint	destwsid	destination workstation identifier.

This function copies the workstation colour, polyline bundle, polymarker bundle, interior bundle, edge bundle, text bundle, pattern bundle and view tables from workstation `sourcewsid` to workstation `destwsid`.

ptk_countchildren

Pint **ptk_countchildren** (Pint stid)

IN	Pint	stid	structure identifier.

The function returns the number of structures executed by structure `stid`. This is the number of direct execute elements in the structure.

ptk_countuniqchildren

Pint **ptk_countuniqchildren** (Pint stid)

IN	Pint	stid	structure identifier.

This function returns the number of unique children of structure `stid`. Thus, if a structure executed structures A, B and C, it would have 3 unique children, regardless of how many times each of A, B and C were executed.

ptk_delelem

void **ptk_delelem** (Pint numelems)

IN	Pint	numelems	number of elements to delete.

This function deletes the `numelms` elements from the open structure, starting at the element pointer. If `numelems` is 0, all elements up to the end of the structure are deleted.

ptk_delelemtype

void **ptk_delelemtype** (Pint stid, Pint lenlst, Pelem_type *eltypelst)

IN	Pint	stid	structure identifier.
IN	Pint	lenlst	length of element type list.
IN	Pelem_type *	eltypelst	list of element types to delete.

This function deletes every element in structure `stid`, whose type matches one of the types in `eltypelst`

ptk_drawcolourtable

void **ptk_drawcolourtable** (Pint stid, Pint llim, pint ulim)

IN	Pint	stid	structure identifier.
IN	Pint	llim	lower index value of colour table range.
IN	Pint	ulim	upper limit of colour table range..

This function draws a rectangular array of boxes representing the range `llim` to `ulim` of the workstation colour table. The boxes are drawn into structure `stid`.

ptk_elemcount

Pint **ptk_elemcount** (Pint stid)

IN	Pint	stid	structure identifier.

This functions counts the number of elements in structure `stid`, returning the number of elements, or -1 if the structure does not exist.

ptk_executeelem

void **ptk_executeelem** (ptkselcontent *elcont)

IN	ptkselcontent *	elcont	element content.

This function creates a structure element which has the contents `elcont`. The element is inserted into the currently open structure.

ptk_findelemtype

void **ptk_findelemtype** (Pelem_type *eltypelst, Pint lenlst, Psearch_dir srchdir,
Psearch_status *srchstat, Pint *elptr, Pint *lstnum)

IN	Pelem_type *	eltypelst	list of element types.
IN	Pint	lenlst	length of element type list.
IN	Psearch_dir	srchdir	search direction, forwards or backwards.
OUT	Psearch_status *	srchstat	search success or failure.
OUT	Pint *	elptr	found element pointer.
OUT	Pint *	lstnum	index of found item in list.

This function searches the currently open structure, starting at the current element pointer and proceeding in direction `srchdir`, for the first element whose type matches any of those given in `contentlst`.

ptk_findlabel

ptkboolean **ptk_findlabel** (Pint label, Pint *elemptr)

IN	Pint	label	label value.
IN/OUT	Pint *	elemptr	element pointer of label element.

This function searches forwards through the currently open structure from the current editing position looking for a label structure element. The function returns TRUE if `label` was found, otherwise FALSE.

ptk_findnextpickid

ptkboolean **ptk_findnextpickid** (Pint stid, Psearch_dir srchdir, Pint *eltptr, Pint *pickid)

IN	Pint	stid	structure identifier.
IN	Psearch_dir	srchdir	search direction.
OUT	Pint *	eltptr	element pointer of pick identifier.
OUT	Pint *	pickid	pick identifier value.

This function searches structure `stid`, starting at element `elptr` and proceeding in direction `srchdir`, looking for a pick identifier structure element. The function returns TRUE if a pick identifier was found, otherwise FALSE.

ptk_framebox

void **ptk_framebox** (Ppoint3 *boxcentre, Ppoint *boxsize, Ppoint *framesize,
Pint boxcolour, Pint edgecolour, Pint tlcolour, Pint brcolour
)

IN	Ppoint3 *	boxcentre	centre of box.
IN	Ppoint *	boxsize	height and width box.
IN	Ppoint *	framesize	height and width of frame.
IN	Pint	boxcolour	box interior colour index.
IN	Pint	edgecolour	box edge colour index.
IN	Pint	tlcolour	frame top-left colour index.
IN	Pint	brcolour	frame bottom-right colour index.

This function draws a box in the open structure with a frame to give a 3D effect.

ptk_getelemtype

void **ptk_getelemtype** (char *elemstr, Pelem_type *eltype)

IN	char *	elemstr	string giving element type.
OUT	Pelem_type *	eltype	element type.

This function converts the string `elemstr` into its corresponding element type. For example the element type for "polyline3", would be PEL_POLYLINE3.

ptk_getelemtypename

void **ptk_getelemtypename** (Pelem_type eltype, Pint size, char *elemstr, Pint *totalsize)

IN	Pelem_type	eltype	element type.
IN	Pint	size	size of buffer allocated by application.
OUT	char *	elemstr	string giving element type.
OUT	Pint *	totalsize	length of string.

This function converts element type `eltype` into the corresponding character string, which is returned in `elemstr`. For example, the string corresponding to PE_POLYLINE3 would be "polyline3".

ptk_getexecuteid

ptkboolean **ptk_getexecuteid** (Pint stid, Pint elptr, Pint *execid)

IN	Pint	stid	structure identifier.
IN	Pint	elptr	element pointer value.
OUT	Pint *	execid	execute structure identifier.

This function inquires the execute element identifier at the element position specified by `elptr` in structure `stid`. The result of the function is TRUE if the element was an execute element, otherwise FALSE..

ptk_getpickid

ptkboolean **ptk_getpickid** (Pint stid, Pint elptr, Pint *pickid)

IN	Pint	stid	structure identifier.
IN	Pint	elptr	element pointer value.
OUT	Pint *	pickid	pick identifier.

This function inquires the pick identifier at the element position specified by `elptr` in structure `stid`. The result of the function is TRUE if the element was a pick identifier, otherwise FALSE.

ptk_getprimitivetypename

void **ptk_getprimitivetypename** (Pattrs attr, Pint size, char *attrstr, Pint *totalsize)

IN	Pattrs	attr	primitive type.
IN	Pint	size	size of buffer allocated by application.
OUT	char *	attrstr	string giving primitive type.
OUT	Pint *	totalsize	length of string.

The function converts the primitive type `attr` to its corresponding character string, which is returned in `artrstr`. For example, P_POLYLINE would give "polyline".

ptk_grid

void **ptk_grid** (Pint stid)

IN	Pint	stid	grid structure identifier.

This functions draws a grid of lines on [0,1], into structure `stid`.

ptk_inqmaxdevicecoords

void **ptk_inqmaxdevicecoords** (Pint wsid, Pfloat *maxdevx, Pfloat *maxdevy)

IN	Pint	wsid	workstation identifier.
OUT	Pfloat *	maxdevx	maximum device coords along x axis.
OUT	Pfloat *	maxdevy	maximum device coords along y axis.

This function returns the the maximum device coordinates for *x* and *y* for workstation `wsid`.

ptk_inqmaxdevicecoords3

void **ptk_inqmaxdevicecoords3** (Pint wsid, Pfloat *maxdevx, Pfloat *maxdevy, Pfloat *maxdevz)

IN	Pint	wsid	workstation identifier.
OUT	Pfloat *	maxdevx	maximum device coords along x axis.
OUT	Pfloat *	maxdevy	maximum device coords along y axis.
OUT	Pfloat *	maxdevz	maximum device coords along z axis.

This function returns the the maximum device coordinates for x, y and z for workstation `wsid`.

ptk_inqpostpriority

void **ptk_inqpostpriority** (Pint wsid, Pint structid, Pfloat *priority, Pint *err)

IN	Pint	wsid	workstation identifier.
IN	Pint	structid	structure identifier.
IN	Pfloat *	priority	priority value.
OUT	Pint *	err	error indicator.

This function may be used to obtain the priority of posted structure `structid`. If `structid` does not exist, `err` is set to 1. If it is not posted to `wsid` then `err` is set to 2.

ptk_inqstructnetids

void **ptk_inqstructnetids** (Pint root, Pint size, Pint_list *stids, Pint *totalsize)

IN	Pint	root	structure network identifer.
IN	Pint	size	number of integers allocated in integer list.
OUT	Pint_list *	stids	list of structure identifiers.
OUT	Pint *	totalsize	actual number of integers in integer list.

This function returns a list of structure identifiers in the structure network whose root is `root`.

ptk_openstruct

void **ptk_openstruct** (Pint structid)

IN	Pint	structid	structure identifier.

This function stores the currently open structure and element pointer on a stack and opens the structure `structid`.

ptk_postrelative

void **ptk_postrelative** (Pint ws, Pint structid, Prel_pri relpriority, Pint relstruct, Pint *error)

IN	Pint	ws	workstation identifier.
IN	Pint	structid	structure identifier.
IN	Prel_pri	relpriority	relative priority.
IN	Pint	relstruct	relative structure identifier.
OUT	Pint *	error	error code.

This function posts structure `structid` at a priority higher or lower than that of structure `relative structure identifier`, according to `relative priority`. If `relative structure identifier` does not exist, `error` is set to 1. Otherwise, its value is 0.

ptk_poststruct

void **ptk_poststruct** (Pint wsid, Pint stid, Pfloat priority)

IN	Pint	wsid	workstation identifier.
IN	Pint	stid	structure identifier.
IN	Pfloat	priority	priority with which to post structure.

This function posts structure `stid` to workstation `wsid`, but only if the structure exists.

ptk_readfloat

Pfloat **ptk_readfloat** (Pint ws, Pfloat defreal, char *prompt, Plimit *echoarea)

IN	Pint	ws	workstation identifier.
IN	Pfloat	defreal	default floating point number.
IN	char *	prompt	prompt string.
IN	Plimit *	echoarea	string echo area.

This function reads and returns a real number from string device 1 on workstation `ws`, using `prompt` as a prompt string. If the user types a carriage-return instead of supplying an real value, the function returns the specified default value `defreal`. `echoarea` specifies the echo area to use for the string device.

ptk_readint

Pint **ptk_readint** (Pint ws, Pint defint, char *prompt, Plimit *echoarea)

IN	Pint	ws	workstation identifier.
IN	Pint	defint	default integer.
IN	char *	prompt	prompt string.
IN	Plimit *	echoarea	string echo area.

This function reads and returns an integer from string device 1 on workstation `ws`, using `prompt` as a prompt string. If the user types a carriage-return instead of supplying an integer, the function returns the specified default value `defint`. `echoarea` specifies the echo area to use for the string device.

ptk_readstring

void **ptk_readstring** (Pint ws, char *defstring, char *prompt, Plimit *echoarea, Pint len, char *instr, Pint *inlen)

IN	Pint	ws	workstation identifier.
IN	char *	defstring	default string.
IN	char *	prompt	prompt string.
IN	Plimit *	echoarea	string echo area.
IN	Pint	len	number of characters allocated for input string.
OUT	char *	instr	input string.
OUT	Pint *	inlen	length of input string.

This function reads and returns a real number from string device 1 on workstation `ws`, using `prompt` as a prompt string. If the user types a carriage-return instead of supplying a string, the function returns the specified default value `defstring`. `echoarea` specifies the echo area to use for the string device.

ptk_redrawallstructs

void **ptk_redrawallstructs** (Pint wsid, Pctrl_flag flag)

IN	Pint	wsid	workstation identifier.
IN	Pctrl_flag	flag	control flag.

This function calls REDRAW ALL STRUCTURES only if the visual state of the workstation is deferred.

ptk_removestruct

ptkboolean **ptk_removestruct** (Pint stid)

IN	Pint	stid	structure identifier.

This function deletes structure `stid`, if it already exists, returning TRUE if the structure was deleted, otherwise FALSE.

ptk_setallattrasf

void **ptk_setallattrasf** (Pasf asf)

IN	Pasf	asf	aspect source flag.

This function inserts a structure element into the open structure to set the aspect source flags for all primitive attribiutes, according to `asf`.

ptk_setattrasf

void **ptk_setattrasf** (Pint numattrs, Paspect *attrs, Pasf asf)

IN	Pint	numattrs	length of attribute list.
IN	Paspect *	attrs	attribute list.
IN	Pasf	asf	aspect source flag.

This function inserts a structure element into the open structure to set the aspect source flags for each of the `numattrs` attributes in the list `attrs`, according to `asf`.

ptk_setedgebundlerec

Pedge_bundle **ptk_setedgebundlerec** (Pedge_flag eflag, Pint etype, Pfloat escale, Pint ecolour)

IN	Pedge_flag	eflag	edge on or off.
IN	Pint	etype	edge type (solid, dotted,..).
IN	Pfloat	escale	edge width scale factor.
IN	Pint	ecolour	edge colour index.

This function sets creates and returns an edge bundle attribute record whose fields are set as specified.

ptk_seteditmode

void **ptk_seteditmode** (Pedit_mode editmode)

IN	Pedit_mode	editmode	edit mode.

This function stores the current edit mode on the editmode stack and then sets the edit mode to given value.

ptk_setinteriorbundlerec

Pint_bundle **ptk_setinteriorbundlerec** (Pint_style istyle, Pint index, Pint icolour)

IN	Pint_style	istyle	interior style (solid, hollow,..).
IN	Pint	index	interior style index for pattern or hatch.
IN	Pint	icolour	interior colour index.

This function sets creates and returns an interior bundle attribute record whose fields are set as specified.

ptk_setpolylinebundlerec

Pline_bundle **ptk_setpolylinebundlerec** (Pint ltype, Pfloat lwidth, Pint lcolour)

IN	Pint	ltype	line type (solid, dotted,..).
IN	Pfloat	lwidth	line width scale factor.
IN	Pint	lcolour	line colour index.

This function sets creates and returns a polyline bundle attribute record whose fields are set as specified.

ptk_setpolymarkerbundlerec

Pmarker_bundle **ptk_setpolymarkerbundlerec** (Pint mtype, Pfloat msize, Pint mcolour)

IN	Pint	mtype	marker type (dot, circle,..).
IN	Pfloat	msize	marker size scale factor.
IN	Pint	mcolour	marker colour index.

This function sets creates and returns an polymarker bundle attribute record whose fields are set as specified.

ptk_setstandardviewport

void **ptk_setstandardviewport** (Plimit3 *vlimits, Pview_rep3 *rep)

IN	Plimit3 *	vlimits	viewport bounding box.
OUT	Pview_rep3 *	rep	view representation.

For a window of [0,1], this function creates in `rep` a view representation for a viewport of `vlimits`.

ptk_settextbundlerec

Ptext_bundle **ptk_settextbundlerec** (Pint tfont, Ptext_prec tprec, Pfloat texp, Pfloat tspace, Pint tcolour)

IN	Pint	tfont	text font.
IN	Ptext_prec	tprec	text precision.
IN	Pfloat	texp	character expansion factor.
IN	Pfloat	tspace	character spacing.
IN	Pint	tcolour	text colour index.

This function sets creates and returns a text bundle attribute record whose fields are set as specified.

ptk_stackstruct

void **ptk_stackstruct** (void)

This function stores the name of the currently open structure, and the position of the element pointer on the structure stack, and closes the structure

ptk_structexists

ptkboolean **ptk_structexists** (Pint stid)

IN	Pint	stid	structure identifier.

This function checks if structure `stid` exists in the CSS, returning TRUE if it exists, otherwise FALSE.

ptk_structsummary

void **ptk_structsummary** (FILE *fileptr)

IN	FILE *	fileptr	file pointer.

This function outputs a summary of all the structures in the CSS to file `fileptr`, which should be an open writeable file. The structure identifier of each structure is printed, together with its hashed string name, if it has one. The format of the list is:

```
List of structures in the CSS
-----------------------------

structure 1 "helicopter"
structure 45
structure 51 "helicopter"
etc.  .
```

ptk_unseteditmode

void **ptk_unseteditmode** (void)

This function restores the current edit mode from the edit mode stack.

ptk_unstackstruct

void **ptk_unstackstruct** (void)

This function pops the structure stack, opens the structure and sets the element pointer.

Chapter 4

Traversal State List functions

ptk_boundingbox

ptkboolean **ptk_boundingbox** (Pint structid, Plimit3 *wcbounds, ptkboolean descend)

IN	Pint	structid	structure network identifier.
OUT	Plimit3 *	wcbounds	bounding box in world coordinates.
IN	ptkboolean	descend	flag to indicate traversal.

This function evaluates the bounding box for a structure or structure network. If `descend` is set to TRUE then the bounding box for the complete structure network is returned, otherwise just for a single structure. The function returns TRUE if the resulting bounding box is valid, otherwise FALSE. A structure with no output primitives returns an invalid bounding box.

ptk_initbbox

void **ptk_initbbox** (void)

This function initialises the current TSL bounding box value to be (1000.0, -1000.0, 1000.0, -1000.0, 1000.0, -1000.0). This enables the real bounding box to be set when an output primitive is reached.

ptk_inittsl

void **ptk_inittsl** (void)

This function initialises the current TSL values to the default values from the PHIGS description table. The default ASF value for each attribute is assumed to be INDIVIDUAL.

ptk_inqboundingbox

void **ptk_inqboundingbox** (Plimit3 *bbox)

OUT	Plimit3 *	bbox	bounding box (WC).

This function may be used to obtain current TSL bounding box value. This is the bounding box of the TSL structure network at the current point of traversal.

ptk_inqtslannotext

void **ptk_inqtslannotext** (Pint *style, Pfloat *charheight, Ppoint *charup, Ptext_align *textalign, Ptext_path *textpath)

OUT	Pint *	style	annotation style.
OUT	Pfloat *	charheight	annotation character height.
OUT	Ppoint *	charup	annotation character up vector.
OUT	Ptext_align *	textalign	annotation text alignment.
OUT	Ptext_path *	textpath	annotation text path.

This function may be used to obtain the current TSL annotation text attributes.

ptk_inqtslattrasf

void **ptk_inqtslattrasf** (Paspect attr, Pasf *asf)

IN	Paspect	attr	attribute type.
OUT	Pasf *	asf	attribute aspect source flag.

This function may be used to obtain the current TSL aspect source flag value for attribute type `attr`.

ptk_inqtslchar

void **ptk_inqtslchar** (Pfloat *exp, Pfloat *spacing, Pfloat *height, Ppoint *charup)

OUT	Pfloat *	exp	character expansion factor.
OUT	Pfloat *	spacing	character spacing.
OUT	Pfloat *	height	character height.
OUT	Ppoint *	charup	character up vector.

This function may be used to obtain the current TSL character attributes.

ptk_inqtslctm

void **ptk_inqtslctm** (Pmatrix3 globaltran, Pmatrix3 localtran)

OUT	Pmatrix3	globaltran	global transformation matrix.
OUT	Pmatrix3	localtran	local transformation matrix.

This function may be used to obtain the current TSL transformation matrices.

ptk_inqtsledge

void **ptk_inqtsledge** (Pint *edgeind, Pedge_flag *edgeflag, Pint *edgetype, Pfloat *edgewidth, Pint *edgecolour)

OUT	Pint *	edgeind	edge index.
OUT	Pedge_flag *	edgeflag	edge flag.
OUT	Pint *	edgetype	edge type.
OUT	Pfloat *	edgewidth	edge width.
OUT	Pint *	edgecolour	edge colour.

This function may be used to obtain the current TSL edge attributes.

ptk_inqtslids

void **ptk_inqtslids** (Pint *pickid, Pint *hlhsrid, Pint *viewind)

OUT	Pint *	pickid	pick identifier.
OUT	Pint *	hlhsrid	HLHSR identifier.
OUT	Pint *	viewind	view index.

This function may be used to obtain the current TSL values for pick identifier, HLHSR identifier and view index.

ptk_inqtslinterior

void **ptk_inqtslinterior** (Pint *intind, Pint *intstyle, Pint *intstyleind, Pint *intcolour)

OUT	Pint *	intind	interior index.
OUT	Pint *	intstyle	interior style.
OUT	Pfloat *	intstyleind	interior style index.
OUT	Pint *	intcolour	interior colour.

This function may be used to obtain the current TSL interior attributes.

ptk_inqtslline

void **ptk_inqtslline** (Pint *lineind, Pint *linetype, Pfloat *linewidth, Pint *line-colour)

OUT	Pint *	lineind	line index.
OUT	Pint *	linetype	line type.
OUT	Pfloat *	linewidth	line width.
OUT	Pint *	linecolour	line colour.

This function may be used to obtain the current TSL line attributes.

ptk_inqtslmarker

void **ptk_inqtslmarker** (Pint *markerind, Pint *markertype, Pfloat *markersize, Pint *markercolour)

OUT	Pint *	markerind	marker index.
OUT	Pint *	markertype	marker type.
OUT	Pfloat *	markersize	marker size.
OUT	Pint *	markercolour	marker colour.

This function may be used to obtain the current TSL marker attributes.

ptk_inqtslnameset

void **ptk_inqtslnameset** (Pint size, Pint_list *nameset, Pint *totalsize)

IN	Pint	size	length of nameset integer list.
OUT	Pint_list *	nameset	current nameset.
OUT	Pint *	totalsize	actual length of nameset integer list.

This function may be used to obtain the current TSL nameset.

ptk_inqtslpattern

void **ptk_inqtslpattern** (Pfloat_size *size, Ppoint3 *refpt, Ppoint3 *refvec)

OUT	Pfloat_size *	size	pattern size.
OUT	Ppoint3 *	refpt	pattern reference point.
OUT	Ppoint3 *	refvec	pattern reference vectors.

This function may be used to obtain the current TSL pattern attributes.

ptk_inqtsltext

void **ptk_inqtsltext** (Pint *textind, Pint *textfont, Ptext_prec *textprec, Ptext_path *textpath, Ptext_align *textalign, Pint *textcolour)

OUT	Pint *	textind	text index.
OUT	Pint *	textfont	text font.
OUT	Ptext_prec *	textprec	text precision.
OUT	Ptext_path *	textpath	text path.
OUT	Ptext_align *	textalign	text alignment.
OUT	Pint *	textcolour	text colour.

This function may be used to obtain the current TSL text attributes.

ptk_settslstart

void **ptk_settslstart** (Pint startstid, Pint startelemid)

IN	Pint	startstid	start structure identifier.
IN	Pint	startelemid	start element number.

This function sets the TSL traversal to start at a certain point in a structure network. This structure is referred to as the TSL structure network.

ptk_stackbbox

void **ptk_stackbbox** (void)

This function puts the current bounding box value on the TSL bounding box stack.

ptk_stacktsl

void **ptk_stacktsl** (void)

This function puts the current TSL value on the TSL stack.

ptk_tsltraversenext

void **ptk_tsltraversenext** (void)

This function makes TSL traverse the next element in the current TSL structure network. If it is an EXECUTE STRUCTURE element TSL will follow the execute path.

ptk_tsltraversepath

void **ptk_tsltraversepath** (Pelem_ref_list *reflst)

IN	Pelem_ref_list *	reflst	structure-element path.

This function makes TSL traverse a structure path and updates the current TSL values appropriately.

ptk_tsltraverserange

void **ptk_tsltraverserange** (Pint startstid, Pint startelemid, Pint stopstid, Pint stopelemid, ptkboolean descend)

IN	Pint	startstid	start structure identifier.
IN	Pint	startelemid	start element number.
IN	Pint	stopstid	stop structure identifier.
IN	Pint	stopelemid	stop element number.
IN	ptkboolean	descend	flag to indicate traversal.

This function makes TSL traverse between two points in a structure network. If `descend` is set to TRUE then any EXECUTE STRUCTURE elements which occur between the two points are followed, otherwise they are ignored.

ptk_tsltraversetoend

void **ptk_tsltraversetoend** (void)

This function makes TSL traverse to the end of the TSL structure network.

ptk_unstackbbox

void **ptk_unstackbbox** (void)

This function gets the topmost bounding box value from the TSL bounding box stack.

ptk_unstacktsl

void **ptk_unstacktsl** (void)

This function gets the topmost TSL value from the TSL stack.

Chapter 5

Colour functions

5.1 Data types

```
typedef enum
{
  PTKEVERYDARK,
  PTKEDARK,
  PTKEMEDIUM,
  PTKELIGHT,
  PTKEVERYLIGHT
} ptkelightness;

typedef enum
{
  PTKEGREYISH,
  PTKEMODERATE,
  PTKESTRONG,
  PTKEVIVID
} ptkesaturation;
```

5.2 Functions

ptk_cnstohsl

ptkboolean **ptk_cnstohsl** (char *colourname, Phls *hsl)

IN	char *	colourname	colour description.
IN	Phls *	hsl	HSL triplet.

This function converts a CNS colour name to the equivalent HSL value, returning TRUE if the conversion was successful, and FALSE if not.

ptk_cnstohsv

ptkboolean **ptk_cnstohsv** (char *colourname, Phsv *hsv)

IN	char *	colourname	colour description.
IN	Phsv *	hsv	HSV triplet.

This function Converts colour name to HSV. Returns TRUE if ok, FALSE if not ok.

ptk_cnstorgb

ptkboolean **ptk_cnstorgb** (char *colourname, Prgb *rgb)

IN	char *	colourname	colour description.
IN	Prgb *	rgb	RGB triplet.

This function converts a CNS colour name to the equivalent RGB value, returning TRUE if the conversion was successful, and FALSE if not.

ptk_hsltorgb

void **ptk_hsltorgb** (Phls *hsl, Prgb *rgb)

IN	Phls *	hsl	HSL triplet.
IN	Prgb *	rgb	RGB triplet.

This function converts from the HSL double-ended hexcone model to the RGB model. Given HSL, the equivalent RGB parameters are computed. All parameters are assumed to be in the range 0.0 to 1.0. The algorithm is adapted from [**?**].

ptk_hsvtorgb

void **ptk_hsvtorgb** (Phsv *hsv, Prgb *rgb)

IN	Phsv *	hsv	HSV triplet.
IN	Prgb *	rgb	RGB triplet.

This function converts a HSV triplet to a RGB triplet. The algorithm is adapted from [**?**].

ptk_inqcnsdefaults

void **ptk_inqcnsdefaults** (ptkelightness *lightness, ptkesaturation *saturation)

OUT	ptkelightness *	lightness	default lightness for colours.
OUT	ptkesaturation *	saturation	default saturation for colours.

This function inquires the default values of lightness and saturation used in the Colour Naming Scheme.

ptk_rgbtohsl

void **ptk_rgbtohsl** (Prgb *rgb, Phls *hsl)

IN	Prgb *	rgb	RGB triplet.
IN	Phls *	hsl	HSL triplet.

This function converts an RGB triplet to a HSL triplet. The algorithm is adapted from [**?**].

ptk_rgbtohsv

void **ptk_rgbtohsv** (Prgb *rgb, Phsv *hsv)

IN	Prgb *	rgb	RGB triplet.
IN	Phsv *	hsv	HSV triplet.

This function converts an RGB value to a HSV value. The algorithm is adapted from [**?**].

ptk_setbackgroundcolour

void **ptk_setbackgroundcolour** (Pint wsid, char *colourname)

IN	Pint	wsid	workstation identifier.
IN	char *	colourname	colour name.

This function sets the colour representation of the zeroth entry in the colour table of workstation `wsid`, to be that specified by `colourname` in the CNS.

ptk_setbackgroundcolourind

void **ptk_setbackgroundcolourind** (Pint wsid, Pint index)

IN	Pint	wsid	workstation identifier.
IN	Pint	index	colour index.

This function sets the colour representation of the zeroth entry in the colour table of workstation `wsid`, to be same as the entry `index` in the colour table.

ptk_setcnsdefaults

void **ptk_setcnsdefaults** (ptkelightness lightness, ptkesaturation saturation)

IN	ptkelightness	lightness	default lightness for colours.
IN	ptkesaturation	saturation	default saturation for colours.

This function sets default values for lightness and saturation for the Colour Naming Scheme. If lightness or saturation is missing when a colour name is subsequently specified, the default is used.

ptk_setcolourrep

void **ptk_setcolourrep** (Pint wsid, char *colourname)

IN	Pint	wsid	workstation identifier.
IN	char *	colourname	colour name.

This function sets colour representations in the colour table of workstation `wsid`, using the colour name `colourname`. The hashstrings table `"colourindex"` is used to derive the index to the colour table.

ptk_setedgecolour

void **ptk_setedgecolour** (Pint wsid, char *colourname)

IN	Pint	wsid	workstation identifier.
IN	char *	colourname	colour name.

This function sets the edge colour index to be that specified by the given colour name in the `"colourindex"` hashtable. The colour representation is set in the workstation colour table if necessary.

ptk_setintcolour

void **ptk_setintcolour** (Pint wsid, char *colourname)

IN	Pint	wsid	workstation identifier.
IN	char *	colourname	colour name.

This function sets the interior colour index to be that specified by the given colour name in the `"colourindex"` hashtable. The colour representation is set in the workstation colour table if necessary.

ptk_setlinecolour

void **ptk_setlinecolour** (Pint wsid, char *colourname)

IN	Pint	wsid	workstation identifier.
IN	char *	colourname	colour name.

This function sets the polyline colour index to be that specified by the given colour name in the `"colourindex"` hashtable. The colour representation is set in the workstation colour table if necessary.

ptk_setmarkercolour

void **ptk_setmarkercolour** (Pint wsid, char *colourname)

IN	Pint	wsid	workstation identifier.
IN	char *	colourname	colour name.

This function sets the polymarker colour index to be that specified by the given colour name in the `"colourindex"` hashtable. The colour representation is set in the workstation colour table if necessary.

ptk_setrgbcolourname

void **ptk_setrgbcolourname** (char *colourname, Prgb *rgb)

IN	char *	colourname	colour name.
IN	Prgb *	rgb	RGB colour value.

This function sets a colour representation in CNS using the colour name and RGB value. The colour name must be different to the names provided by the CNS. This function enables additional names for colours to be specified in addition to those provided by CNS.

ptk_settextcolour

void **ptk_settextcolour** (Pint wsid, char *colourname)

IN	Pint	wsid	workstation identifier.
IN	char *	colourname	colour name.
IN	Pint	index	colour index.

This function sets the text colour index to be that specified by the given colour name in the `"colourindex"` hashtable. The colour representation is set in the workstation colour table if necessary.

ptk_setupcolourtable

void **ptk_setupcolourtable** (Pint wsid, Pint num, char **colournames)

IN	Pint	wsid	workstation identifier.
IN	Pint	num	number of colour names.
IN	char **	colournames	list of colour names.

This function sets colour representations in the colour table of workstation `wsid`, using the list of colour names `colournames`. The hashstrings table `"colourindex"` is used to derive the index to the colour table.

Chapter 6

Miscellaneous functions

ptk_fillareaset

void **ptk_fillareaset** (Pint numsets, Ppoint_list *sets)

IN	Pint	numsets	number of point sets.
IN	Ppoint_list *	sets	list of point sets.

This function creates a FILL AREA SET output primitive and works for both SunPHIGS and DEC PHIGS.

ptk_fillareaset3

void **ptk_fillareaset3** (Pint numsets, Ppoint_list3 *sets)

IN	Pint	numsets	number of point sets.
IN	Ppoint_list3 *	sets	list of point sets.

This function creates a FILL AREA SET3 output primitive and works for both SunPHIGS and DEC PHIGS.

ptk_inqcurelemtype

void **ptk_inqcurelemtype** (Pint *error, Pelem_type *eltype)

OUT	Pint *	error	error indicator.
OUT	Pelem_type *	eltype	element type.

This function may be used to obtain the current element type and works for both SunPHIGS and DEC PHIGS.

ptk_inqcurelemtypesizecontent

void **ptk_inqcurelemtypesizecontent** (Pstore store, Pint *error, ptkselcontent *elcontent)

OUT	Pstore	store	data buffer.
OUT	Pint *	error	error indicator.
OUT	ptkselcontent *	elcontent	element content data record.

This function may be used to obtain the type, size and contents of the current element. The data buffer is used to store the element contents data and is allocated by the function. The buffer should be deallocated by the application when the element data is no longer required. This function works with both SunPHIGS and DEC PHIGS.

ptk_inqelemtype

void **ptk_inqelemtype** (Pint stid, Pint elemid, Pint *error, Pelem_type *eltype)

IN	Pint	stid	structure identifier.
IN	Pint	elemid	element number.
OUT	Pint *	error	error indicator.
OUT	Pelem_type *	eltype	element type.

This function may be used to obtain the element type of element `elemid` in structure `stid` and works for both SunPHIGS and DEC PHIGS.

ptk_inqelemtypesizecontent

void **ptk_inqelemtypesizecontent** (Pint stid, Pint elemid, Pstore store, Pint *error, ptkselcontent *elcontent)

IN	Pint	stid	structure identifier.
IN	Pint	elemid	element number.
OUT	Pstore	store	data buffer.
OUT	Pint *	error	error indicator.
OUT	ptkselcontent *	elcontent	element content data record.

This function may be used to obtain the type, size and contents of element `elemid` in structure `stid`. The data buffer is used to store the element contents data and is allocated by the function. The buffer should be deallocated by the application when the element data is no longer required. This function works with both SunPHIGS and DEC PHIGS.

ptk_pbuildtran

void **ptk_pbuildtran** (Ppoint *pt, Ppoint *shift, Pfloat angle, Ppoint *scale, Pint *err, Pmatrix mat)

IN	Ppoint *	pt	fixed point.
IN	Ppoint *	shift	shift vector.
IN	Pfloat	angle	rotation angle.
IN	Ppoint *	scale	scale vector.
OUT	Pint *	err	error indicator.
OUT	Pmatrix	mat	3x3 matrix.

This function performs the BUILD TRANSFORMATION MATRIX function using the `Ppoint` datatype instead of `Pvec`.

ptk_pbuildtran3

void **ptk_pbuildtran3** (Ppoint3 *pt, Ppoint3 *shift, Pfloat xangle, Pfloat yangle, Pfloat zangle, Ppoint3 *scale, Pint *err, Pmatrix3 mat)

IN	Ppoint3 *	pt	fixed point.
IN	Ppoint3 *	shift	shift vector.
IN	Pfloat	xangle	x rotation angle.
IN	Pfloat	yangle	y rotation angle.
IN	Pfloat	zangle	z rotation angle.
IN	Ppoint3 *	scale	scale vector.
OUT	Pint *	err	error indicator.
OUT	Pmatrix3	mat	4x4 matrix.

This function performs the BUILD TRANSFORMATION MATRIX3 function using the `Ppoint3` datatype instead of `Pvec3`.

ptk_pcomposetran

void **ptk_pcomposetran** (Pmatrix mat, Ppoint *pt, Ppoint *shift, Pfloat angle, Ppoint *scale, Pint *err, Pmatrix result)

IN	Pmatrix	mat	3x3 matrix.
IN	Ppoint *	pt	fixed point.
IN	Ppoint *	shift	shift vector.
IN	Pfloat	angle	rotation angle.
IN	Ppoint *	scale	scale vector.
OUT	Pint *	err	error indicator.
OUT	Pmatrix	result	3x3 matrix.

This function performs the COMPOSE TRANSFORMATION MATRIX function using the `Ppoint` datatype instead of `Pvec`.

ptk_pcomposetran3

void **ptk_pcomposetran3** (Pmatrix3 mat, Ppoint3 *pt, Ppoint3 *shift, Pfloat xangle, Pfloat yangle, Pfloat zangle, Ppoint3 *scale, Pint *err, Pmatrix3 result)

OUT	Pmatrix3	mat	4x4 matrix.
IN	Ppoint3 *	pt	fixed point.
IN	Ppoint3 *	shift	shift vector.
IN	Pfloat	xangle	x rotation angle.
IN	Pfloat	yangle	y rotation angle.
IN	Pfloat	zangle	z rotation angle.
IN	Ppoint3 *	scale	scale vector.
OUT	Pint *	err	error indicator.
OUT	Pmatrix3	result	4x4 matrix.

This function performs the COMPOSE TRANSFORMATION MATRIX3 function using the `Ppoint3` datatype instead of `Pvec3`.

ptk_pevalvieworientationmatrix

void **ptk_pevalvieworientationmatrix** (Ppoint *vrp, Ppoint *vup, Pint *err, Pmatrix mat)

IN	Ppoint *	vrp	view reference point.
IN	Ppoint *	vup	view up vector.
OUT	Pint *	err	error indicator.
OUT	Pmatrix	mat	3x3 matrix.

This function performs the EVALUATE VIEW ORIENTATION MATRIX function using the `Ppoint3` datatype instead of `Pvec3`.

ptk_pevalvieworientationmatrix3

void **ptk_pevalvieworientationmatrix3** (Ppoint3 *vrp, Ppoint3 *vpn, Ppoint3 *vup, Pint *err, Pmatrix3 mat)

IN	Ppoint3 *	vrp	view reference point.
IN	Ppoint3 *	vup	view up vector.
IN	Ppoint3 *	vpn	view plane normal.
OUT	Pint *	err	error indicator.
OUT	Pmatrix	mat	3x3 matrix.

This function performs the EVALUATE VIEW ORIENTATION MATRIX3 function using the `Ppoint3` datatype instead of `Pvec3`.

ptk_pscale

void **ptk_pscale** (Ppoint *scalept, Pint *err, Pmatrix mat)

IN	Ppoint *	scalept	scale vector.
OUT	Pint *	err	error indicator.
OUT	Pmatrix	mat	3x3 matrix.

This function performs the SCALE function using a `Ppoint` datatype instead of `Pvec`.

ptk_pscale3

void **ptk_pscale3** (Ppoint3 *scalept, Pint *err, Pmatrix3 mat)

IN	Ppoint3 *	scalept	scale vector.
OUT	Pint *	err	error indicator.
OUT	Pmatrix3	mat	4x4 matrix.

This function performs the SCALE3 function using a `Ppoint3` datatype instead of `Pvec3`.

ptk_psetannotationcharup

void **ptk_psetannotationcharup** (Ppoint *pt)

IN	Ppoint *	pt	annotation character up vector.

This function performs the SET ANNOTATION CHARACTER UP VECTOR function using a `Ppoint` datatype instead of `Pvec`.

ptk_psetcharup

void **ptk_psetcharup** (Ppoint *pt)

IN	Ppoint *	pt	character up vector.

This function performs the SET CHARACTER UP VECTOR function using a `Ppoint` datatype instead of `Pvec`.

ptk_ptranslate

void **ptk_ptranslate** (Ppoint *tranpt, Pint *err, Pmatrix mat)

IN	Ppoint *	tranpt	translation vector.
OUT	Pint *	err	error indicator.
OUT	Pmatrix	mat	3x3 matrix.

This function performs the TRANSLATE function using a `Ppoint` datatype instead of `Pvec`.

ptk_ptranslate3

void **ptk_ptranslate3** (Ppoint3 *tranpt, Pint *err, Pmatrix3 mat)

IN	Ppoint3 *	tranpt	translation vector.
OUT	Pint *	err	error indicator.
OUT	Pmatrix3	mat	4x4 matrix.

This function performs the TRANSLATE3 function using a `Ppoint3` datatype instead of `Pvec3`.

Chapter 7

Phinter functions

ptk_callphinter

void **ptk_callphinter** (void)

This function provides an interface to phinter and its related functions. The available commands are as follows:

help output this list.

interactive call phinter with stdin for input script.

stringinput call phinter using string device for input and terminal windows for output.

read call phinter with a given input script.

outputfile set output script filename.

informfile set information filename.

outputterm set output terminal window identifier.

informterm set information terminal window identifier.

writestruct write contents of structures to output file.

writestruct write contents of structure networks to output file.

writestruct write contents of all structures to output file.

quit or exit leave callphinter.

ptk_elemcontent

ptkboolean **ptk_elemcontent** (Pint stid, Pint elemid, Pint termid, Pint *error)

IN	Pint	stid	structure identifier.
IN	Pint	elemid	element number.
IN	Pint	termid	terminal window identifier.
OUT	PInt *	error	error code.

This function writes the contents of an element in a terminal window. If the element is an output primitive then it is inserted into the currently open structure at the current editing position and the function returns TRUE, otherwise FALSE.

ptk_phinter

void **ptk_phinter** (FILE *input, FILE *output, FILE *inform)

IN	FILE *	input	file pointer for input script.
OUT	FILE *	output	file pointer for output script.
OUT	FILE *	inform	file pointer for information such as results of inquiry calls..

This function reads a PHIGS script from a file or from standard input. If `stdin` is passed as the input file pointer then phinter becomes interactive and prompts are given for function parameters. The other file pointers are used for writing an output script and for writing data which results from inquiry calls and so on.

ptk_readelem

void **ptk_readelem** (Pint ws, Plimit *echoarea, Pelem_type eltype)

IN	Pint	wsid	workstation identifier.
IN	Plimit *	echo area	echo area for string device.
IN	Pelem_type	eltype	element type to read in.

This function reads the contents of a PHIGS element from the PHIGS string device (number 1). Prompts are given for the required data depending on the element type. An element with the input data is inserted into the currently open structure at the current editing position.

ptk_readphinterscript

ptkboolean **ptk_readphinterscript** (char *scriptname, FILE *output, FILE *inform)

IN	char *	scriptname	script filename.
OUT	FILE *	output	output script file pointer.
OUT	FILE *	inform	information script file pointer.

This function reads a PHIGS script from the file specified by `scriptname`. The file is automatically opened and closed and the function returns TRUE if a PHIGS script has been successfully read.

ptk_strphinter

void **ptk_strphinter** (Pint wsid, Plimit *echoarea, Pint outputterminal, Pint informterminal)

IN	Pint	wsid	workstation identifier.
IN	Plimit *	echo area	echo area for string device.
IN	Pint	outputterminal	terminal window identifier for writing output script to.
IN	Pint	informterminal	terminal window identifier for writing information data.

This function redirects the input of phinter to the PHIGS string device (number 1). Strphinter is always interactive and output is directed to terminal windows instead of files so that it may be displayed in the PHIGS workstation window.

ptk_writeallstruct

void **ptk_writeallstruct** (FILE *fileptr)

OUT	FILE *	fileptr	pointer to file.

This function writes the contents of all the structures in the PHIGS CSS to a file. The structures are written in the PHIGS script format so that they may be read in again using `ptk_phinter`.

ptk_writestruct

void **ptk_writestruct** (FILE *fileptr, Pint_list *stids)

OUT	FILE *	fileptr	pointer to file.
IN	Pint_list *	stids	structure identifier list.

This function writes the contents of a list of structures to a file. The structures are written in the PHIGS script format so that they may be read in again using `ptk_phinter`.

ptk_writestructnet

void **ptk_writestructnet** (FILE *fileptr, Pint_list *stids)

OUT	FILE *	fileptr	pointer to file.
IN	Pint_list *	stids	structure network identifier list.

This function writes the contents of a list of structure networks to a file. The structures are written in the PHIGS script format so that they may be read in again using `ptk_phinter`.

Chapter 8

Structure content functions

ptk_inqstructcontentelemptr

void **ptk_inqstructcontentelemptr** (Pint contentstid, Pint *elemptr, Pint *err)

IN	Pint	contentstid	content structure identifier.
OUT	Pint *	elemptr	element pointer.
OUT	Pint *	err	error indicator.

This function may be used to obtain the element number pointed to by the element arrow in the structure content diagram. The error code = 1 if `contentstid` is not a structure content diagram.

ptk_inqstructcontentrange

void **ptk_inqstructcontentrange** (Pint contentstid, Pint *firstel, Pint *lastel, Pint *err)

IN	Pint	contentstid	content structure identifier.
OUT	Pint *	firstel	element pointer.
OUT	Pint *	lastel	element pointer.
OUT	Pint *	err	error indicator.

This function may be used to obtain the element range which appears in the structure content diagram. The error code = 1 if `contentstid` is not a structure content diagram.

ptk_setstructcontentelemptr

void **ptk_setstructcontentelemptr** (Pint contentstid, Pint elemptr)

IN	Pint	contentstid	content structure identifier.
IN	Pint	elemptr	element pointer.

This function draws an arrow pointing the element `elemptr` in the structure content diagram.

ptk_structcontent

void **ptk_structcontent** (Pint wsid, Pint stid, Pint firstel, Pint lastel, Pint elemptr,
Pint font, Pint *error)

IN	Pint	wsid	workstation identifier.
IN	Pint	stid	structure identifier.
IN	Pint	firstel	first element in range.
IN	Pint	lastel	last element in range.
IN	Pint	elemptr	element pointer.
IN	Pint	font	text font.
OUT	Pint *	error	error code.

This function inserts a diagram of the contents of the structure `stid` in the currently open structure at the current editing position. The diagram consists of a table of elements represented by rectangles and labelled with element type and number. The table has a heading with the format: `"structure N "name" (M elements)"` where N is the structure identifier, name is the structure name extracted from the "structureid" hashtable and M is the total number of elements in the structure. The error code = 1 if there is no open structure and = 2 if `stid` doesn't exist. This function requires hashtable "label".

Chapter 9

Topology Functions

9.1 Data types

```
typedef enum
   {
    PTKESINGLE,
    PTKEGROUP
   } ptkenodetype;

typedef enum
   {
    PTKEBOXTOPOLOGY,
    PTKESTRUCTTOPOLOGY,
    PTKESTRUCTNETTOPOLOGY
   } ptketopologytype;

typedef struct ptkstopologydata
   {
    ptketopologytype topologytype;
    Pint topologyname;
    Pint topologystid;
   } ptkstopologydata;
```

9.2 Functions

ptk_createtopology

void **ptk_createtopology** (Pint topid, Pint root, Pint *err)

IN	Pint	topid	topology identifier.
IN	Pint	root	structure network identifier.
OUT	Pint *	error	error code.

This function creates a diagram of the structure network specified by `root`. The diagram is a PHIGS structure which uses boxes connected by lines to represent structures and EXECUTE STRUCTURE elements. The error code = 1 if the root structure does not exist and = 2 if `topid` already exists. This function requires hashtables "structureid", "label", "name".

ptk_createtopologypaths

void **ptk_createtopologypaths** (Pint topid, Pelem_ref_list_list *paths, Pint *error)

IN	Pint	topid	topology identifier.
IN	Pelem_ref_list_list *	paths	structure-element paths.
OUT	Pint *	error	error code.

This function creates a topology diagram using the structure path list `paths`. This enables a topology diagram to be built of a non-existent structure network or of a subset of an actual structure network. The error code = 1 if `topid` already exists. This function requires hashtables "structureid", "label", "name".

ptk_deltopology

ptkboolean **ptk_deltopology** (Pint topid)

IN	Pint	topid	topology identifier.

This function deletes a topology diagram from the PHIGS Toolkit topology store. The function returns TRUE if a topology is deleted, otherwise FALSE.

ptk_inqnodeposition

void **ptk_inqnodeposition** (Pint topid, Pint structid, Ppoint *nodept, Pint *err)

IN	Pint	topid	topology identifier.
IN	Pint	structid	structure identifier.
OUT	Ppoint *	nodept	node position.
OUT	Pint *	err	error indicator.

This function may be used to obtain the position of a topology node in a topology diagram. The position is returned in the range [0, 1]. The node is specified using the structure identifier of the structure that it represents. The error code = 1 if `topid` doesn't exist and = 2 if `structid` is not a node in the topology.

ptk_inqpostedtopologies

void **ptk_inqpostedtopologies** (Pint wsid, Pint size, Pint_list *topids, Pint *totalsize, Pint *err)

IN	Pint	wsid	workstation identifier.
IN	Pint	size	size of buffer.
OUT	Pint_list *	topids	list of posted topologies.
OUT	Pint *	totalsize	length of posted topologies list.
OUT	Pint *	err	error indicator.

This function may be used to obtain a list of all the topology diagrams which are posted to the workstation `wsid`.

ptk_inqtopologyattrs

void **ptk_inqtopologyattrs** (Pint topid, Pint *txfont, Pint *linecol, Pint *textcol, Pint *edgecol, Pint *intcol, Pint *htedgecol, Pint *htintcol, Pint *err)

IN	Pint	topid	topology identifier.
OUT	Pint *	txfont	label text font.
OUT	Pint *	linecol	polyline colour index.
OUT	Pint *	textcol	text colour index.
OUT	Pint *	edgecol	edge colour index.
OUT	Pint *	intcol	interior colour index.
OUT	Pint *	htedgecol	highlight edge colour index.
OUT	Pint *	htintcol	highlight interior colour index.
OUT	Pint *	err	error indicator.

This function may be used to obtain the text font and colour attribute values of a topology diagram. The error code = 1 if `topid` doesn't exist.

ptk_inqtopologyhighlightnode

void **ptk_inqtopologyhighlightnode** (Pint topid, Pint *topnodestid, Pint *err)

IN	Pint	topid	topology identifier.
OUT	Pint *	topnodestid	topology node structure identifier.
OUT	Pint *	err	error indicator.

This function may be used to obtain the structure identifier of the currently highlighted topology node. The error code = 1 if `topid` doesn't exist and = 2 if there is no highlighted node.

ptk_inqtopologyids

void **ptk_inqtopologyids** (Pint size, Pint_list *topids, Pint *totalsize, Pint *err)

IN	Pint	size	size of buffer.
OUT	Pint_list *	topids	list of topology identifiers.
OUT	Pint *	totalsize	length of topology identifiers list.
OUT	Pint *	err	error indicator.

This function may be used to obtain a list of all the topology diagrams in the PHIGS Toolkit topology store.

ptk_inqtopologyname

void **ptk_inqtopologyname** (Pint topid, Pint *name, Pint *err)

IN	Pint	topid	topology identifier.
OUT	Pint *	name	topology name for nameset filters.
OUT	Pint *	err	error indicator.

This function may be used to obtain the topology name for use in the pick filter. When a topology name is added to the pick filter only the topology nodes are pickable. The error code = 1 if `topid` doesn't exist.

ptk_inqtopologystructid

void **ptk_inqtopologystructid** (Pint topid, Pint *topstid, Pint *err)

IN	Pint	topid	topology identifier.
IN	Pint *	topstid	topology structure identifier.
OUT	Pint *	err	error indicator.

This function may be used to obtain the structure identifier of the topology diagram `topid`. In the case of the BOX topology type the diagram is a single PHIGS structure but for STRUCT and STRUCTNET type diagrams it is a structure network. The error code = 1 if `topid` doesn't exist.

ptk_inqtopologytype

void **ptk_inqtopologytype** (Pint topid, ptketopologytype *toptype, Pint *err)

IN	Pint	topid	topology identifier.
OUT	ptketopologytype *	toptype	topology type.
OUT	Pint *	err	error indicator.

This function may be used to obtain the type of a topology diagram. The possible types are BOX, STRUCT and STRUCTNET, with BOX as the default. The error code = 1 if `topid` doesn't exist.

ptk_posttopology

void **ptk_posttopology** (Pint wsid, Pint topid, Pfloat priority)

IN	Pint	wsid	workstation identifier.
IN	Pint	topid	topology identifier.
IN	Pfloat	priority	display priority.

This function posts a topology diagram structure to the workstation `wsid`.

ptk_restoretopologylayout

void **ptk_restoretopologylayout** (FILE *fileptr, Pint topid)

OUT	FILE *	fileptr	pointer to file.
IN	Pint	topid	topology identifier.

This function reads a topology layout from a text file. The layout is used to reposition the nodes of the topology diagram `topid` but will only really make sense if the layout was stored originally from the same topology.

ptk_setnodeposition

void **ptk_setnodeposition** (Pint topid, Pint structid, Ppoint *nodept, ptkenodetype nodetype)

IN	Pint	topid	topology identifier.
IN	Pint	structid	structure identifier.
IN	Ppoint *	nodept	node position.
IN	ptkenodetype	nodetype	type of node.

This function sets the position of a topology node or group of nodes. The position is given in the range [0, 1]. The node is specified using the structure identifier of the structure that it represents. If nodetype is set to GROUP then all descendent nodes of `structid` are moved relative to it.

ptk_settopologyattrs

void **ptk_settopologyattrs** (Pint topid, Pint txfont, Pint linecol, Pint textcol, Pint edgecol, Pint intcol, Pint htedgecol, Pint htintcol)

IN	Pint	topid	topology identifier.
IN	Pint	txfont	label text font.
IN	Pint	linecol	polyline colour index.
IN	Pint	textcol	text colour index.
IN	Pint	edgecol	edge colour index.
IN	Pint	intcol	interior colour index.
IN	Pint	htedgecol	highlight edge colour index.
IN	Pint	htintcol	highlight interior colour index.

This function sets the text font and colour attribute values of a topology diagram. The text font applies to the structure names which are extracted from the "structureid" hashtable. The highlight colour indicies are used to highlight a single topology node in the function `ptk_settopologyhighlightnode`.

ptk_settopologyhighlightnode

void **ptk_settopologyhighlightnode** (Pint topid, Pint topnodestid)

IN	Pint	topid	topology identifier.
IN	Pint	topnodestid	topology node structure identifier.

This function highlights a single topology node by setting the colour attribute values of the edge and interior of the node box. This function only works for BOX topology types.

ptk_settopologytype

void **ptk_settopologytype** (Pint topid, ptketopologytype toptype)

IN	Pint	topid	topology identifier.
IN	ptketopologytype	toptype	topology type.

This function sets the type of a topology diagram to BOX, STRUCT or STRUCTNET. The BOX topology type is the default and the STRUCT and STRUCTNET types insert parts of the actual structures into the nodes. As a result these topology types do not work well for networks containing SET VIEW INDEX and SET GLOBAL TRANSFORMATION elements.

ptk_storetopologylayout

void **ptk_storetopologylayout** (FILE *fileptr, Pint topid)

OUT	FILE *	fileptr	pointer to file.
IN	Pint	topid	topology identifier.

This function saves a topology layout in a text file. The layout refers to the positions of all the topology nodes. The format of the stored layout is: ** `start (for each node in topology diagram) x y` end

ptk_tidytopology

void **ptk_tidytopology** (Pint wsid, Pint topid, ptkenodetype nodetype, Pint pickdev, Pint pickpet, Ppick_data *pickdatarec, Pint locdev, Pint locpet, Ploc_data *locdatarec)

IN	Pint	wsid	workstation identifier.
IN	Pint	topid	topology identifier.
IN	ptkenodetype	nodetype	type of node.
IN	Pint	pickdev	pick device.
IN	Pint	pickpet	pick prompt/echo type.
IN	Ppick_data *	pickdatarec	pick data record.
IN	Pint	locdev	locator device.
IN	Pint	locpet	locator prompt/echo type.
IN	Ploc_data *	locdatarec	locator data record.

This function enables the user to set the position of a topology node, or group of nodes, interactively. The pick device `pickdev` is used to request a topology node and if successful the locator device `locdev` is used to specify a new node position. Prompt and echo types may be set for both the pick and locator devices.

ptk_unposttopology

void **ptk_unposttopology** (Pint wsid, Pint topid)

IN	Pint	wsid	workstation identifier.
IN	Pint	topid	topology identifier.

This function unposts the topology diagram structure from the workstation `wsid`.

Chapter 10

Menu functions

10.1 Data types

```
#define PTKCMAXSTRINGSIZE 255

typedef enum
   {
    PTKEONED,
    PTKETWOD,
    PTKETHREED
   } ptkerotatortypes;

typedef struct
   {
    Piclass inputclass;
    union
      {
        struct
           {
            Pint locview;
    Ppoint3 locpoint;
           } ptkslocinput;
        struct
           {
            Ppickpath pickdata;
            Ppathorder pathorder;
           } ptkspickinput;
         Pchar str[256];
       } ptkugeninput;
   } ptksmenuinput;

typedef struct
   {
     Pint menuid;
     Pint itemnum;
     boolean measure;
     Ppoint3 value;
   } ptksmenuoutput;
```

```
typedef enum
   {
    PTKEUSERMENU,
    PTKEBOXMENU,
    PTKEROTATOR
   } ptkemenutype;
```

10.2 Functions

ptk_backmenu

void **ptk_backmenu** (Pint wsid, Pint menuid)

IN	Pint	wsid	workstation identifier.
IN	Pint	menuid	menu identifier.

This function sets the post priority of the menu structure so that it is displayed below all other posted menus but in front of all posted windows.

ptk_clearboxmenuhighlight

void **ptk_clearboxmenuhighlight** (Pint menuid)

IN	Pint	menuid	menu identifier.

This function returns the attributes of the highlighted menu item to their original values. If no item is highlighted the function is ignored.

ptk_createboxmenu

void **ptk_createboxmenu** (Pint menuid, Ppoint *tlcorner, Ppoint *boxsize)

IN	Pint	menuid	menu identifier.
IN	Ppoint *	tlcorner	top left corner of menu.
IN	Ppoint *	boxsize	width and height of menu box item.

This function creates a box menu with no initial items. The position of the menu is specified by `tlcorner` which defines the top-left corner of the first menu item. The position and size of box menu items are given in the range [0, 1]. The default menu path is DOWN. This function requires hashtables "structureid", "label", "name".

ptk_createrotator

void **ptk_createrotator** (Pint wsid, Pint menuid, ptkerotatortype rottype, Ppoint *size, char *titlestr, Pfloat titleheight)

IN	Pint	wsid	workstation identifier.
IN	Pint	menuid	rotator identifier.
IN	ptkerotatortype	rottype	rotator type.
IN	Ppoint *	size	rotator size.
IN	char *	titlestr	rotator title.
IN	Pfloat	titleheight	rotator title height.

This function creates a special form of user menu called a rotator. Rotators consist of an arrangement of arrows and are useful for defining rotation values and direction in a user interface. There are three types of rotator available: 1D, 2D and 3D, and each having an increasing number of arrows. This function requires hashtables "structureid", "label", "name".

ptk_createstructmenuitem

void **ptk_createstructmenuitem** (Pint menuid, Pint structure, Pint itemno, Pedit_mode editmode, Pint *error)

IN	Pint	menuid	menu identifier.
IN	Pint	structure	menu item structure identifier.
IN	Pint	itemno	menu item number.
IN	Pedit_mode	editmode	insert or replace menu item.
IN	Pint *	error	error code.

This function creates a menu item defined by `structure`. In the case of box menus the structure is mapped into the item box with aspect ratio preserved. No transformation is applied for user menu items. The error code = 1 if `menuid` doesn't exist.

ptk_createtextmenuitem

void **ptk_createtextmenuitem** (Pint menuid, char *str, Pint itemno, Pedit_mode editmode, Pint *error)

IN	Pint	menuid	menu identifier.
IN	char *	str	text string.
IN	Pint	itemno	menu item number.
IN	Pedit_mode	editmode	insert or replace menu item..
IN	Pint *	error	error code.

This function creates a box menu item containing the character string `str`. The string is automatically scaled to fit inside the menu item box. This function may only be used with box menus. The error code = 1 if `menuid` doesn't exist and = 2 if the menu is not a box menu.

ptk_createusermenu

void **ptk_createusermenu** (Pint menuid, Pint menustid)

IN	Pint	menuid	menu identifier.
IN	Pint	menustid	menu structure identifier.

This function creates a user menu using the structure `menustid`. Each menu item must be an individual PHIGS structure or network and the menu initially contains no menu items. This function requires hashtables "label", "name".

ptk_delmenu

ptkboolean **ptk_delmenu** (Pint menuid)

IN	Pint	menuid	menu identifier.

This function deletes a menu from the PHIGS Toolkit menu store. The function returns TRUE if `menuid` is deleted, otherwise FALSE.

ptk_delmenuitem

ptkboolean **ptk_delmenuitem** (Pint menuid, Pint itemno)

IN	Pint	menuid	menu identifier.
IN	Pint	itemno	menu item to delete.

This function deletes the menu item `itemno`. The function returns TRUE if the menu item is deleted, otherwise FALSE.

ptk_frontmenu

void **ptk_frontmenu** (Pint wsid, Pint menuid)

IN	Pint	wsid	workstation identifier.
IN	Pint	menuid	menu identifier.

This function sets the post priority of the menu structure so that it is displayed on top of all the other posted menus and windows.

ptk_inqboxmenuattrs

void **ptk_inqboxmenuattrs** (Pint menuid, Ptext_path *menupath, Pint *font, Pint *textcolour, Pint *intcolour, Pint *edgecolour, Pint *boxtlcolour, Pint *boxbrcolour, Pint *httextcolour, Pint *htintcolour, Pint *htedgecolour, Pint *err)

IN	Pint	menuid	menu identifier.
OUT	Ptext_path *	menupath	path of box menu (left, right, up, down).
OUT	Pint *	font	text font.
OUT	Pint *	textcolour	colour index for text.
OUT	Pint *	intcolour	colour index for interior.
OUT	Pint *	edgecolour	colour index for edges.
OUT	Pint *	boxtlcolour	colour index for top-left of box.
OUT	Pint *	boxcolour	colour index for bottom-right of box.
OUT	Pint *	httextcolour	colour index for highlight text.
OUT	Pint *	htintcolour	colour index for highlight interior.
OUT	Pint *	htedgecolour	colour index for highlight edges.
OUT	Pint *	err	error indicator.

This function may be used to obtain the boxmenu attribute values for `menuid`. These include the menu path, text font and colour indicies. The error code = 1 if `menuid` doesn't exist and = 2 if it is not a box menu.

ptk_inqboxmenuhighlightitem

void **ptk_inqboxmenuhighlightitem** (Pint menuid, Pint *item, Pint *err)

IN	Pint	menuid	menu identifier.
OUT	Pint *	item	hightlight item number.
OUT	Pint *	err	error indicator.

This function may be used to obtain the item number of the currently highlight box menu item. The error code = 1 if `menuid` doesn't exist and = 2 if there is no highlighted item.

ptk_inqfrontbackmenuid

ptkboolean **ptk_inqfrontbackmenuid** (Pint wsid, Pint *frontid, Pint *backid, Pint *err)

IN	Pint	wsid	workstation identifier.
OUT	Pint *	frontstid	front menu identifier.
OUT	Pint *	backstid	back menu identifier.
OUT	Pint *	err	error indicator.

This function may be used to obtain the menu identifiers of the front and back menus which are posted to workstation `wsid`. These are the menus with the highest and lowest post priority.

**

ptk_inqmenuids

void **ptk_inqmenuids** (Pint size, Pint_list *menuids, Pint *totalsize, Pint *err)

IN	Pint	size	size of buffer.
OUT	Pint_list *	menuids	list of menus.
OUT	Pint *	totalsize	length of menus list.
OUT	Pint *	err	error indicator.

This function may be used to obtain a list of all menus in the PHIGS Toolkit menu store.

ptk_inqmenuname

void **ptk_inqmenuname** (Pint menuid, Pint *name, Pint *err)

IN	Pint	windid	menu identifier.
OUT	Pint *	name	menu name.
OUT	Pint *	err	error indicator.

This function may be used to obtain the menu name for use in the pick filter. The error code = 1 if `menuid` doesn't exist.

ptk_inqmenuposition

void **ptk_inqmenuposition** (Pint menuid, Ppoint *position, Pint *err)

IN	Pint	menuid	menu identifier.
OUT	Ppoint *	position	menu position.
OUT	Pint *	err	error indicator.

This function may be used to obtain the position of the top-left corner of first menu item. The position is returned in the range [0, 1]. The error code = 1 if `menuid` doesn't exist.

ptk_inqmenustructid

void **ptk_inqmenustructid** (Pint menuid, Pint *menustid, Pint *err)

IN	Pint	menuid	menu identifier.
OUT	Pint *	menustid	menu structure identifier.
OUT	Pint *	err	error indicator.

This function may be used to obtain the identifier of a menu structure. The error code = 1 if `menuid` doesn't exist.

ptk_inqpostedmenus

void **ptk_inqpostedmenus** (Pint wsid, Pint size, Pint_list *menuids, Pint *totalsize, Pint *err)

IN	Pint	wsid	workstation identifier.
IN	Pint	size	size of buffer.
OUT	Pint_list *	menuids	list of posted menus.
OUT	Pint *	totalsize	length of posted menus list.
OUT	Pint *	err	error indicator.

This function may be used to inquire the list of all menus which are posted to workstation `wsid`. The error code is a standard PHIGS error code.

ptk_inqrotatorattrs

void **ptk_inqrotatorattrs** (Pint menuid, Pint *titlefont, Pint *titlecolour, Pint *arrowcolour, Pint *arrowedgecolour, Pint *intcolour, Pint *edgecolour, Pint *bannercolour, Pint *boxtlcolour, Pint *boxbrcolour, Pint *err)

IN	Pint	menuid	rotator identifier.
OUT	Pint *	titlefont	title font.
OUT	Pint *	titlecolour	title colour index.
OUT	Pint *	arrowcolour	arrow interior colour index.
OUT	Pint *	arrowedgecolour	arrow edge colour index.
OUT	Pint *	intcolour	rotator interior colour index.
OUT	Pint *	edgecolour	edge colour index.
OUT	Pint *	bannercolour	banner colour index.
OUT	Pint *	boxtlcolour	top-left colour index.
OUT	Pint *	boxbrcolour	bottom-right colour index.
OUT	Pint *	err	error indicator.

This function may be used to obtain the attribute values of a rotator menu. The error code = 1 if `menuid` doesn't exist and = 2 if the menu is not a rotator.

ptk_inqrotatortitle

void **ptk_inqrotatortitle** (Pint menuid, Pint len, char *titlestr, Pint *totlen, Pint *err)

IN	Pint	menuid	menu identifier.
IN	Pint	len	length of string.
IN	char *	titlestr	title string of rotator banner.
OUT	Pint *	totlen	actual length of string.
OUT	Pint *	err	error indicator.

This function may be used to obtain the title string of a rotator menu. The error code = 1 if `menuid` doesn't exist and = 2 if the menu is not a rotator.

ptk_locscanmenus

ptkboolean **ptk_locscanmenus** (Pint wsid, Ppoint *point, Pint *menuid, Pint *itemnum, Ppoint *value)

IN	Pint	wsid	workstation identifier.
IN	Ppoint *	point	input point.
OUT	Pint *	menuid	menu identifier.
OUT	Pint *	itemnum	item number.
OUT	Ppoint *	value	position of point within item.

This function uses the INCREMENTAL SPATIAL SEARCH function of PHIGS to test if `point` lies within a posted menu. The menus are tested begining the highest priority menu and working through to the back menu. The position of `point` relative to bottom-left corner of the menu item bounding box is returned in `value`. The function returns TRUE if `point` lies within a menu, otherwise FALSE.

ptk_pickscanmenus

ptkboolean **ptk_pickscanmenus** (Ppick_path *pickpath, Ppath_order pathorder, Pint *menuid, Pint *itemnum)

IN	Ppick_path *	pickpath	path through structure network..
IN	Ppath_order	pathorder	order of data in pickpath.
OUT	Pint *	menuid	menu identifier.
OUT	Pint *	itemnum	item number.

This function tests the pick path to inquire if a menu item was picked. The function returns TRUE if a menu item was picked, otherwise FALSE.

ptk_postmenu

void **ptk_postmenu** (Pint wsid, Pint menuid)

IN	Pint	wsid	workstation identifier.
IN	Pint	menuid	menu identifier.

This function posts the menu structure to the workstation `wsid`. The priority of the menu structure is controlled by the PHIGS Toolkit menu system to provide an ordered stacking of displayed menus. When `menuid` is posted it becomes the front menu.

ptk_scanmenus

ptkboolean **ptk_scanmenus** (Pint wsid, ptksgeneralinput *input, ptksmenuoutput *output)

IN	Pint	wsid	workstation identifier.
IN	ptksgeneralinput *	input	input data.
IN	ptksmenuoutput *	output	output data.

This function performs a scan on all posted menus using either string, pick or locator data. The function returns TRUE if a menu item has been selected, otherwise FALSE.

ptk_setboxmenuattrs

void **ptk_setboxmenuattrs** (Pint wsid, Pint menuid, Ptext_path menupath, Pint font, Pint textcolour, Pint intcolour, Pint edgecolour, Pint boxtlcolour, Pint boxbrcolour, Pint httextcolour, Pint htintcolour, Pint htedgecolour)

IN	Pint	wsid	workstation identifier.
IN	Pint	menuid	menu identifier.
IN	Ptext_path	menupath	path of box menu (left, right, up, down).
IN	Pint	font	text font.
IN	Pint	textcolour	colour index for text.
IN	Pint	intcolour	colour index for interior.
IN	Pint	edgecolour	colour index for edges.
IN	Pint	boxtlcolour	colour index for top-left of box.
IN	Pint	boxcolour	colour index for bottom-right of box.
IN	Pint	httextcolour	colour index for highlight text.
IN	Pint	htintcolour	colour index for highlight interior.
IN	Pint	htedgecolour	colour index for highlight edges.

This function sets the box menu path, text font and colour attribute values. The highlight colour indicies are used by the function `ptk_setboxmenuhighlightitem` to highlight a single menu item.

ptk_setboxmenuhighlightitem

void **ptk_setboxmenuhighlightitem** (Pint menuid, Pint itemnum)

IN	Pint	menuid	menu identifier.
IN	Pint	itemnum	menu item number.

This function highlights a menu item by setting the colour index values for the text, interior and edge of a box menu item.

ptk_setboxmenutextfont

void **ptk_setboxmenutextfont** (Pint wsid, Pint menuid, Pint font)

IN	Pint	wsid	workstation identifier.
IN	Pint	menuid	menu identifier.
IN	Pint	font	text font.

This function sets the text font of all text menu items in the menu structure. The menu must be a box menu.

ptk_setmenuposition

void **ptk_setmenuposition** (Pint menuid, Ppoint *menupos)

IN	Pint	menuid	menu identifier.
IN	Ppoint *	menupos	menu position.

This function sets the position of the top-left corner of the first menu item. The position is given in the range [0, 1]. If the position results in part of the menu being clipped then the actual position is adjusted so that as much as possible of the menu is visible.

ptk_setrotatorattrs

void **ptk_setrotatorattrs** (Pint wsid, Pint menuid, Pint titlefont, Pint titlecolour, Pint arrowcolour, Pint arrowedgecolour, Pint intcolour, Pint edgecolour, Pint bannercolour, Pint boxtlcolour, Pint boxbrcolour)

IN	Pint	wsid	workstation identifier.
IN	Pint	menuid	rotator identifier.
IN	Pint	titlefont	title font.
IN	Pint	titlecolour	title colour index.
IN	Pint	arrowcolour	arrow interior colour index.
IN	Pint	arrowedgecolour	arrow edge colour index.
IN	Pint	intcolour	rotator interior colour index.
IN	Pint	edgecolour	edge colour index.
IN	Pint	bannercolour	banner colour index.
IN	Pint	boxtlcolour	top-left colour index.
IN	Pint	boxbrcolour	bottom-right colour index.

This function sets the attribute values of a rotator menu. The arrows are drawn within a box similar to that of box menu items and the box has an area for a title string. All arrows are drawn with the same colour values defined by `arrowcolour` for the interior and `arrowedgecolour` for the outline.

ptk_setrotatortitle

void **ptk_setrotatortitle** (Pint menuid, char *titlestr)

IN	Pint	menuid	menu identifier.
IN	char *	titlestr	title string of rotator banner.

This function sets the title string of the rotator menu to be `titlestr`. The string is automatically scaled to fit in the rotator title box.

ptk_stringscanmenus

ptkboolean **ptk_stringscanmenus** (Pint wsid, char *str, Pint *menuid, Pint *itemnum)

IN	Pint	wsid	workstation identifier.
IN	char *	str	string.
OUT	Pint *	menuid	menu identifier.
OUT	Pint *	itemno	item number.

This function compares the character string `str` with the items of all posted menus. The comparison begins with the highest priority menu and works through to the back menu. User menu items are also searched for text primitives with which to compare the string. The comparison is case sensitive so that "item 1" is not the same as "IteM 1". The function returns TRUE if the string matches a menu item, otherwise FALSE.

ptk_unpostallmenu

void **ptk_unpostallmenu** (Pint wsid)

IN	Pint	wsid	workstation identifier.
IN	Pint	menuid	menu identifier.

This function unposts all menus from the workstation `wsid`.

ptk_unpostmenu

void **ptk_unpostmenu** (Pint wsid, Pint menuid)

IN	Pint	wsid	workstation identifier.
IN	Pint	menuid	menu identifier.

This function unposts a menu structure from the workstation `wsid`. The front and back menus are updated if necessary.

Chapter 11

Window functions

11.1 Data types

```
typedef enum
   {
    PTKESTRUCTWINDOW,
    PTKETERMINALWINDOW,
    PTKETOPOLOGYWINDOW,
    PTKECONTENTWINDOW
   } ptkewindowtype;

typedef enum
   {
    PTKEWINDOWBANNER,
    PTKEWINDOWFRAME,
    PTKEWINDOWVIEW,
    PTKEWINDOWICON
   } ptkewindowarea;

typedef enum
   {
    PTKEWINDOWOPEN,
    PTKEWINDOWCLOSED,
   } ptkewindowstate;

typedef enum
   {
    PTKECAMERAON,
    PTKECAMERAOFF,
   } ptkecamerastate;

typedef struct
   {
    Pint windowid;
    ptkewindowarea windowarea;
    boolean measure;
    Ppoint value;
   } ptkswindowoutput;
```

11.2 Functions

ptk_backwindow

void **ptk_backwindow** (Pint windid)

IN	Pint	windid	window identifier.

This function sets the post priority of a window structure so that it is displayed behind all the other posted windows and menus.

ptk_clearterminal

void **ptk_clearterminal** (Pint windid)

IN	Pint	windid	terminal window identifier.

This function empties the structure containing all the text written to the TERMINAL window.

ptk_closewindow

void **ptk_closewindow** (Pint windid)

IN	Pint	windid	window identifier.

This function posts the icon structure and unposts the window structure from the window's workstation.

ptk_createwindow

void **ptk_createwindow** (Pint wsid, Pint windid, Ppoint *size, Ppoint *position,
char *titlestring)

IN	Pint	wsid	workstation identifier.
IN	Pint	windid	window identifier.
IN	Ppoint *	size	window size.
IN	Ppoint *	position	window position.
IN	char *	titlestring	title string.

This function creates a window structure which may be used for viewing PHIGS structures, PHIGS Toolkit topology diagrams and PHIGS Toolkit structure content diagrams. A terminal window type which contains only text. The window size and position are given in the range [0, 1]. Each window has a virtual camera which is useful for moving around a scene. This function requires hashtables "structureid", "label", "name", "viewindex".

ptk_delwindow

ptkboolean **ptk_delwindow** (Pint windid)

IN	Pint	windid	window identifier.

This function deletes a window from the PHIGS Toolkit window store.

ptk_frontwindow

void **ptk_frontwindow** (Pint windid)

IN	Pint	windid	window identifier.

This function sets the post priority of the window structure so that it is displayed on top of all other posted windows but has a lower prioity than the current back menu.

ptk_inqbannerheight

void **ptk_inqbannerheight** (Pint windid, Pfloat *bannerheight, Pint *err)

IN	Pint	windid	window identifier.
OUT	Pfloat *	bannerheight	height of banner.
OUT	Pint *	err	error indicator.

This function may be used to obtain the height of a window banner. It is returned in the range [0, 1]. The error code = 1 if `windid` doesn't exist.

ptk_inqbannertitle

void **ptk_inqbannertitle** (Pint windid, Pint len, char *titlestr, Pint *totlen, Pint *err)

IN	Pint	windid	window identifier.
IN	Pint	len	length of string.
IN	char *	titlestr	title string of banner.
OUT	Pint *	totlen	actual length of string.
OUT	Pint *	err	error indicator.

This function may be used to obtain the title of a window. The error code = 1 if `windid` doesn't exist.

ptk_inqcameralimits

void **ptk_inqcameralimits** (Pint windid, Plimit3 *limits, Pint *err)

IN	Pint	windid	window identifier.
OUT	Plimit3 *	limits	camera world limits.
OUT	Pint *	err	error indicator.

This function may be used to obtain the camera view volume. This defines the volume of a scene that the camera knows about and is returned in World Coordinates. The error code = 1 if `windid` doesn't exist.

ptk_inqcameraposition

void **ptk_inqcameraposition** (Pint windid, Ppoint3 *position, Pint *err)

IN	Pint	windid	window identifier.
OUT	Ppoint3 *	position	camera position.
OUT	Pint *	err	error indicator.

This function may be used to obtain the camera position in World Coordinates. The error code = 1 if `windid` doesn't exist.

ptk_inqcameraprojtype

void **ptk_inqcameraprojtype** (Pint windid, Pproj_type *projtype, Pint *err)

IN	Pint	windid	window identifier.
OUT	Pproj_type *	projtype	camera projection type.
OUT	Pint *	err	error indicator.

This function may be used to obtain the camera view projection type which is either PARALLEL or PERSPECTIVE. The error code = 1 if `windid` doesn't exist.

ptk_inqcameraptinterest

void **ptk_inqcameraptinterest** (Pint windid, Ppoint3 *ptinterest, Pint *err)

IN	Pint	windid	window identifier.
OUT	Ppoint3 *	ptinterest	camera point of interest.
OUT	Pint *	err	error indicator.

This function may be used to obtain the camera point of interest in World Coordinates. The error code = 1 if `windid` doesn't exist.

ptk_inqcamerastate

void **ptk_inqcamerastate** (Pint windid, ptkecamerastate *cameraswitch, Pint *err)

IN	Pint	windid	window identifier.
OUT	ptkecamerastate *	cameraswitch	camera ON/OFF switch.
OUT	Pint *	err	error indicator.

This function may be used to obtain the camera state which is either ON or OFF. The error code = 1 if `windid` doesn't exist.

ptk_inqcontentviewrange

void **ptk_inqcontentviewrange** (Pint windid, Pint *range1, Pint *range2, Pint *err)

IN	Pint	windid	terminal window identifier.
IN	Pint	range1	element number.
IN	Pint	range2	element number.
OUT	Pint *	err	error indicator.

This function may be used to obtain the viewing range of a structure content diagram which is posted to a CONTENT window. The error code = 1 if `windid` doesn't exist and = 2 if it is not a content window.

ptk_inqframesize

void **ptk_inqframesize** (Pint windid, Ppoint *size, Pint *err)

IN	Pint	windid	window identifier.
OUT	Ppoint *	size	frame size.
OUT	Pint *	err	error indicator.

This function may be used to obtain the dimensions of the window frame. They are returned in the range [0, 1] and the default dimensions are (0.01, 0.01). The error code = 1 if `windid` doesn't exist.

ptk_inqfrontbackwindowid

ptkboolean **ptk_inqfrontbackwindowid** (Pint wsid, Pint *frontid, ptkewindowstate *frontstate, Pint *backid, ptkewindowstate *backstate, Pint *err)

IN	Pint	wsid	workstation identifier.
OUT	Pint *	frontstid	front window identifier.
OUT	ptkewindowstate *	frontstate	front window state.
OUT	Pint *	backstid	back window identifier.
OUT	ptkewindowstate *	backstate	back window state.
OUT	Pint *	err	error indicator.

This function may be used to obtain the identifiers of the front and back windows and there current state (OPEN or CLOSED).

ptk_inqiconposition

void **ptk_inqiconposition** (Pint windid, Ppoint *position, Pint *err)

IN	Pint	windid	window identifier.
OUT	Ppoint *	position	icon position.
OUT	Pint *	err	error indicator.

This function may be used to obtain the position of the centre of a window icon which is returned in the range [0, 1]. The error code = 1 if `windid` doesn't exist.

ptk_inqiconsize

void **ptk_inqiconsize** (Pint windid, Ppoint *size, Pint *err)

IN	Pint	windid	window identifier.
OUT	Ppoint *	size	icon size.
OUT	Pint *	err	error indicator.

This function may be used to obtain the size of a window's icon structure and is returned in the range [0, 1]. The error code = 1 if `windid` doesn't exist.

ptk_inqpositionaxis

void **ptk_inqpositionaxis** (Pint windid, Ppoint3 *axis, Pint *err)

IN	Pint	windid	window identifier.
OUT	Ppoint3 *	axis	camera position axis of rotation.
OUT	Pint *	err	error indicator.

This function may be used to obtain the axis of rotation which is used to rotate the camera position in the function `ptk_rotatecameraposition`. The error code = 1 if `windid` doesn't exist.

ptk_inqpostedwindows

void **ptk_inqpostedwindows** (Pint wsid, Pint size, Pint_list *windowids, Pint *totalsize, Pint *err)

IN	Pint	wsid	workstation identifier.
IN	Pint	size	size of buffer.
OUT	Pint_list *	windowids	list of posted windows.
OUT	Pint *	totalsize	length of posted windows list.
OUT	Pint *	err	error indicator.

This function may be used to obtain a list of all windows posted to the workstation `wsid`. The error code is a standard PHIGS error code.

ptk_inqptinterestaxis

void **ptk_inqptinterestaxis** (Pint windid, Ppoint3 *axis, Pint *err)

IN	Pint	windid	window identifier.
OUT	Ppoint3 *	axis	camera point of interest axis of rotation.
OUT	Pint *	err	error indicator.

This function may be used to obtain the axis of rotation which is used to rotate the camera point of interest in the function `ptk_rotatecameraptinterest`. The error code = 1 if `windid` doesn't exist.

ptk_inqterminaldata

void **ptk_inqterminaldata** (Pint windid, Pint *numlines, Pint *numcolms, Pint *txfont, Pint *txcolour, Pint *err)

IN	Pint	windid	terminal window identifier.
OUT	Pint *	numlines	number of lines in window.
OUT	Pint *	numcolmns	number of columns in window.
OUT	Pint *	txfont	text font.
OUT	Pint *	txcolour	text colour.
OUT	Pint *	err	error indicator.

This function may be used to obtain the number of lines displayed in a TERMINAL window and the number of characters in a line. Also the text font and colour used are returned in `txfont` and `txcolour`. The error code = 1 if `windid` doesn't exist and = 2 if it is not a terminal window.

ptk_inqterminalfloatformat

void **ptk_inqterminalfloatformat** (Pint windid, Pint size, Pint *totalsize, char *rformat, Pint *err)

IN	Pint	windid	terminal window identifier.
IN	Pint	size	size of buffer, in bytes, as allocated by application.
IN	Pint *	totalsize	actual size of buffer.
IN	char *	rformat	string giving C-type float number output style.
OUT	Pint *	err	error indicator.

This function may be used to obtain the floating-point used for writing floating-point numbers to a TERMINAL window. The error code = 1 if `windid` doesn't exist and = 2 if it is not a terminal window.

ptk_inqterminalstructid

void **ptk_inqterminalstructid** (Pint windid, Pint *termtextstid, Pint *err)

IN	Pint	windid	terminal window identifier.
IN	Pint *	rformat	string giving C-type real number output style.
OUT	Pint *	err	error indicator.

This function may be used to obtain the identifier of the structure used to display the text written to a TERMINAL window. The error code = 1 if `windid` doesn't exist and = 2 if it is not a terminal window.

ptk_inqtopologyviewarea

void **ptk_inqtopologyviewarea** (Pint windid, Plimit *viewarea, Pint *err)

IN	Pint	windid	window identifier.
OUT	Plimit *	viewarea	area of topology to view.
OUT	Pint *	err	error indicator.

This function may be used to obtain the viewing area of a topology diagram posted to a TOPOLOGY window. The error code = 1 if `windid` doesn't exist and = 2 if it is not a topology window.

ptk_inqusericon

void **ptk_inqusericon** (Pint windid, Pint *iconstid, Pint *err)

IN	Pint	windid	window identifier.
OUT	Pint *	iconstid	user icon structure identifier.
OUT	Pint *	err	error indicator.

This function may be used to obtain the identifier of a user created icon structure. The structure is referenced by the window's icon structure. The error code = 1 if `windid` doesn't exist and = 2 if the default icon is used.

ptk_inqwindowattrs

void **ptk_inqwindowattrs** (Pint windid, Pint *titlefont, Pint *titlecolour, Pint *bannercolour, Pint *backgdcolour, Pint *edgecolour, Pint *frametlcolour, Pint *framebrcolour, Pint *err)

IN	Pint	windid	window identifier.
OUT	Pint *	titlefont	title string font.
OUT	Pint *	titlecolour	title string colour index.
OUT	Pint *	bannercolour	banner colour index.
OUT	Pint *	backgdcolour	background colour index of window.
OUT	Pint *	edgecolour	edge colour index of window.
OUT	Pint *	frametlcolour	top-left frame colour index.
OUT	Pint *	framebrcolour	bottom-right frame colour index.
OUT	Pint *	err	error indicator.

This function may be used to obtain the text font and colour attribute values of a window. The error code = 1 if `windid` doesn't exist.

ptk_inqwindowids

void **ptk_inqwindowids** (Pint size, Pint_list *windowids, Pint *totalsize, Pint *err)

IN	Pint	size	size of buffer.
OUT	Pint_list *	windowids	list of windows.
OUT	Pint *	totalsize	length of windows list.
OUT	Pint *	err	error indicator.

This function may be used to obtain a list of all windows in the PHIGS Toolkit window store.

ptk_inqwindowname

void **ptk_inqwindowname** (Pint windid, Pint *name, Pint *err)

IN	Pint	windid	window identifier.
OUT	Pint *	name	window name.
OUT	Pint *	err	error indicator.

This function may be used to obtain the window name to be used in namesets for the pick, invisibility and highlighting filters. The error code = 1 if `windid` doesn't exist.

ptk_inqwindowposition

void **ptk_inqwindowposition** (Pint windid, Ppoint *position, Pint *err)

IN	Pint	windid	window identifier.
OUT	Ppoint *	position	window position.
OUT	Pint *	err	error indicator.

This function may be used to obtain the position of the centre of a window. The position is returned in the range [0, 1]. The error code = 1 if `windid` doesn't exist.

ptk_inqwindowsize

void **ptk_inqwindowsize** (Pint windid, Ppoint *size, Pint *err)

IN	Pint	windid	window identifier.
OUT	Ppoint *	size	window size.
OUT	Pint *	err	error indicator.

This function may be used to obtain the size of a window which is returned in the range [0, 1]. The error code = 1 if `windid` doesn't exist.

ptk_inqwindowstate

void **ptk_inqwindowstate** (Pint windid, ptkewindowstate *state, Pint *err)

IN	Pint	windid	window identifier.
OUT	ptkewindowstate *	state	window state, open or closed.
OUT	Pint *	err	error indicator.

This function may be used to obtain the window state, open or closed. The error code = 1 if `windid` doesn't exist.

ptk_inqwindowstructid

void **ptk_inqwindowstructid** (Pint windid, Pint *windowstid, Pint *iconstid, Pint *err)

IN	Pint	windid	window identifier.
OUT	Pint *	windowstid	window structure identifier.
OUT	Pint *	iconstid	window structure identifier.
OUT	Pint *	err	error indicator.

This function may be used to obtain the identifier of a window structure and its corresponding icon structure. The window structure is a network with references to all the items posted to the window. The default icon is a single structure containing the window identifier. The error code = 1 if `windid` doesn't exist.

ptk_inqwindowtype

void **ptk_inqwindowtype** (Pint windid, ptkewindowtype *type, Pint *err)

IN	Pint	windid	window identifier.
OUT	ptkewindowtype *	type	window type.
OUT	Pint *	err	error indicator.

This function may be used to inquire the type of a window. The available types are STRUCT, TOPOLOGY, CONTENT and TERMINAL. The default window type is STRUCT and may be used to view any PHIGS structures. The error code = 1 if `windid` doesn't exist.

ptk_inqwindowviewrep

void **ptk_inqwindowviewrep** (Pint windid, Pview_rep3 *rep, Pint *err)

IN	Pint	windid	window identifier.
OUT	Pview_rep3 *	rep	window view representation.
OUT	Pint *	err	error indicator.

This function may be used to obtain the viewing parameters which are used to set the window view when the camera is switched OFF. The error code = 1 if `windid` doesn't exist.

ptk_locscanwindows

ptkboolean **ptk_locscanwindows** (Pint wsid, Ppoint *point, Pint *windowid, ptkewindowarea *windowarea, Ppoint *value)

IN	Pint	wsid	workstation identifier.
IN	Ppoint *	point	input point.
OUT	Pint *	windowid	window identifier.
OUT	ptkewindowarea *	windowarea	window area.
OUT	Ppoint *	value	position of point within window area.

This function uses the INCREMENTAL SPATIAL SEARCH function of PHIGS to test if `point` lies within a window posted to workstation `wsid`. The window area (one of BANNER, VIEW, FRAME or ICON) and the position of `point` relative to the bottom-left corner of the bounding box of the window area are returned in `windowarea` and `value`. The function returns TRUE if `point` lies within a window, otherwise FALSE.

ptk_openwindow

void **ptk_openwindow** (Pint windid)

IN	Pint	windid	window identifier.

This function posts the window structure and unposts the icon structure from the window's workstation.

ptk_pickscanwindows

ptkboolean **ptk_pickscanwindows** (Ppick_path *pickpath, Ppath_order pathorder, Pint *windowid, ptkewindowarea *windowarea)

IN	Ppick_path *	pickpath	path through structure network.
IN	Ppath_order	pathorder	order of path through network.
OUT	Pint *	windowid	window identifier.

This function tests the pick path to inquire if a window structure was picked. The window area picked may be one of BANNER, VIEW, FRAME or ICON. The function returns TRUE if a window was picked, otherwise FALSE.

ptk_posttowindow

void **ptk_posttowindow** (Pint windid, Pint id)

IN	Pint	windid	window identifier.
IN	Pint	id	item identifier.

This function posts an item to a window depending on the window type. In the case of STRUCT and CONTENT windows, `id` is a structure identifier. For TOPOLOGY windows, `id` is a topology identifier. If the window is a TERMINAL window this function is ignored.

ptk_postwindow

void **ptk_postwindow** (Pint windid)

IN	Pint	windid	window identifier.

This function posts a window structure to the workstation specified when the window was initially created. Windows are bound to workstation because they each use one view table entry to define the window view. The priority of the window structure is controlled by the PHIGS Toolkit window system to provide an ordered stacking mechanism for windows.

ptk_printfterminal

void **ptk_printfterminal** (Pint windid, char *format, va_alist)

IN	Pint	windid	terminal window identifier.
IN	char *	format	format string.
IN		va_alist	variable length argument list.

This function writes to a TERMINAL window using a variable length argument list and a format string like the C printf function.

ptk_refreshterminal

void **ptk_refreshterminal** (Pint windid)

IN	Pint	windid	terminal window identifier.

This function refreshes the TERMINAL window so that the last text line is visible.

ptk_resetcamera

void **ptk_resetcamera** (Pint windid)

IN	Pint	windid	window identifier.

This function resets the camera variables to their default values. The camera view volume is left unchanged.

ptk_rotatecameraposition

void **ptk_rotatecameraposition** (Pint windid, Pfloat angle)

IN	Pint	windid	window identifier.
IN	Pfloat	angle	rotation angle in degrees.

This function rotates the camera position in a circle centred at the point of interest and about the axis defined by the function `ptk_setpositionaxis`. The amount of rotation is `angle` degrees and the function is useful for rotating around an object.

ptk_rotatecameraptinterest

void **ptk_rotatecameraptinterest** (Pint windid, Pfloat angle)

IN	Pint	windid	window identifier.
IN	Pfloat	angle	rotation angle in degrees.

This function rotates the camera point of interest in a circle centred at the camera position and about the axis defined by the function `ptk_setptinterestaxis`. The amount of rotation is `angle` degrees and the function is useful for panning around a scene.

ptk_rotatecameraupvector

void **ptk_rotatecameraupvector** (Pint windid, Pfloat angle)

IN	Pint	windid	window identifier.
IN	Pfloat	angle	rotation angle in degrees.

This function rotates the camera up vector about the axis joining the camera position to the point of interest by `angle` degrees.

ptk_rotatepositionxaxis

void **ptk_rotatepositionxaxis** (Pint windid, Pfloat angle)

IN	Pint	windid	window identifier.
IN	Pfloat	angle	rotation angle in degrees.

This function rotates the camera position in a circle centred at the point of interest and about the x axis of a right-handed coordinate system whose z axis is defined by the camera position and point of interest.

ptk_rotatepositionyaxis

void **ptk_rotatepositionyaxis** (Pint windid, Pfloat angle)

IN	Pint	windid	window identifier.
IN	Pfloat	angle	rotation angle in degrees.

This function rotates the camera position in a circle centred at the point of interest and about the y axis of a right-handed coordinate system whose z axis is defined by the camera position and point of interest.

ptk_rotateptinterestxaxis

void **ptk_rotateptinterestxaxis** (Pint windid, Pfloat angle)

IN	Pint	windid	window identifier.
IN	Pfloat	angle	rotation angle in degrees.

This function rotates the camera point of interest in a circle centred at the camera position and about the x axis of a right-handed coordinate system whose z axis is defined by the camera position and point of interest.

ptk_rotateptinterestyaxis

void **ptk_rotateptinterestyaxis** (Pint windid, Pfloat angle)

IN	Pint	windid	window identifier.
IN	Pfloat	angle	rotation angle in degrees.

This function rotates the camera point of interest in a circle centred at the camera position and about the y axis of a right-handed coordinate system whose z axis is defined by the camera position and point of interest.

ptk_scaleviewwindow

void **ptk_scaleviewwindow** (Pint windid, Pfloat scalefactor)

IN	Pint	windid	window identifier.
IN	Pfloat	scalefactor	scale factor.

This function scales the view window uniformly by `scalefactor`. A scalefactor in the range [0, 1] will scale down the view window and create a zoom-in effect. A scalefactor greater than 1.0 will give a zoom-out effect.

ptk_scanwindows

ptkboolean **ptk_scanwindows** (Pint wsid, ptksgeneralinput *input, ptkswindowoutput *output)

IN	Pint	wsid	workstation identifier.
IN	ptksgeneralinput *	input	input data.
IN	ptkswindowoutput *	output	output data.

This function performs a scan on all posted windows using either string, pick or locator data. The function returns TRUE if a window has been selected, otherwise FALSE.

ptk_scrollterminal

void **ptk_scrollterminal** (Pint windid, Ptext_path scrolldir, Pint numlines)

IN	windid	terminal window identifier.
IN	scrolldir	scroll direction (up or down).
IN	numlines	number of lines to scroll by.

This function scrolls the contents of the TERMINAL window either UP or DOWN by `numlines`.

ptk_setbannercolours

void **ptk_setbannercolours** (Pint windid, Pint bannercolour, Pint titlecolour)

IN	Pint	windid	window identifier.
IN	Pint	bannercolour	banner colour index.
IN	Pint	titlecolour	title string colour index.

This function sets the colour indicies of a window banner. It is useful for highlighting a current window, for example in a 'point and click' window system.

ptk_setbannerheight

void **ptk_setbannerheight** (Pint windid, Pfloat bannerheight)

IN	Pint	windid	window identifier.
IN	Pfloat	bannerheight	height of banner.

This function sets the height of the window banner to `bannerheight` which is given in the range [0, 1]. The window title is re-scaled to fit the new height.

ptk_setbannertitle

void **ptk_setbannertitle** (Pint windid, char *titlestring)

IN	Pint	windid	window identifier.
IN	char *	titlestring	title string of window banner.

This function sets the title string of a window. The title is displayed in the window banner and is automatically scaled to fit inside the banner area.

ptk_setcameralimits

void **ptk_setcameralimits** (Pint windid, Plimit3 *limits, ptkboolean adjust)

IN	Pint	windid	window identifier.
IN	Plimit3 *	limits	camera viewing limits in WC.
IN	ptkboolean	adjust	adjust given limits.

This function initialises the view volume of the camera to the specified bounding box given in World Coordinates. If `adjust` is set to TRUE then the bounding box will be adjusted to be the bounding box of a sphere which encloses the original box. The camera position and point of interest are set accordingly.

ptk_setcameraposition

void **ptk_setcameraposition** (Pint windid, Ppoint3 *position)

IN	Pint	windid	window identifier.
IN	Ppoint3 *	position	camera position.

This function sets the camera position to `position` which is given in World Coordinates.

ptk_setcameraprojtype

void **ptk_setcameraprojtype** (Pint windid, Pproj_type proj)

IN	Pint	windid	window identifier.
IN	Pproj_type	proj	camera projection type.

This function sets the projection type of the view given by the camera to PARALLEL or PERSPECTIVE. The default is PARALLEL.

ptk_setcameraptinterest

void **ptk_setcameraptinterest** (Pint windid, Ppoint3 *ptinterest)

IN	Pint	windid	window identifier.
IN	Ppoint3 *	ptinterest	point of interest.

This function sets the camera point of interest to `ptinterest` which is given in World Corrdinates.

ptk_setcamerastate

void **ptk_setcamerastate** (Pint windid, ptkecamerastate cameraswitch)

IN	Pint	windid	window identifier.
IN	ptkecamerastate	cameraswitch	camera on or off.

This function sets the camera of the window `windid` to ON or OFF. The default is ON.

ptk_setcameraworld

void **ptk_setcameraworld** (Pint windid, Pint_list *stids)

IN	Pint	windid	window identifier.
IN	Pint_list *	stids	structure identifier list.

This function initialises the view volume of the camera so that it contains the combined bounding box of all the structures and structure networks in `stids`. The camera position and point of interest are set accordingly.

ptk_setcontentviewrange

void **ptk_setcontentviewrange** (Pint windid, Pint range1, Pint range2)

IN	Pint	windid	window identifier.
IN	Pint	range1	start element number.
IN	Pint	range2	end element number.

This function sets the range of elements of a structure content diagram to view in a CONTENT window.

ptk_setframesize

void **ptk_setframesize** (Pint windid, Ppoint *size)

IN	Pint	windid	window identifier.
IN	Ppoint *	size	frame size.

This function sets the thickness of the window frame. The x and y dimensions are given in the range [0, 1].

ptk_seticonposition

void **ptk_seticonposition** (Pint windid, Ppoint *position)

IN	Pint	windid	window identifier.
IN	Ppoint *	position	icon position.

This function sets the position of the centre of the window's icon structure. The position is given in the range [0, 1].

ptk_seticonsize

void **ptk_seticonsize** (Pint windid, Ppoint *size)

IN	Pint	windid	window identifier.
IN	Ppoint *	size	icon size.

This function sets the size of the window's icon structure. The size is given in the range [0, 1].

ptk_setpositionaxis

void **ptk_setpositionaxis** (Pint windid, Ppoint3 *axis)

IN	Pint	windid	window identifier.
IN	Ppoint3 *	axis	axis of rotation.

This function sets the axis of rotation for rotating the camera position using the function `ptk_rotatecameraposition`.

ptk_setptinterestaxis

void **ptk_setptinterestaxis** (Pint windid, Ppoint3 *axis)

IN	Pint	windid	window identifier.
IN	Ppoint3 *	axis	axis of rotation.

This function sets the axis of rotation for rotating the camera point of interest using the function `ptk_rotatecameraptinterest`.

ptk_setterminaldata

void **ptk_setterminaldata** (Pint windid, Pint numlines, Pint txfont, Pint txcolour)

IN	Pint	windid	terminal window identifier.
IN	Pint	numlines	number of lines in window.
IN	Pint	numcolmns	number of columns in window.
IN	Pint	txfont	text font.
IN	Pint	txcolour	text colour.

This function sets the number of lines to be displayed in a TERMINAL window and which text font and colour to use.

ptk_setterminalfloatformat

void **ptk_setterminalfloatformat** (Pint windid, char *rformat)

IN	Pint	windid	terminal window identifier.
IN	char *	rformat	string giving C-type float number output style.

This function sets the format for writing floating-point numbers to a TERMINAL window. The format used is the same syntax as in the C language.

ptk_settopologyviewarea

void **ptk_settopologyviewarea** (Pint windid, Plimit *viewarea)

IN	Pint	windid	window identifier.
IN	Plimit *	viewarea	area of topology to view.

This function sets the viewing area of a topology diagram posted to a TOPOLOGY window. The area is defined in the range [0, 1].

ptk_setusericon

void **ptk_setusericon** (Pint windid, Pint usericon)

IN	Pint	windid	window identifier.
IN	Pint	user icon	icon structure identifier.

This function enables the application to specify a structure identifier to use as a window icon. The structure is executed from the window's default icon structure and the icon size and position functions still apply provided the user icon is defined within the World Coordinate range [0, 1].

ptk_setviewclipping3

void **ptk_setviewclipping3** (Pint windid, Plimit3 *cliplims, Pclip_ind clipxy, Pclip_ind clipback, Pclip_ind clipfront)

IN	Pint	windid	window identifier.
IN	Plimit3 *	cliplims	view clipping limits.
IN	Pclip_ind	clipxy	x-y clipping indicator.
IN	Pclip_ind	clipback	back plane clipping indicator.
IN	Pclip_ind	clipfront	front plane clipping indicator.

This function sets the window view clipping values. The camera must be switched OFF for these values to be set.

ptk_setviewmapping3

void **ptk_setviewmapping3** (Pint windid, Plimit *window, Plimit3 *viewport, Pproj_type proj, Ppoint3 *prp, Pfloat viewplane, Pfloat backplane, Pfloat frontplane, Pint *error)

IN	Pint	windid	window identifier.
IN	Plimit *	window	view window.
IN	Plimit3 *	viewport	projection viewport.
IN	Pproj_type	proj	projection type.
IN	Ppoint3 *	prp	projection reference point.
IN	Pfloat	viewplane	view plane distance.
IN	Pfloat	backplane	back plane distance.
IN	Pfloat	frontplane	front plane distance.
OUT	Pint *	error	error indicator.

This function sets the window view mapping values. The camera must be switched OFF for these values to be set. The largest square within the window is defined to be the device coordinates area which the view maps onto. The error code is a standard PHIGS error code.

ptk_setvieworientation3

void **ptk_setvieworientation3** (Pint windid, Ppoint3 *vrp, Ppoint3 *vpn, Ppoint3 *vup, Pint *error)

IN	Pint	windid	window identifier.
IN	Ppoint3 *	vrp	view reference point.
IN	Ppoint3 *	vpn	view plane normal.
IN	Ppoint3 *	vup	view up vector.
OUT	Pint *	error	error indicator.

This function sets the window view orientation values. The camera must be switched OFF for these values to be set. The error code is a standard PHIGS error code.

ptk_setwindowattrs

void **ptk_setwindowattrs** (Pint windid, Pint titlefont, Pint titlecolour, Pint bannercolour, Pint backgdcolour, Pint edgecolour, Pint frametlcolour, Pint framebrcolour)

IN	Pint	windid	window identifier.
IN	Pint	titlefont	title string font.
IN	Pint	titlecolour	title string colour index.
IN	Pint	bannercolour	banner colour index.
IN	Pint	backgdcolour	background colour index of window.
IN	Pint	edgecolour	edge colour index of window.
IN	Pint	frametlcolour	top-left frame colour index.
IN	Pint	framebrcolour	bottom-right frame colour index.

This function sets the window text font and colour attribute values. Each window has a banner region which contains the title string of the window. The text font value applies to this string.

ptk_setwindowposition

void **ptk_setwindowposition** (Pint windid, Ppoint *position)

IN	Pint	windid	window identifier.
IN	Ppoint *	position	window position.

This function sets the position of the centre of the window. The position is given in the range [0, 1]. If the position results in part of the window being clipped then the position is adjusted so that the whole window is visible.

ptk_setwindowsize

void **ptk_setwindowsize** (Pint windid, Ppoint *size)

IN	Pint	windid	window identifier.
IN	Ppoint *	size	window size.

This function sets the size of the window using the x value as the width and the y value as the height. The values are given in the range [0, 1]. If the size results in part of the window being clipped then the window size is adjusted to give as large a window as possible.

ptk_setwindowtraninputpri

void **ptk_setwindowtraninputpri** (Pint windid, Pint refwindid, Prel_pri priority)

IN	Pint	windid	window identifier.
IN	Pint	refwindid	reference window identifier.
IN	Prel_pri	priority	relative priority.

This function sets the transformation input priority of the window's view representation relative to another window. The relative priority is also set relative to view index 0.

ptk_setwindowtype

void **ptk_setwindowtype** (Pint windid, ptkewindowtype windtype)

IN	Pint	windid	window identifier.
IN	ptkewindowtype	windtype	window type.

This function sets the type of a window to one of STRUCT, TOPOLOGY, CONTENT and TERMINAL. The default window type is STRUCT. All items are unposted from the window before the type is set.

ptk_shiftcamera

void **ptk_shiftcamera** (Pint windid, Ppoint3 *shift)

IN	Pint	windid	window identifier.
IN	Ppoint3 *	shift	shift factor.

This function shifts the camera point of interest and camera position about the axes of a right-handed coordinate system whose z axis is defined by the camera position and point of interest.

ptk_stringscanwindows

ptkboolean **ptk_stringscanwindows** (Pint wsid, char *str, Pint *windowid)

IN	Pint	wsid	workstation identifier.
IN	char *	str	string.
OUT	Pint *	windowid	window identifier.

This function compares the character string `str` with title string of all the windows posted to workstation `wsid`. The string comparison is case sensitive and begins with the front window and works back to the lowest priority window. The function returns TRUE if a match is found, otherwise FALSE.

ptk_unpostallfromwindow

void **ptk_unpostallfromwindow** (Pint windid)

IN	Pint	windid	window identifier.

This function unposts all items posted to window `windid`. If the window is a TERMINAL window this function is ignored.

ptk_unpostfromwindow

void **ptk_unpostfromwindow** (Pint windid, Pint id)

IN	Pint	windid	window identifier.
IN	Pint	id	item identifier.

This function unposts an item from a window depending on the window type. In the case of STRUCT and CONTENT windows, id is a structure identifier. For TOPOLOGY windows, id is a topology identifier. If the window is a TERMINAL window this function is ignored.

ptk_unpostwindow

void **ptk_unpostwindow** (Pint windid)

IN	Pint	windid	window identifier.

This function unposts a window from the workstation it is bound to.

ptk_writefloatterminal

void **ptk_writefloatterminal** (Pint windid, Pfloat number)

IN	Pint	windid	terminal window identifier.
IN	Pfloat	number	float number to write to window.

This function writes a floating point number to a TERMINAL window.

ptk_writeintterminal

void **ptk_writeintterminal** (Pint windid, Pint number)

IN	Pint	windid	terminal window identifier.
IN	Pint	number	integer to write to window.

This function writes an integer to a TERMINAL window.

ptk_writelnterminal

void **ptk_writelnterminal** (Pint windid, char *str)

IN	Pint	windid	terminal window identifier.
IN	char *	str	string to write to window.

This function writes a character string to the current line of the TERMINAL window. A new line is started when the end of the current line is reached and at the next call to a TERMINAL write function.

ptk_writeterminal

void **ptk_writeterminal** (Pint windid, char *str)

IN	Pint	windid	terminal window identifier.
IN	char *	str	string to write to window.

This function writes a character string to the current line of the TERMINAL window. A new line is started when the end of the current line is reached.

Chapter 12

Debugger functions

ptk_debugger

void **ptk_debugger** (Pint wsid, Pint stid)

IN	Pint	wsid	workstation identifier.
IN	Pint	stid	structure identifier.

This function starts the PHIGS debugger on workstation `wsid`, and makes structure `stid` the root of the structure network to be debugged. This function requires hashtables "structureid", "label", "name", "viewindex", "topologyid", "menuid", "windowid".

ptk_inqdebuggerattrs

void **ptk_inqdebuggerattrs** (Pint *menufont, Pint *windowfont, Pint *menucol, Pint *menutextcol, Pint *windowcol, Pint *bannercol, Pint *bannertextcol, Pint *tlcol, Pint *brcol, Pint *arrowcol, Pint *arrowedgecol)

IN	Pint *	menufont	menu text font.
IN	Pint *	windowfont	window text font.
IN	Pint *	menucol	menu colour index.
IN	Pint *	menutextcol	menu text colour index.
IN	Pint *	windowcol	window interior colour index.
IN	Pint *	bannercol	window banner colour index.
IN	Pint *	bannertextcol	window banner text colour index.
IN	Pint *	tlcol	top-left colour index.
IN	Pint *	brcol	bottom-right colour index.
IN	Pint *	arrowcol	arrow colour index.
IN	Pint *	arrowedgecol	arrow edge colour index.

This function may be used to obtain the text font and colour attribute values of menus and windows used in the PHIGS debugger.

ptk_setdebuggerattrs

void **ptk_setdebuggerattrs** (Pint menufont, Pint windowfont, Pint menucol, Pint menutextcol, Pint windowcol, Pint bannercol, Pint bannertextcol, Pint tlcol, Pint brcol, Pint arrowcol, Pint arrowedgecol)

IN	Pint	menufont	menu text font.
IN	Pint	windowfont	window text font.
IN	Pint	menucol	menu colour index.
IN	Pint	menutextcol	menu text colour index.
IN	Pint	windowcol	window interior colour index.
IN	Pint	bannercol	window banner colour index.
IN	Pint	bannertextcol	window banner text colour index.
IN	Pint	tlcol	top-left colour index.
IN	Pint	brcol	bottom-right colour index.
IN	Pint	arrowcol	arrow colour index.
IN	Pint	arrowedgecol	arrow edge colour index.

This function enables an application to set the fonts and colours of menus and windows in the PHIGS debugger.

Chapter 13

Viewing functions

ptk_inqvieweditorattrs

void **ptk_inqvieweditorattrs** (Pint *menufont, Pint *windowfont, Pint *menucol, Pint *menutextcol, Pint *windowcol, Pint *bannercol, Pint *bannertextcol, Pint *tlcol, Pint *brcol, Pint *arrowcol, Pint *arrowedgecol)

IN	Pint *	menufont	menu text font.
IN	Pint *	windowfont	window text font.
IN	Pint *	menucol	menu colour index.
IN	Pint *	menutextcol	menu text colour index.
IN	Pint *	windowcol	window interior colour index.
IN	Pint *	bannercol	window banner colour index.
IN	Pint *	bannertextcol	window banner text colour index.
IN	Pint *	tlcol	top-left colour index.
IN	Pint *	brcol	bottom-right colour index.
IN	Pint *	arrowcol	arrow colour index.
IN	Pint *	arrowedgecol	arrow edge colour index.

This function may be used to obtain the text font and colour attribute values of menus and windows used in the PHIGS view editor.

ptk_setvieweditorattrs

void **ptk_setvieweditorattrs** (Pint menufont, Pint windowfont, Pint menucol, Pint menutextcol, Pint windowcol, Pint bannercol, Pint bannertextcol, Pint tlcol, Pint brcol, Pint arrowcol, Pint arrowedgecol)

IN	Pint	menufont	menu text font.
IN	Pint	windowfont	window text font.
IN	Pint	menucol	menu colour index.
IN	Pint	menutextcol	menu text colour index.
IN	Pint	windowcol	window interior colour index.
IN	Pint	bannercol	window banner colour index.
IN	Pint	bannertextcol	window banner text colour index.
IN	Pint	tlcol	top-left colour index.
IN	Pint	brcol	bottom-right colour index.
IN	Pint	arrowcol	arrow colour index.
IN	Pint	arrowedgecol	arrow edge colour index.

This function enables the application to set the text font and colour attribute values for the menus and windows used in the PHIGS view editor.

ptk_vieweditor

void **ptk_vieweditor** (Pint wsid, Pint_list *stids, Pview_rep3 *rep)

IN	Pint	wsid	workstation identifier.
IN	Pint_list *	stids	list of structure identifiers.
OUT	Pview_rep3 *	rep	view representation.

This function starts the interactive PHIGS view editor, on workstation `ws`. This function requires hashtables "structureid", "label", "name", "viewindex", "topologyid", "menuid", "windowid".

C Function Index