ESV Workstation

Reference Manual

EVANS & SUTHERLAND COMPUTER CORPORATION
Salt Lake City, Utah

DOCUMENTATION WARRANTY:

PURPOSE: This documentation is provided to assist an Evans & Sutherland trained BUYER in using a product purchased from Evans & Sutherland. It may contain errors or omissions that only a trained individual may recognize. Changes may have occurred to the hardware/software, to which this documentation refers, which are not included in this documentation or may be on a separate errata sheet. Use of this documentation in such changed hardware/software could result in damage to hardware/software. User assumes full responsibility of all such results of the use of this data.

WARRANTY: This document is provided, and Buyer accepts such documentation, "AS-IS" and with "ALL FAULTS, ERRORS, AND OMISSIONS." BUYER HEREBY WAIVES ALL IMPLIED AND OTHER WARRANTIES, GUARANTIES, CONDITIONS OR LIABILITIES, EXPRESSED OR IMPLIED ARISING BY LAW OR OTHERWISE, INCLUDING, WITHOUT LIMITATIONS, ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. BUYER FURTHER HOLDS SELLER HARMLESS OF ANY DIRECT OR INDIRECT DAMAGES, INCLUDING CONSEQUENTIAL DAMAGES.

Part Number: 517940-102 AA

April, 1991

1. ES/PEX

Table of Contents

1. ES/PEX

Introduction

This chapter contains the following sections:

- "Introduction" (this section) describes the ES/PEX releases and graphics standards.
- "What is PEX?" describes the X Model, the PHIGS Model, and the PEX Model.
- "Functional Overview" contains a general discussion of the ES/PEX function types supported by the ESV Workstation.
- "PHIGS Functions" contains a list of all of the PHIGS and PHIGS PLUS functions and identifies those that are not currently supported.
- "Compatibility with Prior Releases" describes the porting process for running applications written prior to the 2.0 Release.
- "**OPEN XPHIGS**" describes the **OPEN XPHIGS** function.
- "**OPEN PEX**" describes the **OPEN PEX** function.
- "**OPEN WORKSTATION**" describes the **OPEN WORKSTATION** function.
- "**GENERALIZED DRAWING PRIMITIVE**" describes the **GDP**s, which are used to create 2D elements.
- "**GENERALIZED DRAWING PRIMITIVE 3**" describes the **GDP3**s, which are used to create 3D elements.
- "**GENERALIZED STRUCTURE ELEMENT**" describes the **GSE**s, which are used to create implementation-dependent structure elements.
- "**SET HLHSR ID**" describes the **SET HLHSR ID** function.
- "**SET HLHSR MODE**" describes the **SET HLHSR MODE** function.
- "PHIGS Input with ESV Devices" describes PHIGS input and contains examples for the knob box, button box, and Spaceball.
- "Function Numbers" contains a list of the function numbers with the corresponding function name. Function numbers are returned with error messages.
- "Error Messages" contains a list of the error numbers with the corresponding error description. Error numbers are returned with error messages.

- "PHIGS Tables" contains the PHIGS Description Table, PHIGS PLUS Description Table, PHIGS Workstation Description Table, and the PHIGS PLUS Workstation Description Table.
- "C and FORTRAN Bindings" contains a list of the C and FORTRAN bindings for the PHIGS and PHIGS PLUS functions.
- "Sample Programs" contains four programs implemented in two different ways: one way using the Xlib calls, and the other way using the Motif Graphical User Interface.
- "Bibliography" contains a list of PEX references.

This chapter assumes that the reader has a working knowledge of the C programming language, an understanding of the X programming environment, a general understanding of PHIGS, and knowledge of computer graphics principles.

ES/PEX Releases

The ESV Workstation supports PEX (PHIGS Extension to X), which gives the user access to the X Window System, the PHIGS (Programmer's Hierarchical Interactive Graphics System) standard interface, and the proposed PHIGS PLUS (PHIGS Plus Lumière und Surfaces) standard.

PEX has not yet been released by the X Consortium. Evans & Sutherland is a sponsor of the X Consortium, and ES/PEX is based on the PEX-SI from the X Consortium using the PEX protocol level 5.0P. Evans & Sutherland will continue to follow the PEX development, and, upon release of the PEX extension to public domain, will provide a PEX-compatible server on the ESV Workstation.

Since the Application Programmers Interface (API) for the C language to PHIGS and PHIGS PLUS is not yet an official standard, the PEX development work by the X Consortium was initially done using the Sun-defined C language interface. This has been updated to include the current drafts and those comments that are likely to be accepted. This is also the interface currently used by ES/PEX.

It should be emphasized that PEX is still in the development period. Therefore, programs that run under the current release of ES/PEX may have to be recompiled or altered for future releases.

Graphics Standards

There are a number of different computer graphics products that are referred to as "standards." This section discusses several of these and explains how they relate to the ESV Workstation software.

ISO Standards

International Organization for Standardization (ISO) standards are developed by representatives of nations that are a part of ISO. This means that for a particular standard, the representatives attending from a country usually come from the national standard organization with responsibility for that particular area.

The United States is represented by ANSI. If a standard is being developed for computer graphics, the group within ANSI that is responsible for computer graphics will send representatives to ISO computer graphics standards meetings.

Several steps are involved the development process of an ISO standard:

- *New Work Item* (NWI)

  A standard always starts as a NWI, which is a proposal for development of a standard. If the NWI is accepted by the member nations, the task of developing a standard is assigned to a *Standing Committee* (SC) *Working Group* (WG).

- Draft

  The SC/WG produces a draft of the standard. Very often the draft will exist before the NWI is approved, which was the case with both PHIGS and PHIGS PLUS. The SCs and WGs have numbers assigned. For example, the responsibilities for graphics standards is in SC24/WG2.

- *Committee Draft* (CD)

  After a draft is produced by the WG, it is submitted as a CD and is voted on by the member nations. After the vote, comments are processed by the WG and changes are made to the draft.

- *Draft International Standard* (DIS)

  If the vote is for approval and the changes are not substantial, the CD is submitted as a DIS. If substantial changes are made as a result of the comments, the document may go through a second CD ballot before proceeding on to DIS. The DIS vote can only result in editorial changes to the standard.

- *International Standard* (IS)

  After the voting is complete, the editorial changes are made and the document is submitted as an IS.

The PHIGS functional description is an ISO standard (ISO 9592-1:1988), and the PHIGS PLUS functional description (ISO 9592-4:199x) is currently under development in ISO as a standard. The PHIGS standard is a functional description that is independent of any language and is composed of the following parts:

- The PHIGS functional description (Part 1),
- The archive file format description (Part 2),
- The clear text encoding of the archive file (Part 3), and
- The proposed PHIGS PLUS functional description (Part 4).

Along with these four parts is another standard called the *language bindings* for PHIGS. A language binding is a specific description of how the PHIGS functional description is to be represented in a particular programming language. Each of these bindings goes through the standardization process just like PHIGS and PHIGS PLUS.

Figure 1-1 shows the position of PHIGS and related standards in the development process of an ISO standard.

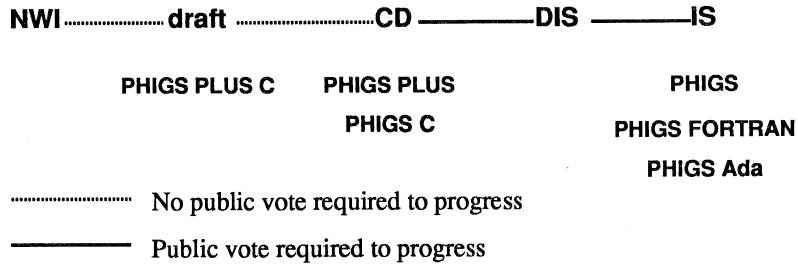


Figure 1-1. Approval steps for ISO standards

ANSI Standards

The approval steps for an American National Standards Institute (ANSI) standard are very similar to the ISO steps. There is also an ANSI procedure for participation in the development of ISO standards. Like ISO, all ANSI standards start with an NWI, followed by a draft, a *draft proposed American National Standard* (dpANS), and finally an *American National Standard* (ANS).

ANSI forms committees in different areas called *Accredited Standards Committees* (ASC) and assigns them a letter and number. These committees

have subgroups responsible for different areas of standards development. A subgroup may be broken down into task groups for specific standards work.

PHIGS and PHIGS PLUS fall under task group 1 of the subgroup on Computer Graphics Standards (H3) of the committee on Information Processing Systems (X3). The full designation for the ASC on the development and maintenance of PHIGS and PHIGS PLUS is X3H3.1. However, since PHIGS PLUS is being developed in ISO, the work of the ASC task group is to provide input and comment to X3 as to how the U.S. should vote on PHIGS PLUS. The X3H3.1 group also provides representatives to the ISO SC24/WG2 meetings.

Evans & Sutherland has representatives on ASC X3H3.1 PHIGS PLUS and X3H3.2 Computer Graphics Reference Models, as well as representatives on ISO SC24/WG2 PHIGS PLUS.

PHIGS was developed as an ANS but was recently replaced by the ISO version, which is identical. This means that there is now only one standard and that the responsibility for maintenance of PHIGS is in ISO.

Also of interest is the X Window System standard. This is not being developed in ISO but rather in the ANSI ASC task group X3H3.6. This standard is called the *X Window Data Stream Definition* and is divided into three parts:

- Functional Specification,
- Data Stream Encoding, and
- KEYSYM Encoding.

The *X Window Data Stream Definition* has just completed the dpANS stage.

U.S. Government Standards

These are not standards that have been developed by the U.S. Government, but rather standards that have been accepted for specification in government contracts.

- Federal Information Processing Standard (FIPS)

  A FIPS is a requirement for inclusion of a particular standard in U.S. Government procurement contracts. PHIGS is not currently a FIPS.

- Military Specifications (Milspec)

  A Milspec is a requirement for inclusion of a particular standard in a military procurement specification. PHIGS is not yet a Milspec.

Industry Standards

The term "industry standard" applied to a product implies the wide-spread use of that product in the industry. The term can lead to some confusion because

anyone can claim to have an "industry standard." There are two industry standards of particular interest on the ESV Workstation.

- X Window System

  The X Window System consists of a *protocol* definition and an *implementation* of that protocol. The protocol is maintained by the X Consortium, which consists of members from different areas of industry and education, and the implementation is maintained by MIT. The current level of the X Window System is X11R4, and this is the version currently supported on the ESV Workstation.

- PEX

  The X Window System specification allows for extensions to be added. There are several extensions that come with the release from MIT. Several years ago, a PEX Consortium was formed to define a protocol for an extension to the X Window System that would work with PHIGS and PHIGS PLUS applications. Sun Microsystems was selected to implement the protocol, and the implementation is currently in release to the X Consortium members.

  Evans & Sutherland is a member of the PEX Consortium. ES/PEX is currently based on the preliminary release R1.

  There are no current plans to process PEX as an ANSI or ISO standard.

ESV Workstation Conformance to PHIGS Standards

The native graphics language of the ESV Series Workstations is PHIGS/PHIGS PLUS. ES/PEX is Evans & Sutherland's implementation of the PEX-SI Release from the X Consortium. The PEX-SI is based on the PHIGS functional description (ISO/IEC 9592-1:1989) and the PHIGS PLUS updated draft functional description (ISO/IEC SC24-N454(20 March 90)).

The ES/PEX implementation provides the ISO Standard (ISO/IEC 9593-4:1990) PHIGS FORTRAN (F77 subset) binding and the ISO (DP on ISO/IEC 9593-4:199x) PHIGS C binding. Future releases of the PHIGS C binding will be modified, as needed, to match the ISO DIS when it becomes available.

Released standards do not yet exist for PHIGS PLUS, or PHIGS PLUS bindings to FORTRAN and C. ES/PEX provides a PHIGS PLUS C binding based on PEX-SI functionality as it matches the Working Draft amendments to ISO/IEC 9593-4:199x. There is no current draft proposal for PHIGS PLUS FORTRAN. ES/PEX provides a PHIGS PLUS FORTRAN binding based on PEX-SI functionality as it matches the SunPHIGS/PHIGS PLUS extensions.

Standards Conformance Tests do not exist for PHIGS C, PHIGS PLUS C or for PHIGS PLUS FORTRAN. The ES/PEX implementation of PHIGS FORTRAN has not been submitted for Conformance Testing pending completion of the following functionality: cell arrays, CIELUV color model, incremental spatial search, modelling clip, metafiles, and stroke device.

What is PEX?

The X Window System is a publicly available protocol that supports 2D graphics. PEX is an extension to the X Window System which allows X to support 3D graphics and gives a user application access to the PHIGS and PHIGS PLUS functions. To understand PEX, we must first look at the X Model and the PHIGS Model. Figure 1-2 shows a simplified schematic of the X Model, figure 1-3 shows a simplified schematic of the PHIGS Model, and figure 1-4 shows a simplified schematic of the PEX Model.

The X Model

The X Model is divided into two parts: the *client* and the *server*. The *server* controls the graphics display and is the interface between the client and the graphics display. The *client* is a user application that may or may not be running on the same system as the graphics display.

The X Window System defines the device-independent *protocol* between the client and the server. Xlib and the X server are sample implementations of the X protocol on a specific system, such as the ESV Workstation, and are device-dependent. A user application calls the Xlib functions, which, in turn, generate data packets defined by the X protocol. The X server translates these data packets into commands that control the graphics display.

The PHIGS Model

PHIGS is a functional specification defining the interface between a user application and the graphics system that displays the application. PHIGS is device-independent.

PHIGS creates application data structures that are stored in an area called the central store structure (CSS), which is also created by PHIGS. The data structures can be posted to one or more *workstations*, or *devices*, which are also created by PHIGS. A *workstation* may or may not be equivalent to a hardware system, such as the ESV Workstation.

The PEX Model

The X Window System permits the addition of extensions, which are also protocols. PEX is one extension. Other extensions on the ESV Workstation include the X Input extension, X Picking extension, X Overlay extension, and the X Multiscreen extension. PEX is an addition to the X protocol which gives a user application access to the X Window System through PHIGS and PHIGS PLUS functions.

With the PEX, X Input, and X Picking extensions added to the X Window System, a user application has access to the Xlib functions, the PHIGS and PHIGS PLUS functions, and the X Input, X Picking, X Overlay, and X Multiscreen functions. An application call to a PHIGS or PHIGS PLUS

function is translated into a PEX protocol data packet which is sent to the X server. The X server recognizes the data packet as being from PEX and transfers it to the PEX routines in the X server for processing. The X Input, X Picking, X Overlay, X Multiscreen data packets are processed in a similar manner.

If a PHIGS workstation is created on the ESV Workstation, the display surface of the PHIGS workstation will be mapped to an X window that is opened on the ESV Workstation. If the PHIGS workstation is closed, the X window will remain open but will revert to a 2D window.

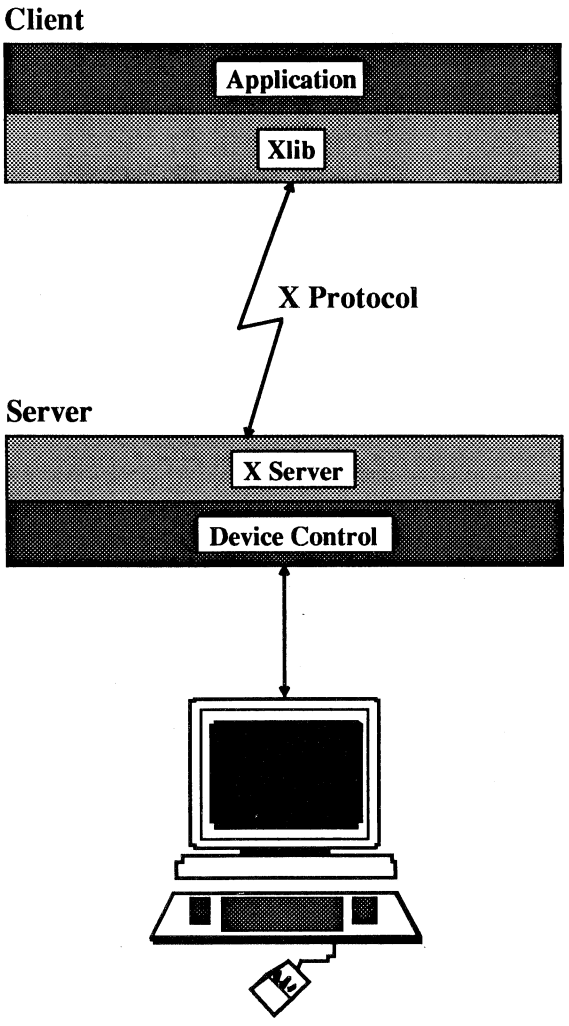


Figure 1-2. The X model

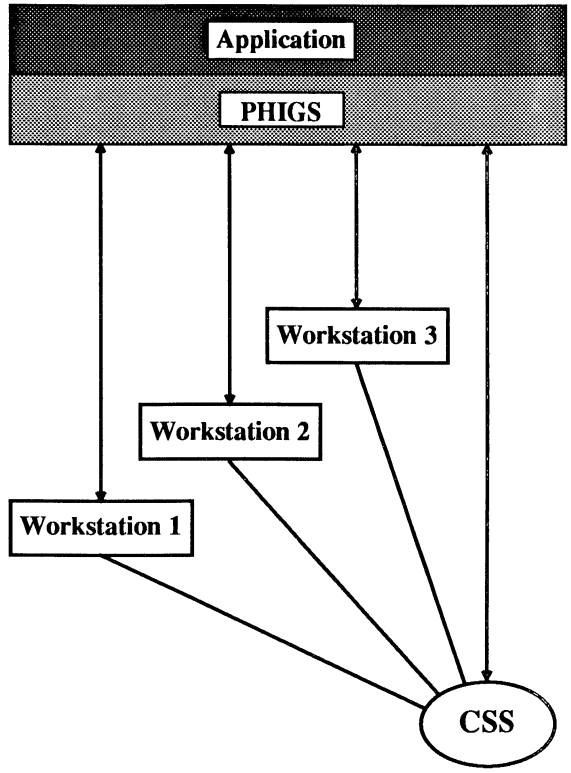


Figure 1-3. The PHIGS model

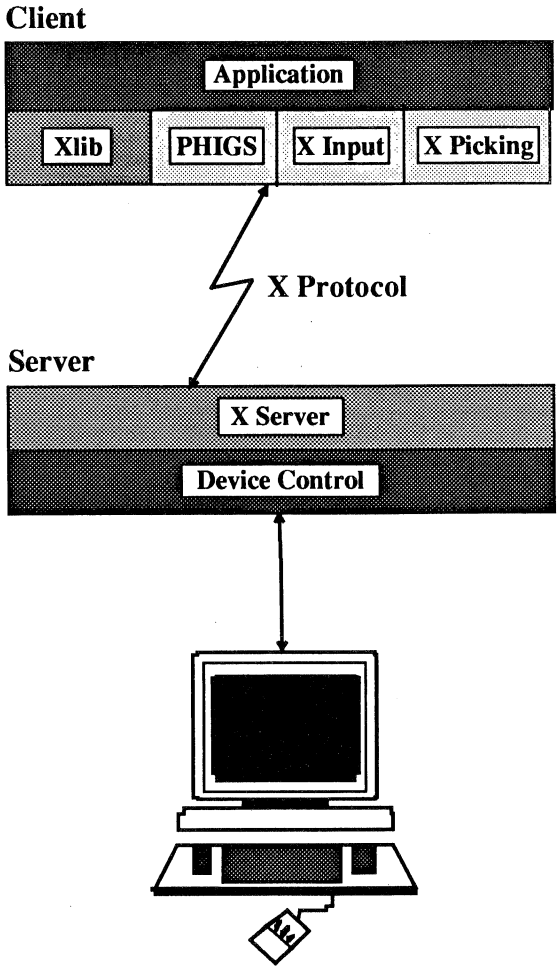


Figure 1-4. The PEX model

Functional Overview

PHIGS and PHIGS PLUS capabilities are expressed by functions and the parameter ranges of those functions. It should be understood that not all implementations will be able to support all capabilities. The PHIGS standard outlines a set of minimum support criteria; and, depending on the implementation, the parameter range provided by a specific implementation may exceed the minimum criteria.

The following list is a very generalized outline of the PHIGS and PHIGS PLUS functions not currently supported by the ESV Workstation.

- B-spline curves and surfaces
- Cell arrays
- Curve and surface approximation by subdivision
- Incremental spatial search
- Input stroke device
- Line width
- Metafiles
- Model space clip
- Patterning and hatching of fill areas
- Normal or dot product shading
- Raster or polygon text
- Text precision other than stroke
- Trimming curves

Interior Styles

The ESV Workstation supports interior styles **Solid**, **Empty**, and **Hollow**.

Light Source Types and Table Indices

The ESV Workstation supports the following light source types:

- **ambient** (1)
- **directional** (2)
- **positional** (3)
- **spot** (4)

Linetypes and Edgetypes

The ESV Workstation supports the following four required edgetypes and linetypes:

- **solid** (1)
- **dashed** (2)
- **dotted** (3)
- **dashed-dotted** (4)

Predefined Polyline Bundles

The ESV Workstation supports the following five linetypes: **solid**, **dashed**, **dotted**, **dot-dashed**, and **long-dashed**. At present, the ESV Workstation does not support the following three linetypes: **dot-dashed-dot-dotted**, **center**, and **phantom**.

Text Font and Precision Pairs

The ESV Workstation supports two distinct text fonts. These two fonts support the **Stroke** precision only.

Workstation Category and Type

The ESV Workstation supports workstations of type **OUTPUT** and **OUTIN**.

PHIGS Functions

In the following list, functions shown in **bold** are currently supported by the ESV Workstation, and functions shown in *italics* are not currently supported by the ESV Workstation. PHIGS PLUS functions are indicated with a "+." The **FILL AREA 3 WITH DATA** function is not supported under the 2.0 server, but it is supported if the thin layer is used.

ADD NAMES TO SET

ANNOTATION TEXT RELATIVE

ANNOTATION TEXT RELATIVE 3

APPLICATION DATA

ARCHIVE ALL STRUCTURES

ARCHIVE STRUCTURE NETWORKS

ARCHIVE STRUCTURES

AWAIT EVENT

BUILD TRANSFORMATION MATRIX

BUILD TRANSFORMATION MATRIX 3

CELL ARRAY

CELL ARRAY 3

CHANGE STRUCTURE IDENTIFIER

CHANGE STRUCTURE IDENTIFIER AND REFERENCES

CHANGE STRUCTURE REFERENCES

CLOSE ARCHIVE FILE

CLOSE PHIGS

CLOSE STRUCTURE

CLOSE WORKSTATION

COMPOSE MATRIX

COMPOSE MATRIX 3

COMPOSE TRANSFORMATION MATRIX

COMPOSE TRANSFORMATION MATRIX 3

COMPUTE FILL AREA SET GEOMETRIC NORMAL +

COPY ALL ELEMENTS FROM STRUCTURE

DELETE ALL STRUCTURES

DELETE ALL STRUCTURES FROM ARCHIVE

DELETE ELEMENT

DELETE ELEMENT RANGE

DELETE ELEMENTS BETWEEN LABELS
DELETE STRUCTURE
DELETE STRUCTURE NETWORK
DELETE STRUCTURE NETWORKS FROM ARCHIVE
DELETE STRUCTURES FROM ARCHIVE
ELEMENT SEARCH
EMERGENCY CLOSE PHIGS
EMPTY STRUCTURE
ERROR HANDLING
ERROR LOGGING
ESCAPE
EVALUATE VIEW MAPPING MATRIX
EVALUATE VIEW MAPPING MATRIX 3
EVALUATE VIEW ORIENTATION MATRIX
EVALUATE VIEW ORIENTATION MATRIX 3
EXECUTE STRUCTURE
EXTENDED CELL ARRAY 3 +
FILL AREA
FILL AREA 3
FILL AREA 3 WITH DATA +
FILL AREA SET
FILL AREA SET 3
FILL AREA SET 3 WITH DATA +
FLUSH DEVICE EVENTS
GENERALIZED DRAWING PRIMITIVE
GENERALIZED DRAWING PRIMITIVE 3
GENERALIZED STRUCTURE ELEMENT
GET CHOICE
GET ITEM TYPE FROM METAFILE
GET LOCATOR
GET LOCATOR 3
GET PICK
GET STRING
GET STROKE

GET STROKE 3

GET VALUATOR

INCREMENTAL SPATIAL SEARCH

INCREMENTAL SPATIAL SEARCH 3

INITIALIZE CHOICE

INITIALIZE CHOICE 3

INITIALIZE LOCATOR

INITIALIZE LOCATOR 3

INITIALIZE PICK

INITIALIZE PICK 3

INITIALIZE STRING

INITIALIZE STRING 3

INITIALIZE STROKE

INITIALIZE STROKE 3

INITIALIZE VALUATOR

INITIALIZE VALUATOR 3

INQUIRE ALL CONFLICTING STRUCTURES

INQUIRE ANNOTATION FACILITIES

INQUIRE ARCHIVE FILES

INQUIRE ARCHIVE STATE VALUE

INQUIRE CHOICE DEVICE STATE

INQUIRE CHOICE DEVICE STATE 3

INQUIRE COLOUR FACILITIES

INQUIRE COLOUR MAPPING FACILITIES +

INQUIRE COLOUR MAPPING METHOD FACILITIES +

INQUIRE COLOUR MAPPING REPRESENTATION +

INQUIRE COLOUR MAPPING STATE +

INQUIRE COLOUR MODEL

INQUIRE COLOUR MODEL FACILITIES

INQUIRE COLOUR REPRESENTATION

INQUIRE CONFLICT RESOLUTION

INQUIRE CONFLICTING STRUCTURES IN NETWORK

INQUIRE CURRENT ELEMENT CONTENT

INQUIRE CURRENT ELEMENT TYPE AND SIZE

INQUIRE CURVE AND SURFACE FACILITIES +
INQUIRE DEFAULT CHOICE DEVICE DATA
INQUIRE DEFAULT CHOICE DEVICE DATA 3
INQUIRE DEFAULT DISPLAY UPDATE STATE
INQUIRE DEFAULT LOCATOR DEVICE DATA
INQUIRE DEFAULT LOCATOR DEVICE DATA 3
INQUIRE DEFAULT PICK DEVICE DATA
INQUIRE DEFAULT PICK DEVICE DATA 3
INQUIRE DEFAULT STRING DEVICE DATA
INQUIRE DEFAULT STRING DEVICE DATA 3
INQUIRE DEFAULT STROKE DEVICE DATA
INQUIRE DEFAULT STROKE DEVICE DATA 3
INQUIRE DEFAULT VALUATOR DEVICE DATA
INQUIRE DEFAULT VALUATOR DEVICE DATA 3
INQUIRE DEPTH CUE FACILITIES +
INQUIRE DEPTH CUE REPRESENTATION +
INQUIRE DIRECT COLOUR MODEL FACILITIES +
INQUIRE DISPLAY SPACE SIZE
INQUIRE DISPLAY SPACE SIZE 3
INQUIRE DISPLAY UPDATE STATE
INQUIRE DYNAMICS OF STRUCTURES
INQUIRE DYNAMICS OF WORKSTATION ATTRIBUTES
INQUIRE EDGE FACILITIES
INQUIRE EDGE REPRESENTATION
INQUIRE EDIT MODE
INQUIRE ELEMENT CONTENT
INQUIRE ELEMENT POINTER
INQUIRE ELEMENT TYPE AND SIZE
INQUIRE ERROR HANDLING MODE
INQUIRE EXTENDED DYNAMICS OF WORKSTATION ATTRIBUTES +
INQUIRE EXTENDED EDGE REPRESENTATION +
INQUIRE EXTENDED INTERIOR FACILITIES +
INQUIRE EXTENDED INTERIOR REPRESENTATION +
INQUIRE EXTENDED PATTERN REPRESENTATION +

INQUIRE EXTENDED POLYLINE FACILITIES +

INQUIRE EXTENDED POLYLINE REPRESENTATION +

INQUIRE EXTENDED POLYMARKER REPRESENTATION +

INQUIRE EXTENDED TEXT REPRESENTATION +

INQUIRE EXTENDED WORKSTATION STATE TABLE LENGTHS +

INQUIRE GENERALIZED DRAWING PRIMITIVE

INQUIRE GENERALIZED DRAWING PRIMITIVE 3

INQUIRE GENERALIZED STRUCTURE ELEMENT FACILITIES

INQUIRE HIGHLIGHTING FILTER

INQUIRE HLHSR FACILITIES

INQUIRE HLHSR MODE

INQUIRE INPUT QUEUE OVERFLOW

INQUIRE INTERIOR FACILITIES

INQUIRE INTERIOR REPRESENTATION

INQUIRE INVISIBILITY FILTER

INQUIRE LIGHT SOURCE FACILITIES +

INQUIRE LIGHT SOURCE REPRESENTATION +

INQUIRE LIST OF AVAILABLE GENERALIZED DRAWING PRIMITIVES

INQUIRE LIST OF AVAILABLE GENERALIZED DRAWING PRIMITIVES 3

INQUIRE LIST OF AVAILABLE GENERALIZED STRUCTURE ELEMENTS

INQUIRE LIST OF AVAILABLE WORKSTATION TYPES

INQUIRE LIST OF COLOUR INDICES

INQUIRE LIST OF COLOUR MAPPING INDICES +

INQUIRE LIST OF DEPTH CUE INDICES +

INQUIRE LIST OF EDGE INDICES

INQUIRE LIST OF INTERIOR INDICES

INQUIRE LIST OF LIGHT SOURCE INDICES +

INQUIRE LIST OF PATTERN INDICES

INQUIRE LIST OF POLYLINE INDICES

INQUIRE LIST OF POLYMARKER INDICES

INQUIRE LIST OF TEXT INDICES

INQUIRE LIST OF VIEW INDICES

INQUIRE LOCATOR DEVICE STATE

INQUIRE LOCATOR DEVICE STATE 3

INQUIRE MODELLING CLIPPING FACILITIES
INQUIRE MORE SIMULTANEOUS EVENTS
INQUIRE NUMBER OF AVAILABLE LOGICAL INPUT DEVICES
INQUIRE NUMBER OF DISPLAY PRIORITIES SUPPORTED
INQUIRE OPEN STRUCTURE
INQUIRE PATHS TO ANCESTORS
INQUIRE PATHS TO DESCENDANTS
INQUIRE PATTERN FACILITIES
INQUIRE PATTERN REPRESENTATION
INQUIRE PHIGS FACILITIES
INQUIRE PICK DEVICE STATE
INQUIRE PICK DEVICE STATE 3
INQUIRE POLYLINE FACILITIES
INQUIRE POLYLINE REPRESENTATION
INQUIRE POLYMARKER FACILITIES
INQUIRE POLYMARKER REPRESENTATION
INQUIRE POSTED STRUCTURES
INQUIRE PREDEFINED COLOUR MAPPING REPRESENTATION +
INQUIRE PREDEFINED COLOUR REPRESENTATION
INQUIRE PREDEFINED DEPTH CUE REPRESENTATION +
INQUIRE PREDEFINED EDGE REPRESENTATION
INQUIRE PREDEFINED EXTENDED EDGE REPRESENTATION +
INQUIRE PREDEFINED EXTENDED INTERIOR REPRESENTATION +
INQUIRE PREDEFINED EXTENDED PATTERN REPRESENTATION +
INQUIRE PREDEFINED EXTENDED POLYLINE REPRESENTATION +
INQUIRE PREDEFINED EXTENDED POLYMARKER REPRESENTATION +
INQUIRE PREDEFINED EXTENDED TEXT REPRESENTATION +
INQUIRE PREDEFINED INTERIOR REPRESENTATION
INQUIRE PREDEFINED LIGHT SOURCE REPRESENTATION +
INQUIRE PREDEFINED PATTERN REPRESENTATION
INQUIRE PREDEFINED POLYLINE REPRESENTATION
INQUIRE PREDEFINED POLYMARKER REPRESENTATION
INQUIRE PREDEFINED TEXT REPRESENTATION
INQUIRE PREDEFINED VIEW REPRESENTATION

INQUIRE RENDERING COLOUR MODEL FACILITIES +
INQUIRE SET OF OPEN WORKSTATIONS
INQUIRE SET OF WORKSTATIONS TO WHICH POSTED
INQUIRE STRING DEVICE STATE
INQUIRE STRING DEVICE STATE 3
INQUIRE STROKE DEVICE STATE
INQUIRE STROKE DEVICE STATE 3
INQUIRE STRUCTURE IDENTIFIERS
INQUIRE STRUCTURE STATE VALUE
INQUIRE STRUCTURE STATUS
INQUIRE SYSTEM STATE VALUE
INQUIRE TEXT EXTENT
INQUIRE TEXT FACILITIES
INQUIRE TEXT REPRESENTATION
INQUIRE VALUATOR DEVICE STATE
INQUIRE VALUATOR DEVICE STATE 3
INQUIRE VIEW FACILITIES
INQUIRE VIEW REPRESENTATION
INQUIRE WORKSTATION CATEGORY
INQUIRE WORKSTATION CLASSIFICATION
INQUIRE WORKSTATION CONNECTION AND TYPE
INQUIRE WORKSTATION STATE TABLE LENGTHS
INQUIRE WORKSTATION STATE VALUE
INQUIRE WORKSTATION TRANSFORMATION
INQUIRE WORKSTATION TRANSFORMATION 3
INTERPRET ITEM
LABEL
MESSAGE
OFFSET ELEMENT POINTER
OPEN ARCHIVE FILE
OPEN PHIGS
OPEN STRUCTURE
OPEN WORKSTATION
NON-UNIFORM B-SPLINE CURVE +

NON-UNIFORM B-SPLINE SURFACE +

POLYLINE

POLYLINE 3

POLYLINE SET 3 WITH DATA +

POLYMARKER

POLYMARKER 3

POST STRUCTURE

QUADRILATERAL MESH 3 WITH DATA +

READ ITEM FROM METAFILE

REDRAW ALL STRUCTURES

REMOVE NAMES FROM SET

REQUEST CHOICE

REQUEST LOCATOR

REQUEST LOCATOR 3

REQUEST PICK

REQUEST STRING

REQUEST STROKE

REQUEST STROKE 3

REQUEST VALUATOR

RESTORE MODELLING CLIPPING VOLUME

RETRIEVE ALL STRUCTURES

RETRIEVE ANCESTORS OF STRUCTURE

RETRIEVE DESCENDANTS OF STRUCTURE

RETRIEVE STRUCTURE IDENTIFIERS

RETRIEVE STRUCTURE NETWORKS

RETRIEVE STRUCTURES

ROTATE

ROTATE X

ROTATE Y

ROTATE Z

SAMPLE CHOICE

SAMPLE LOCATOR

SAMPLE LOCATOR 3

SAMPLE PICK

SAMPLE STRING

SAMPLE STROKE

SAMPLE STROKE 3

SAMPLE VALUATOR

SCALE

SCALE 3

SET ANNOTATION STYLE

SET ANNOTATION TEXT ALIGNMENT

SET ANNOTATION TEXT CHARACTER HEIGHT

SET ANNOTATION TEXT CHARACTER UP VECTOR

SET ANNOTATION TEXT PATH

SET AREA PROPERTIES +

SET BACK AREA PROPERTIES +

SET BACK INTERIOR COLOUR +

SET BACK INTERIOR REFLECTANCE EQUATION +

SET BACK INTERIOR SHADING METHOD +

SET BACK INTERIOR STYLE +

SET BACK INTERIOR STYLE INDEX +

SET BACK PARAMETRIC SURFACE CHARACTERISTICS +

SET CHARACTER EXPANSION FACTOR

SET CHARACTER HEIGHT

SET CHARACTER SPACING

SET CHARACTER UP VECTOR

SET CHOICE MODE

SET COLOUR MAPPING INDEX +

SET COLOUR MAPPING REPRESENTATION +

SET COLOUR MODEL

SET COLOUR REPRESENTATION

SET CONFLICT RESOLUTION

SET CURVE APPROXIMATION CRITERIA +

SET DEPTH CUE INDEX +

SET DEPTH CUE REPRESENTATION +

SET DISPLAY UPDATE STATE

SET EDGE COLOUR +

SET EDGE COLOUR INDEX
SET EDGE FLAG
SET EDGE INDEX
SET EDGE REPRESENTATION
SET EDGETYPE
SET EDGEWIDTH SCALE FACTOR
SET EDIT MODE
SET ELEMENT POINTER
SET ELEMENT POINTER AT LABEL
SET ERROR HANDLING MODE
SET EXTENDED EDGE REPRESENTATION +
SET EXTENDED INTERIOR REPRESENTATION +
SET EXTENDED PATTERN REPRESENTATION +
SET EXTENDED POLYLINE REPRESENTATION +
SET EXTENDED POLYMARKER REPRESENTATION +
SET EXTENDED TEXT REPRESENTATION +
SET FACE CULLING MODE +
SET FACE DISTINGUISHING MODE +
SET GLOBAL TRANSFORMATION
SET GLOBAL TRANSFORMATION 3
SET HIGHLIGHTING FILTER
SET HLHSR IDENTIFIER
SET HLHSR MODE
SET INDIVIDUAL ASF
SET INTERIOR COLOUR +
SET INTERIOR COLOUR INDEX
SET INTERIOR INDEX
SET INTERIOR REFLECTANCE EQUATION +
SET INTERIOR REPRESENTATION
SET INTERIOR SHADING METHOD +
SET INTERIOR STYLE
SET INTERIOR STYLE INDEX
SET INVISIBILITY FILTER
SET LIGHT SOURCE REPRESENTATION +

SET LIGHT SOURCE STATE +
SET LINETYPE
SET LINEWIDTH SCALE FACTOR
SET LOCAL TRANSFORMATION
SET LOCAL TRANSFORMATION 3
SET LOCATOR MODE
SET MARKER SIZE SCALE FACTOR
SET MARKER TYPE
SET MODELLING CLIPPING INDICATOR
SET MODELLING CLIPPING VOLUME
SET MODELLING CLIPPING VOLUME 3
SET OF FILL AREA SET 3 WITH DATA +
SET PARAMETRIC SURFACE CHARACTERISTICS +
SET PATTERN REFERENCE POINT
SET PATTERN REFERENCE POINT AND VECTORS
SET PATTERN REPRESENTATION
SET PATTERN SIZE
SET PICK FILTER
SET PICK IDENTIFIER
SET PICK MODE
SET POLYLINE COLOUR +
SET POLYLINE COLOUR INDEX
SET POLYLINE INDEX
SET POLYLINE REPRESENTATION
SET POLYLINE SHADING METHOD +
SET POLYMARKER COLOUR +
SET POLYMARKER COLOUR INDEX
SET POLYMARKER INDEX
SET POLYMARKER REPRESENTATION
SET RENDERING COLOUR MODEL +
SET STRING MODE
SET STROKE MODE
SET SURFACE APPROXIMATION CRITERIA +
SET TEXT ALIGNMENT

SET TEXT COLOUR +
SET TEXT COLOUR INDEX
SET TEXT FONT
SET TEXT INDEX
SET TEXT PATH
SET TEXT PRECISION
SET TEXT REPRESENTATION
SET TRIMMING CURVE APPROXIMATION CRITERIA +
SET VALUATOR MODE
SET VIEW INDEX
SET VIEW REPRESENTATION
SET VIEW REPRESENTATION 3
SET VIEW TRANSFORMATION INPUT PRIORITY
SET WORKSTATION VIEWPORT
SET WORKSTATION VIEWPORT 3
SET WORKSTATION WINDOW
SET WORKSTATION WINDOW 3
TEXT
TEXT 3
TRANSFORM POINT
TRANSFORM POINT 3
TRANSLATE
TRANSLATE 3
TRIANGLE STRIP 3 WITH DATA +
UNPOST ALL STRUCTURES
UNPOST STRUCTURE
UPDATE WORKSTATION
WRITE ITEM TO METAFILE

Compatibility with Prior Releases

The 2.0 Release is based on the PEX-SI release to the X Consortium and represents a major step towards the stability and interoperability of PEX. The 2.0 Release is based on the 5.0P protocol and the API binding that is the most current and closest to the proposed bindings for PHIGS and PHIGS PLUS. This is also the system that multiple major vendors will be using to demonstrate interoperability during 1991.

The change to the 5.0P protocol means that programs which run on releases prior to the 2.0 Release will not run on the 2.0 Release, and programs which run on the 2.0 Release will not run on releases prior to the 2.0 Release. The protocols are not compatible. For X-only programs there is no protocol change, and they will work on any release.

The 2.0 Release maintains compatibility with the API binding (function calls) to PHIGS and PHIGS PLUS. With the exception of **pgse**, **pgdp** and **pescape**, no change is required. The 2.0 Release provides a thin layer binding which translates the old C-binding calls (now referred to as the *1.3 binding*) to the new C-binding calls. There has been no change in the FORTRAN binding, and for these applications you need only to recompile and link.

- The location of the **include** files changes slightly in the 2.0 Release. This change was made by the X Consortium. The **include** files for the new PHIGS binding are located in the following directory:

  /usr/include/X11/phigs

- The **include** files for the old PHIGS binding are located in the following directory:

  /usr/include/X11/phigs1.3

- To compile your old code you must change the following in your application:

  #include <X11/extensions/phigs.h>

  to

  #include <X11/phigs1.3/phigs.h>

- If you use a compile line option to specify include paths, then add the following to your compile line or **Makefile**:

  -I/usr/include/X11/phigs1.3

- To compile for the new binding, put the following in your source code:

  #include <X11/phigs/phigs.h>

Alternately, you can put the following in your source code:

#include <phigs.h>

and add the following to your compile line:

-I/usr/include/X11/phigs

- Similar changes need to be made to paths to locate **phigs77.h** for FORTRAN programs. **phigs77.h** has moved from **/usr/include/X11/extensions** to **/usr/include/X11/phigs**.

The other change in the 2.0 Release is in the libraries which are available. When using the 1.3 binding, you must link in **libPEXapi1.3.a** prior to **libPEXapi.a**. The graphics libraries on the ESV Workstation are:

libMrm.a	Motif resource manager library
libPEX77.a	PHIGS FORTRAN library
libPEXapi.a	PHIGS latest C library
libPEXapi1.3.a	PHIGS 1.3 binding C library
libPEXt.a	E&S PHIGS toolkit library
libUil.a	Motif UIL library
libX11.a	X graphics library
libXEandSext.a	E&S X extension library
libXau.a	X authorization library
libXaw.a	X Athena widget library
libXdmcp.a	X display manager communications protocol
libXext.a	X extension library
libXinput.a	X input extension library
libXm.a	Motif widget library
libXmu.a	X widget utilities library
libXpick.a	E&S X pick extension library
libXt.a	X Toolkit library
liboldX.a	X old compatibility library

OPEN XPHIGS

Name

OPEN XPHIGS - open and initialize PHIGS in the X environment.

Syntax

```
void
popen_xphigs (error_file,memory,xinfo_mask,xinfo)
char            *error_file;
Plong           memory;
unsigned long   xinfo_mask;
Pxphigs_info    *xinfo;
```

FORTRAN SYNTAX IN 2.0 RELEASE NOTES
POPXP(...)
POPPH (... ?

Required PHIGS Operating States

(PHCL,WSCL,STCL,ARCL)

Description

OPEN XPHIGS is similar to **OPEN PHIGS** but allows specification of additional run-time options. It initializes the API and enables access to the PHIGS functions. **OPEN PHIGS** or **OPEN XPHIGS** must be called prior to calling any other PHIGS functions.

Input Parameters

error_file A pointer to the error file in which to log PHIGS error messages. The error file can be either a pointer to a valid UNIX file name or a null pointer, for example **(char*)0**. A null pointer implies that standard error is to be used as the error file. If a file name is specified, PHIGS will attempt to access the file for writing. If this attempt fails, **OPEN XPHIGS** will fail and the appropriate error will be reported to standard error.

The error file argument passed to **OPEN XPHIGS** will be passed to **ERROR HANDLING. ERROR HANDLING** will also pass this argument to **ERROR LOGGING**. If for some reason, **ERROR LOGGING** cannot access the specified file, the error message will be written to standard error. **ERROR LOGGING** appends messages to the error file; it does not truncate the file when **OPEN XPHIGS** is called. If the specified file does not exist, it will only be created if **ERROR LOGGING** is called.

ERROR LOGGING writes the abstract PHIGS function name, the error number, and an error description to the error file. If for some reason the text for the function name and/or error description can't be determined, **ERROR LOGGING** will just

write the function number and the error number.

memory Not used by the PEX-SI API.

xinfo_mask A bitmask indicating which of the options are being set. This mask is a bitwise OR of one or more of the valid option mask bits and indicates which fields of the **Pxphigs_info** structure are set.

PXPHIGS_INFO_DISPLAY The display pointer: ***display***

PXPHIGS_INFO_RMDB The resource manager: **rmdb**

PXPHIGS_APPL_ID The application name and class: ***appl_id***

PXPHIGS_INFO_ARGS The command line arguments: ***args***

PXPHIGS_INFO_FLAGS_NO_MON
The No Monitor flag: ***flags.no_monitor***

PSPHIGS_INFO_FLAGS_CLIENT_SS
The Force Client SS flag: ***flags.force_client_SS***

xinfo A pointer to a **Pxphigs_info** structure. This structure is used to specify X-related options to PHIGS. **xinfo** is defined in **phigs.h** as follows:

```
typedef struct {
Display          *display;     /* valid display pointer */
XrmDatabase      rmdb;         /* a valid database */
struct {
  char           *name;
  char           *class;
} appl_id;       /* for resolving database attributes */
struct {
  int            *argc_p;
  char           **argv;
} args;          /* for merging args into specified database */
struct {
  unsigned       no_monitor: 1;
                 /* 1 ==> monitor will not be executed */
  unsigned       force_client_SS;
                 /* 1 ==> always use client-side CSS */
}flags;
} Pxphigs_info;
```

Only the fields indicated by ***xinfo_mask*** are examined.

display Specifies both the PHIGS default server and the connection PHIGS is to use when communicating with it. The PHIGS default server holds the master copy of the central structure store, and is where tool workstations are opened if their location is not specified to the **OPEN WORKSTATION** function. PHIGS uses the specified connection for all communication to the default server, even if a different connection is specified for a drawable workstation in a subsequent call to **OPEN WORKSTATION**. PHIGS uses the specified connection for the duration of the PHIGS session; it must not be closed before calling **CLOSE PHIGS**. If ***display*** is not specified, the default server will be the one returned by a call to **XDisplay Name()** with an argument of **NULL**.

rmdb An X resource database handle. PHIGS uses this database to build the default workstation description tables. Database searches are on the resource names and classes listed below. Each search is qualified by the name and class specified in ***appl_id***. If arguments are also specified, PHIGS will merge them into the database prior to searching the database for resources PHIGS recognizes. Any merged arguments will be removed from the argument list. The resources PHIGS recognizes are

Argument String	Resource Name	Resource Class	Type	Valid Values
-display	***display***	***Display***	String	
-bufmode	***bufMode***	***BufMode***	String	single | double
=	***geometry***	***Geometry***	String	
-label	***label***	***Label***	String	
-iconlabel	***iconLabel***	***IconLabel***	String	

The ***display*** resource specifies the name of the default server. The ***bufMode*** resource specifies the default buffering mode, "single" or "double." Single buffering is not available on the ESV. The ***geometry*** resource specifies the default window location and size for **phigs_ws_type_x_tool** workstation types. The geometry is specified in the standard X geometry format:

```
<width>x<height>{+-}<xoffset>{+-}<yoffset>
```

The ***label*** resource specifies the window label for **phigs_ws_type_x_tool** workstation types. The ***iconLabel*** resource specifies the icon label for **phigs_ws_type_x_tool** workstation types. ***appl_id.name*** and ***appl_id.class*** are the application name and class to use when resolving resource database attributes. If not specified, the name ***phigs*** and class ***Phigs*** are used.

args.argc_p

A pointer to the argument count.

args.argv The array of command line arguments. The arguments are searched for attributes recognized by PHIGS and are merged into the specified database, if any.

flags.no_monitor

Indicates whether the PHIGS Monitor (PM) is executed during **popen_phigs()**. The PM is a separate program started by PHIGS that handles window events and PHIGS input for PHIGS workstations. PHIGS has complete control over this program; no user action is required to deal with it other than to indicate if it should be used or not. If the PM is not executed (***flags.no_monitor*** **= 1**), PHIGS will not monitor and respond to X window events nor will it provide any PHIGS workstations that support PHIGS input devices. PHIGS workstations that support PHIGS input devices. All the predefined workstation types will be of category **OUTPUT** (output only). If the PM is executed (***flags.no_monitor*** **= 0**), PHIGS will monitor relevant window events and will provide predefined workstation types of category **OUTIN** (input and output).

flags.force_client_SS

Indicates whether the API should use client side structure storage even if server side structure storage is available in X servers with the PEX extension. ***flags.force_client_SS*** **= 1** indicates that client side structure storage should always be used. ***flags.force_client_SS*** **= 0** indicates that server side structure storage should be used if available. Client side structure storage is not available on the ESV.

Execution

OPEN XPHIGS sets internal state information and then calls **popen_phigs()**.

OPEN PEX

Name

OPEN PEX - open and initialize the PEX environment - 1.3 compatibility only

Syntax

```
void
popenpex(error_file,memory,xinfo)
Pchar          *error_file;
size_t         memory;
Popenpexinfo   *xinfo;
```

Required PHIGS Operating States

(PHCL,WSCL,STCL,ARCL)

Description

OPEN PEX is similar to **OPEN PHIGS**. It initializes the API and enables access to the PHIGS functions. It also allows specification of run-time options and the merging of resource manager database (RMDB) attributes which **OPEN PHIGS** does not allow. **OPEN PHIGS** or **OPEN PEX** must be called prior to calling any other PHIGS functions.

Input Parameters

error_file A pointer to the error file where PEX error messages are logged. The error file can be either a pointer to a valid UNIX file name or a null pointer (*e.g.*, **(Pchar*)0**). A null pointer implies that standard error is to be used as the error file. If a file name is specified, PEX will attempt to access the file for writing. If this attempt fails, **OPEN PEX** will fail and the appropriate error will be reported to standard error.

The ***error_file*** argument passed to **OPEN PEX** will be passed to **ERROR HANDLING**. **ERROR HANDLING** will also pass this argument to **ERROR LOGGING**. If for some reason **ERROR LOGGING** cannot access the specified error file, the error message will be written to standard error. **ERROR LOGGING** appends messages to the error file, and it does not truncate the file when **OPEN PEX** is called. If the specified file does not exist, it will only be created if **ERROR LOGGING** is called.

ERROR LOGGING writes the PHIGS or PHIGS PLUS function name, the error number, and an error description to the error file. If for some reason the text for the function name and/or error description cannot be determined, **ERROR LOGGING** will just write the function number and the error number.

memory Ignored by the API.

xinfo A pointer to a **Popenpexinfo** structure. This structure is used to specify X-related options to PHIGS. ***xinfo*** is defined in **phigs.h** as follows:

```
typedef struct {
        Display             *display;
        DisXrmDatabase      rdb;
        char                *name;
        char                *classname;
        int                 *argc_p;
        char                **argv;
        struct {
          unsigned          no_monitor: 1;
          unsigned          force_client_SS;
        } flags;
} Popenpexinfo;
```

Popenpexinfo Parameters

Note: All fields must be set to a valid value or NULL.

display The display pointer for the server connection to use, rather than creating a connection to the default server. It must be either NULL or a valid display pointer. If a valid display pointer is specified, then the API will use it as the start-up server. If ***display*** is NULL, the API will attempt to connect to the default server.

rdb An X resource database handle. PHIGS uses this database to build the default workstation description tables.

name The application name to use when resolving resource database attributes. If not specified, the name ***phigs*** is used.

classname The class to use when resolving resource database attributes. If not specified, the class ***Phigs*** is used.

argc_p A pointer to the argument count.

argv The array of command line arguments. The argument is searched for attributes recognized by PHIGS, and they are merged into the specified database, if any.

flags.no_monitor
Should be set to 1 if X is going to be used for input, or to 0 if PHIGS input is used.

flags.force_client_SS
Should be set to 0.

Execution

OPEN PEX fills in a 2.0 **Pxphigs_info** structure with values from the **Popenpexinfo** structure. It then calls **popen_xphigs**. Previously the ***rdb***, ***name***, ***classname***, ***argc_p***, ***argv***, and ***flags.force_client_SS*** fields were ignored. With the 2.0 release, these fields must be either used or initialized to the default values below.

When **popenphigs**() is used to initialize the API it behaves as though the following values were assigned to the fields of **Popenpexinfo**:

****display***	=	NULL
rdb	=	NULL
****name***	=	NULL
****classname***	=	NULL
****argc_p***	=	NULL
*****argv***	=	NULL
flags.no_monitor	=	0
flags.force_client_SS	=	0

OPEN WORKSTATION

Name

OPEN WORKSTATION - create a PHIGS workstation

Syntax

```
void
popen_ws(ws_id,conn_id,ws_type)
Pint            ws_id;
char            *conn_id;
Pint            ws_type;
```

FORTRAN SYNTAX IN 2.0 RELEASE NOTES

POPWK(...)

Required PHIGS Operating States

(PHOP,*,*,*)

Description

OPEN WORKSTATION opens a workstation of the specified workstation type. The workstation state list is created and initialized to conform as nearly as possible to the workstation description table associated with the specified workstation type. If the workstation is successfully opened, the PHIGS workstation state variable is set to **WSOP**.

PEX-SI supports two predefined workstation types, x_tool and x_drawable. Their characteristics are described below. In addition, the application can create and modify its own workstation types with the **WORKSTATION TYPE CREATE** and **S-2 WORKSTATION TYPE SET** functions.

If the workstation is opened successfully, a specific workstation type is created and associated with the open workstation. This specific workstation type contains the workstation description table that describes the capabilities of the opened workstation. The specific workstation type can be retrieved with **INQUIRE WORKSTATION CONNECTION AND TYPE**.

Input Parameters

ws_id The workstation identifier to be associated with this workstation. This value is used to identify the workstation in subsequent PHIGS function calls.

conn_id A pointer to the connection identifier of the workstation. The type of value to use depends on the workstation type.

x_tool If the ***conn_id*** is NULL, a window is created on the default server. If the ***conn_id*** is not NULL, it is interpreted as a display name. A window for the workstation is created on the named server, provided that server supports the X extension. An example of using this would be

```
Popen_ws(ws_id,(char*)"unix:0",phigs_ws_type_x_tool);
```

x_drawable

The connection identifier must be a pointer to a **Pconnid_x_drawable** structure, cast to a **char***. **Pconnid_x_drawable** is defined in **phigs.h**.

ws_type The type of workstation to open. Recognized types are described fully in the "Available Workstation Types" section below. They are declared in **phigs.h**. A short summary is listed here.

phigs_ws_type_x_tool
PHIGS creates an X window for the workstation on a specified or default server.

phigs_ws_type_x_drawable
PHIGS uses a specified X window for the workstation.

Execution

OPEN WORKSTATION opens a PHIGS workstation of the specified type and associates it with the specified workstation identifier.

When a workstation is opened, PHIGS creates a copy of the workstation type specified in the **OPEN WORKSTATION** call and binds it to the opened workstation. This copy is called the specific workstation type. The workstation description table (WDT) of this specific type is checked against the capabilities of the server and window PHIGS is using for the workstation. If the capabilities specified by the WDT cannot be realized with that server and window, PHIGS modifies the WDT of the specified workstation type to reflect the available capabilities. The workstation type parameter to **OPEN WORKSTATION** is not modified; only the specific workstation type is (potentially) modified. The specific workstation type bound to a workstation can be retrieved with the PHIGS function **INQUIRE WORKSTATION CONNECTION AND TYPE**.

Some of a workstation type's workstation description table values can be changed by the application prior to opening a workstation of that type. See **WORKSTATION TYPE CREATE** and **WORKSTATION TYPE SET** for more information on this.

Available Workstation Types

phigs_ws_type_x_tool

PHIGS creates an X window on a specified or default server and uses it for the PHIGS workstation's display surface. If the PHIGS Monitor is running (see **OPEN PHIGS**), the default category of this workstation type is **OUTIN**, which indicates that both input and output are available; otherwise the default category is **OUTPUT**

If the category is **OUTIN**, PHIGS creates an additional input-only window that it uses to detect pointer events for input devices. This window is transparent, overlies the output window completely, and duplicates any size and position changes made to the output window.

Many of the characteristics of an x_tool workstation type, such as its size and position, can be set prior to opening the workstation. See the **WORKSTATION TYPE CREATE** and **WORKSTATION TYPE SET** manual pages for a complete list of the modifiable characteristics and their default settings.

phigs_ws_type_x_drawable

PHIGS uses an application-specified X window the for PHIGS workstation's display source. The window to use is specified in the connection identifier parameter. The window must be open and associated with a server that supports the PEX extension.

If the PHIGS Monitor is running (see **OPEN PHIGS**), the default category of this workstation type is **OUTIN**, which indicates that both input and output are available; otherwise, the default category is **OUTPUT**.

Some of the characteristics of an x_tool drawable workstation type, such as its size and position, can be set prior to opening the workstation. See **WORKSTATION TYPE CREATE** and **WORKSTATION TYPE SET** man pages for a complete list of the modifiable characteristics and their default settings.

Window System Interaction

Workstation DC limits correspond to the window size used by the PHIGS workstation when it is opened. The units are drawable coordinates. When the API responds to a window resize event, more or less of the window will be exposed; the PHIGS output will not be scaled. Decreases in size cause portions of the PHIGS output to be clipped away if the new size is less than the PHIGS viewport limits. Size increase beyond the viewport limits will not reveal any additional PHIGS output.

PHIGS automatically redraws the PHIGS workstation when portions of it are exposed, such as when it is brought to the top of other windows or moved from an iconic state to an open state. This redrawing may potentially change portions of the workstation state list by making the state of visual representation correct and making all requested entries current.

For both tool and drawable workstations, a **DestroyNotify** event on the workstation's window will likely cause PHIGS to exit. A **DestroyNotify** event is generated either by the application or more likely by the operator when he destroys or "Quits" the window.

GENERALIZED DRAWING PRIMITIVE

Name

GENERALIZED DRAWING PRIMITIVE (GDP) - create 2D **GDP** elements (none available on ESV Workstations)

Syntax

```
void
pgdp ( point_list, gdp_id, gdp_data )
Ppoint_list     *point_list;      array of points
Pint            gdp_id;           GDP function identifier
Pgdp_data       *gdp_data;        data record pointer
```

Required PHIGS Operating States

(PHOP, *, STOP, *)

Input Parameters

point_list A pointer to a structure containing a list of *x* and *y* values in Modelling Coordinates (MC). **Ppoint _list** is defined in **phigs.h** as:

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

Ppoint is defined in **phigs.h** as:

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

point_list is ignored for ESV Series Workstations.

gdp_id An integer specifying the **GDP** to be performed.

gdp_data A pointer to a **Pgdp_data** union containing the information needed to perform the function specified by ***gdp_id***. **Pgdp_data** is defined in **phigs.h** as:

```
typedef union {
        struct {
        Pint    unused;
} gdp_r1;
        Pdata   unsupp;     /* unsupported GDP data record*/
                            /* implementation dependent */
} Pgdp_data;
```

The ***unsupp*** field in the **Pgse_data** structure is of type **Pdata** which is defined in **phigs.h**:

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

Description

GDP creates an implementation-dependent drawing primitive. On the ESV Series Workstations there are no 2D drawing primitives available.

Execution

If the current edit mode is insert, the structure element created by the **GDP** function is inserted into the open structure after the element pointed to by the element pointer. If the current edit mode is replace, the **GDP** element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new structure element.

Support for **GDP**s is implementation- and workstation-dependent. On the ESV Series Workstations there are no 2D drawing primitives available.

Errors

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

GENERALIZED DRAWING PRIMITIVE 3

Name

GENERALIZED DRAWING PRIMITIVE 3 (GDP3) - create 3D **GDP** elements

Syntax

```
void
pgdp3 ( point_list, gdp3_id, gdp_data )
Ppoint_list3  *point_list;      array of points
Pint          gdp3_id;          GDP function identifier
Pgdp_data3    *gdp_data;        data record pointer
```

Required PHIGS Operating States

(PHOP, *, STOP, *)

Input Parameters

point_list A pointer to a structure containing a list of *x* and *y* values in Modelling Coordinates (MC).

Ppoint _list3 is defined in **phigs.h** as:

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

where:

num_points Number of points passed in the points parameter. This is ignored for ESV Series Workstations.

points A pointer to a list ***num_points*** long of **Ppoint** structures containing *x* and *y* values in Modelling Coordinates (MC).This is ignored for ESV Series Workstations.

Ppoint3 is defined in **phigs.h** as:

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

gdp3_id An integer specifying the **GDP** to be performed. The following **GDP**s are defined for the ESV workstation:

PES_GDP_SPHERE
PES_GDP_SPHERE_RADIUS
PES_GDP_SPHERE_COLR
PES_GDP_SPHERE_RADIUS_COLR
PES_GDP_CYLINDER
PES_GDP_CYLINDER_RADIUS
PES_GDP_CYLINDER_COLR
PES_GDP_CYLINDER_RADIUS_COLR

gdp_data A pointer to a **Pgdp_data** union containing the information needed to perform the function specified by ***gdp3_id***.

Pgdp_data is defined in **phigs.h** as:

```
typedef union {

        struct {
        Pint        unused;
        } gdp_r1;
        Pdata       unsupp;      /* unsupported GDP data record */
                                 /* implementation dependent */
} Pgdp_data;
```

The ***unsupp*** field in the **Pgse_data** structure is of type **Pdata** which is defined in **phigs.h**:

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

Description

GDP creates an implementation-dependent drawing primitive. On the ESV Series Workstations the following **GDP**s are available:

- Spheres with inherited colour and radius.
  (PES_GDP_SPHERE)
- Spheres with specified radius and inherited colour.
  (PES_GDP_SPHERE_RADIUS)
- Spheres with specified colour and inherited radius.
  (PES_GDP_SPHERE_COLR)
- Spheres with specified radius and colour.
  (PES_GDP_SPHERE_RADIUS_COLR)
- Cylinders with inherited colour and radius.
  (PES_GDP_CYLINDER)
- Cylinders with specified radius and inherited colour.
  (PES_GDP_CYLINDER_RADIUS)
- Cylinders with specified colour and inherited radius.
  (PES_GDP_CYLINDER_COLR)
- Cylinders with specified colour and radius.
  (PES_GDP_CYLINDER_RADIUS_COLR)

Execution

If the current edit mode is insert, the structure element created by the **GDP** function is inserted into the open structure after the element pointed to by the element pointer. If the current edit mode is replace, the **GDP** element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new structure element.

Support for **GDP**s is implementation- and workstation-dependent.

Individual GDPs

For convenience, each individual **GDP** is provided its own defined structures to hold data unique to it. These structures should be pointed at by the ***data*** field in the **Pdata** structure described above. Following is a description of each **GDP** and its associated data types and definitions. All of these structures are defined in **esgdp.h**.

List of Spheres

Multiple spheres may be specified in a single **gdp3** call. All sphere elements are characterized by a list of 3D center points that define sphere centers. Spheres have attributes of ***radius***, ***colour***, and ***precision***. All spheres are drawn as surfaces. Sphere ***precision*** is an attribute established by a **GSE** element that defines how accurately spheres are drawn. Spheres that are drawn with lower precision values are drawn faster but look less like spheres than spheres drawn with higher precision values.

There are four forms of the Spheres element. The most simple form holds only a list of center points of spheres. A second form holds a center and a radius for each sphere. A third form holds a center and a colour for each sphere. The fourth form contains both a center, a colour and a radius for each sphere. The four different forms of the sphere element all start with a common header structure, defined as follows:

```
typedef struct {
    Pint    colr_model;  /* colour model */
    Pint    count;       /* number of sphere definitions */
 /* List of sphere structures */
} Psphere_data;
```

Simple List of Spheres

The Simple List of Spheres element has only center point data. The sphere colour comes from the current surface ***colour*** attribute. The sphere radius is taken from the Sphere Radius Attribute that is established by a **GSE** node (see the function **pgse**). The structure supporting the Simple List of Spheres element is:

```
/* Simple Spheres */
typedef struct _Psphere_simple_ {
    Psphere_data        head;
    Psphere             data[1]; /* Variable length array of centers */
} Psphere_simple;
```

Note that the ***data*** field in the above structure is defined with only 1 array element, but it is actually used as a variable length array. (This is the only way to name structure fields of variable length arrays in the C language.) The field is provided for the convenience of application programmers who prefer to access their data by array element name rather than by doing pointer arithmetic on a pointer variable.

A macro is also provided to determine the amount of memory needed for a **Psphere_simple** structure that holds ***n*** sphere center points:

```
/* Macro to calculate size of Simple Spheres structure */
/* n is the number of spheres defined in the list */
#define sz_Psphere_simple(n)
((n-1) * sizeof(Psphere) + sizeof(Psphere_simple));
```

List of Spheres With Radius

The List of Spheres With Radius element has center point data and sphere radii. The sphere colour comes from the current surface ***colour*** attribute. The sphere precision is taken from the Sphere Precision Attribute that is established by a **GSE** node (see the function **pgse**). The structure supporting the List of Spheres With Radius element is:

```
/* Spheres with radii */
typedef struct {
    Ppoint3    center;          /* center of sphere */
    Pfloat     radius;          /* radius of sphere */
} Psphere_radius;
typedef struct _Psphere_w_radius_ {
    Psphere_data    head;
    Psphere_radius  data[1]; /* Variable length array of centers and radii */
} Psphere_w_radius;
/* Macro to calculate size of Spheres With Radius structure */
/* n is the number of spheres defined in the list */
#define sz_Psphere_w_radius(n)
((n-1)*sizeof(Psphere_radius) + sizeof(Psphere_w_radius));
```

List of Spheres With Colour

The List of Spheres With Colour element has center point data and sphere colours. The sphere radius comes from the current sphere ***radius*** attribute. The sphere precision is taken from the Sphere Precision Attribute that is established by a **GSE** node (see the function **pgse**). The structure supporting the List of Spheres With Colour element is:

```
/* Spheres with colour */
typedef struct {
    Ppoint3   center;          /* center of sphere */
    Pcoval    colr;            /* colour of sphere */
} Psphere_colr;
typedef struct _Psphere_w_colr_ {
    Psphere_data head;
    Psphere_colr data[1]; /* Variable length array of centers and colour */
} Psphere_w_colr;
/* Macro to calculate size of Spheres With Colour structure */
/* n is the number of spheres defined in the list */
#define sz_Psphere_w_colr(n)
((n-1)*sizeof(Psphere_colr) + sizeof(Psphere_data));
```

List of Spheres With Radius and Colour

The List of Spheres With Radius and Colour element has center point data, sphere radii and sphere colours. Each sphere radius and sphere colour comes from data within the element. The sphere precision is taken from the Sphere Precision Attribute that is established by a **GSE** node (see the function **pgse**). The structure supporting the List of Spheres With Radius and Colour element is:

```
/* Spheres with radius and colour. */
typedef struct {
    Ppoint3         center;         /* center of sphere */
    Pfloat          radius;         /* radius of sphere */
    Pcoval          colr;           /* colour of sphere */
} Psphere_radius_colr;
typedef struct _Psphere_w_radius_colr_ {
    Psphere_data            head;
     Psphere_radius_colr    data[1]; /* Variable length array of centers,
                                        radii, colr */
} Psphere_w_radius_colr;
/* Macro to calculate size of Spheres With Radius and Colour structure */
/* n is the number of spheres defined in the list */
#define sz_Psphere_w_radius_colr(n)
((n-1)*sizeof(Psphere_radius_colr) + sizeof(Psphere_w_radius_colr));
```

Cylinders

Multiple cylinders may be specified in a single **gdp3** call. Cylinder **GDP**s are defined as "list of cylinder" lists. A single list of cylinders is defined much like a polyline. The cylinders in the list are connected end to end by a common center point on their end caps. Each cylinder **GSE** element can have multiple lists of cylinders within it.

Like spheres, cylinders can be defined very simply by specifying only the centers of their capping circles, taking their ***radius*** and ***colour*** attributes from attribute elements previously defined in the structure. Or, they can include within their definition their colour, or radius, or both. Cylinders have attributes of ***radius***, ***colour***, and ***precision***. All cylinders are drawn as surfaces.

Cylinder ***precision*** is an attribute established by a **GSE** element that defines how accurately cylinders are drawn. Cylinders that are drawn with lower precision values are drawn faster but look less like cylinders than cylinders drawn with higher precision values.

There are four forms of the Cylinders **GDP** element. The most simple form holds only a list of center points of cylinder end or capping circles. A second form holds a center and a radius for each cylinder end. A third form holds a center and a colour for each cylinder. The fourth form contains both a center, a colour and a radius for each cylinder. The data defining the cylinder element must reside in contiguous memory. The four different forms of the cylinder element all start with a common header structure, defined as follows:

```
typedef struct {
    Pint        colr_model;  /* colour model */
    Pint        num_lists;   /* number of lists of cylinders */
/* List of cylinder lists */
} Pcyl_list_data;

typedef struct {
    Pint        num_cyl_ends; /* number of cylinder end points */
/* List of cylinder structures */
} Pcyl_list;

typedef struct {
    Ppoint3     center;      /* center of cylinder */
} Pcylinder;
```

Simple Cylinders

The most simple Cylinder element contains only 3D end points. The cylinder colour comes from the current surface ***colour*** attribute. The cylinder radius is inherited from the Cylinder Radius Attribute that is established by a **GSE** node (see the function **pgse**). Each list within the simple list of cylinder lists has the following structure:

```
typedef struct _Pcyl_list_simple_ {
    Pcyl_list   head;
    Pcylinder   data[1];      /* list of endpoints */
} Pcyl_list_simple;
```

The complete data necessary to create a simple Cylinder element consists of the Cylinder header structure **Pcyl_list_data** described above followed by the desired number of **Pcyl_list_simple** structures.

The size of any given list with ***n*** cylinders within the list of lists of a simple Cylinder element can be determined with the following macro where ***n*** is the number of cylinders defined in the list:

```
#define sz_Pcyl_list_simple(n) (sizeof(Pcyl_list_simple) + (n-1) *
sizeof(Pcylinder))
```

Cylinders With Radius

The Cylinder With Radius element can contain a radius along with 3D end points. The cylinder colour comes from the current surface ***colour*** attribute. The cylinder precision is taken from the Cylinder Precision Attribute that is established by a **GSE** node (see the function **pgse**). A pointer to a cylinder list within the list of cylinder lists can be cast as the following type:

```
typedef struct _Pcyl_list_radius_ {
    Pcyl_list          head;
    Pcylinder_radius   data[1];      /* list of endpoints and radii */
} Pcyl_list_radius;
```

where the structure **Pcylinder_radius** is defined as:

```
typedef struct {
    Ppoint3   center;          /* center of cylinder */
    Pfloat    radius;          /* radius of cylinder */
} Pcylinder_radius;
```

The complete data necessary to create a Cylinder With Radius element consists of the Cylinder header structure **Pcyl_list_data** described above followed by the desired number of **Pcyl_list_radius** structures.

The size of any given list within the list of lists of a Cylinder With Radius element can be determined with the following macro:

```
#define sz_Pcyl_list_radius(n)
(sizeof(Pcyl_list_radius) + (n-1) * sizeof(Pcylinder_radius))
```

Cylinders With Colour

The Cylinder With Colour element can contain a Colour along with 3D end points. The cylinder radius is inherited from the Cylinder Radius Attribute established by a **GSE** element. The cylinder precision is taken from the Cylinder Precision Attribute that is established by a **GSE** element (see the function **pgse**). A pointer to a cylinder list within the list of cylinder lists can be cast as the following type:

```
typedef struct _Pcyl_list_Colr_ {
    Pcyl_list       head;          /*# of points in the cylinder */
    Pcylinder_colr  data[1];       /* list of endpoints and colours */
} Pcyl_list_colr;
```

where the structure **Pcylinder_colr** is defined as:

```
typedef struct {
    Ppoint3   center;      /* center of cylinder */
    Pcoval    colr;        /* colour of cylinder */
} Pcylinder_colr;
```

The complete data necessary to create a Cylinder With Colour element consists of the Cylinder header structure **Pcyl_list_data** described above followed by the desired number of **Pcyl_list_colr** structures.

The size of any given list within the list of lists of a Cylinder With Colour element can be determined with the following macro, where ***n*** is the number of cylinders defined in the list:

```
#define sz_Pcyl_list_colr(n)
(sizeof(Pcyl_list_colr) + (n-1) * sizeof(Pcylinder_colr))
```

Cylinders With Radius and Colour

The Cylinder With Radius and Colour element can contain a radius and colour along with 3D end points. The cylinder precision is taken from the Cylinder Precision Attribute that is established by a **GSE** element (see the function **pgse**). A pointer to a cylinder list within the list of cylinder lists can be cast as the following type:

```
typedef struct _Pcyl_list_radius_colr_ {
    Pcyl_list                 head;       /* # of points in the cylinder */
    Pcylinder_radius_colr     data[1];    /* list of cyl data */
} Pcyl_list_radius_colr;
```

where the structure **Pcylinder_radius_colr** is defined as:

```
typedef struct {
    Ppoint3     center;       /* center of cylinder */
    Pfloat      radius;       /* radius of cylinder */
    Pcoval      colr;         /* colour of cylinder */
} Pcylinder_radius_colr;
```

The complete data necessary to create a Cylinder With Radius and Colour element consists of the Cylinder header structure **Pcyl_list_data** described above followed by the desired number of **Pcyl_list_radius_colr** structures.

The size of any given list within the list of lists of a Cylinder With Radius and Colour element can be determined with the following macro, where ***n*** is the number of cylinders defined in the list:

```
#define sz_Pcyl_list_radius_colr(n)
(sizeof(Pcyl_list_radius_colr) + (n-1) * sizeof(Pcylinder_radius_colr))
```

Errors

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

GENERALIZED STRUCTURE ELEMENT

Name

GENERALIZED STRUCTURE ELEMENT (GSE) - create a **GSE**

Syntax

```
void
pgse ( id, gse )
Pint          id;          GSE identifier
Pgse_data     *gse;        GSE data record
```

Required PHIGS Operating States

(PHOP, *, STOP, *)

Input Parameters

id The identifier of the generalized structure element to insert. Recognized identifiers defined in **esgdp.h** are:

- **PES_GSE_SPHERE_RADIUS**
- **PES_GSE_SPHERE_DIVISIONS**
- **PES_GSE_CYLINDER_RADIUS**
- **PES_GSE_CYLINDER_DIVISIONS**
- **PES_GSE_STEREO_VIEW_INDICES**
- **PES_GSE_FILLAREA_TOLERANCE**
- **PES_GSE_FRONT_BACK_DISTINGUISH**
- **PES_GSE_POLYLINE_QUALITY**
- **PES_GSE_LINEPATTERN_MASK**
- **PES_GSE_EDGEPATTERN_MASK**
- **PES_GSE_INFORMATION**
- **PES_GSE_TRANSPARENCY**

gse A pointer to a **Pgse_data** union containing the information needed to perform the function specified by ***id***. **Pgse_data** is defined in **phigs.h** as:

```
typedef union {
struct {
   Pint     unused;
   } gse_r1;
   Pdata    unsupp;          /* Unsupported GSE data */
} Pgse_data;
```

The ***unsupp*** field in the **Pgse_data** structure is of type **Pdata** which is defined in **phigs.h**.

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

Description

GSE creates an implementation-dependent structure element. On the ESV Series Workstations a **GSE** element may be used to:

- Set the radius of spheres.
  (PES_GSE_SPHERE_RADIUS)
- Set the precision used to render spheres.
  (PES_GSE_SPHERE_DIVISIONS)
- Set the radius of cylinders.
  (PES_GSE_CYLINDER_RADIUS)
- Set the precision used to render cylinders.
  (PES_GSE_CYLINDER_DIVISIONS)
- Set the left and right-eye view indices for stereo.
  (PES_GSE_STEREO_VIEW_INDICES)
- Set the offset distance that separates polylines and fillareas that lie in the same plane.
  (PES_GSE_FILLAREA_TOLERANCE)
- Set the offset value that separates fillarea front and back faces.
  (PES_GSE_FRONT_BACK_DISTINGUISH)
- Set the polyline quality as jaggy or smooth.
  (PES_GSE_POLYLINE_QUALITY)
- Set the line pattern mask.
  (PES_GSE_LINEPATTERN_MASK)
- Set the edge pattern mask.
  (PES_GSE_EDGEPATTERN_MASK)
- Establish a traversal information ID elementor matrix information element.
  (PES_GSE_INFORMATION)
- Set the surface transparency attribute.
  (PES_GSE_TRANSPARENCY)

Execution

If the current edit mode is insert, then **GSE** is inserted into the currently open structure after the element currently pointed to by the element pointer. If the edit mode is replace, then **GSE** replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

Individual GSEs

Each individual **GSE** is provided its own structure type to hold the data unique to it. The structure should be pointed at by the ***data*** field in the **Pdata** structure described above. Following is a description of each **GSE** and its associated data types and definitions. All of these are structures are defined in **esgdp.h**.

Sphere Radius

The Sphere Radius **GSE** establishes the radii of all spheres that do not have radius data as part of the sphere **GDP** element. The radius should be placed in the ***radius*** field of the following structure:

```
typedef struct {
    Pfloat      radius;     /* default radius for spheres */
} Pgse_sphere_radius_data;
```

Sphere Divisions

The Sphere Divisions **GSE** establishes the precision with which spheres will be drawn. Spheres will be drawn with ***div*** number of latitude lines and a corresponding number of longitude lines. The number of divisions should be placed in the ***div*** field of the following structure:

```
typedef struct {
    Pint        div;        /* number of lat. and long. divisions */
} Pgse_sphere_div_data;
```

Cylinder Radius

The Cylinder Radius **GSE** establishes the radii of all cylinders that do not have radius data as part of the cylinder structure element. The radius should be placed in the ***radius*** field of the following structure:

```
typedef struct {
    Pfloat      radius;     /* default radius for cylinders */
} Pgse_cyl_radius_data;
```

Cylinder Divisions

The Cylinder Divisions **GSE** establishes the number of sides (2****div*** + 1) that the cylinder will be broken into for rendering. A number less than 0 defaults to a reasonable value. The number of divisions should be placed in the ***div*** field of the following structure:

```
typedef struct {
    Pint      div;        /* number of lat. and long. divisions */
} Pgse_cyl_div_data;
```

Stereo View Indices

The Stereo View Indices **GSE** provides a more general version of the PHIGS **SET_VIEW_INDICES** structure element, and can be used in place of the **SET_VIEW_INDICES** structure element. This element stores three view table indices: a ***mono*** view index that will be used by the structure walker if the structure is being displayed on a regular monoscopic screen; and ***left*** and ***right*** view indices that are used by the structure walker if the structure is being displayed on a stereo screen.

```
typedef struct {
    Pint      mono;
    Pint      left;
    Pint      right;
} Pgse_stereo_view_indices;
```

Polylines Over Fillarea Tolerance

This **GSE** is used to make polylines appear in front of fillareas when they may lie on the same plane as the fillarea. All polylines and fillarea edges are moved slightly forward relative to the fillarea interior. This **GSE** defines how far forward they are moved. By setting a ***tolerance*** of 0.0, applications forbid polylines from being moved. The tolerance value should be a value between -1.0 and 1.0. Lines are moved in NPC space in front of fillareas. The default is .003.

```
typedef struct {
    Pfloat    tolerance;
} Pgse_fillarea_tolerance;
```

Fillarea Front/Back Face Distinguish

To avoid ambiguity where the front and back fillareas of a surface meet (that is, along the silhouette), back fillareas are moved slightly backwards relative to front fillareas. This **GSE** defines how far back to move back fillareas. The ***distinguish*** field below is an integer that is subtracted to the back face z values before sending them to the z-buffer for testing. A value of 2 (the default) is typically sufficient for all back faces. A value of 0 disables any distinguishing between front and back fillareas.

```
typedef struct {
    Pint        distinguish;
} Pgse_front_back_distinguish;
```

The polyline over fillarea ***tolerance*** is a float while the front/back face ***distinguish*** is an integer because ***tolerance*** needs to vary depending on the model and may be quite large, while a small constant value is sufficient for ***distinguish***.

Polyline Quality

The Polyline Quality **GSE** provides a structure element that will enable or disable the anti-aliasing of polylines. If the **PSMOOTH** identifier is used in the ***flag*** field of the following structure, polylines will be anti-aliased. If the **PJAGGY** identifier is used, polylines will not be anti-aliased.

```
typedef enum {
    PSMOOTH,
    PJAGGY
} Pgse_polyline_quality_flag;

typedef struct {
    Pgse_polyline_quality_flag      flag;
} Pgse_polyline_quality;
```

Line Pattern Mask

This **GSE** sets the polyline type to a pattern other than the predefined polyline types. If the polyline type set by a call to the **SET_LINETYPE** function is set to **PES_PLINE_MASK** (defined in **esgdp.h**) the line pattern is taken as defined by this **GSE**. The ***length*** field of the following structure is the number of bits in the pattern, from 1 to 32. Each bit in the pattern represents one pixel: if a bit is set to 1 it causes the corresponding pixel to be drawn, if the bit is 0 drawing is suppressed. The bits in the pattern begin with the least significant bit.

```
typedef struct {
    Pint        length;        /* must be from 1 to 32 */
    Plong       pattern;
} Pgse_linepattern_mask;
```

Edge Pattern Mask

This **GSE** sets the fillarea edge type to a pattern other than the predefined edge types. If the edge type set by a call to the **SET_EDGETYPE** function is set to **PES_PLINE_MASK** (defined in **esgdp.h**) the edge pattern is taken as defined by this **GSE**.The ***length*** field of the following structure is the number of bits in the pattern, from 1 to 32. Each bit in the pattern represents one pixel: if a bit is set to 1 it causes the corresponding pixel to be drawn, if the bit is 0 drawing is suppressed. The bits in the pattern begin with the least significant bit.

```
typedef struct {
    Pint        length;    /* must be from 1 to 32 */
    Plong       pattern;
} Pgse_edgepattern_mask;
```

Traversal Information

The Traversal Information **GSE** provides a special structure element that will only be looked at by the structure walker if a special information traversal has been requested (see the X extension routine **XGetTraversalInfo**). When this **GSE** is traversed, the structure walker buffers either a matrix which is the composite local and global matrices, or a user defined ID. If the ***type*** field in the following structure is set to **PES_INFORMATION_MATRIX**, then the composite matrix is buffered and returned to the application. If the ***type*** field is set to **PES_INFORMATION_ID**, then the integer placed in the ***id*** field below is buffered and returned to the application. This functionality has been provided to aid molecular modeling applications in doing distance monitoring and energy calculations. This functionality could also be used in a number of other settings, such as collision detection.

```
typedef enum {
    PES_INFORMATION_MATRIX,
    PES_INFORMATION_ID
} Pes_information_type;

typedef struct {
    Pes_information_type  type;
     union {
    Pint        unused;
    Pint        id;
     } rec;
} Pgse_information_data;
```

Transparency

This **GSE** sets the ***transparent*** attribute for surfaces. The attribute is a floating point number from 0.0 to 1.0. A value of 1.0 is the most transparent (almost clear). A value of 0.0 turns the transparency functionality off. The ***data*** field of the **Pdata** structure should point to the following structure:

```
typedef struct _Pgse_transparency {
    Pfloat      transparent; /* Ranges from 0.0 to 1.0 */
} Pgse_transparency;
```

Errors

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

SET HLHSR ID

Name

SET HLHSR IDENTIFIER - create a structure element to set the current hidden line/hidden surface removal attribute.

Syntax

void pset_hlhsr_id (Id)

Pint Id; HLHSR identifier

Required PHIGS Operating States

(PHOP, *, STOP, *)

Description

SET HLHSR IDENTIFIER creates a structure element containing a value for the Hidden Line and Hidden Surface Removal (HLHSR) identifier attribute. During traversal, this attribute replaces the current HLHSR identifier and is applied to all output primitives that follow in the structure network in a workstation-dependent way. On the ESV workstation, it can turn *z*-buffering on and off if **PHIGS_HLHSR_MODE_ZBUFF** is being used. Also, it can be used to give special rendering instructions to the graphics processor for drawing surfaces that are transparent or lines that lie on polygons.

The HLHSR identifier in the structure network is used in conjunction with the HLHSR mode on the workstation during traversal. Presently, both must be on to enable *z*-buffering Hidden Surface Removal.

If the current edit mode is insert, then a **SET HLHSR IDENTIFIER** element is inserted into the currently open structure after the element pointed to by the current element pointer. If the edit mode is replace, then the new **SET HLHSR IDENTIFIER** element replaces the element pointed to by the element pointer. In either case, the element pointer is updated to point to the new element.

Input Parameter

id The HLHSR identifier value. Presently supported values are:

PHIGS_HLHSR_ID_OFF	Turn off z-buffering.
PHIGS_HLHSR_ID_ON	Turn on z-buffering.
PES_HLHSR_ID_BEG_SURFACES	Begin rendering surfaces.
PES_HLHSR_ID_BEG_SURF_EDGES	Begin rendering surface edges.
PES_HLHSR_ID_BEG_CTHRU	Begin rendering transparent objects.
PES_HLHSR_ID_BEG_LINES	Begin rendering lines.

Execution

When the **SET HLHSR IDENTIFIER** element is traversed, the current HLHSR identifier entry in the traversal state list is set to the HLHSR identifier that is stored in this element. The current HLHSR identifier is applied to output primitives that follow in the structure network.

On the ESV workstation, if the current HLHSR mode is **PHIGS_HLHSR_MODE_ZBUFF**, then the HLHSR identifier will turn z-buffering on or off.

The default state for the HLHSR identifier is **PHIGS_HLHSR_ID_OFF**.

Special ESV HLHSR IDs

On the ESV workstation, some images that include the use of transparencies, edges on surfaces such as fillareas, or polylines that pass through or are coincident with surfaces, can exhibit rendering problems. These problems are a natural result of trying to anti-alias or blend edges or transparent surfaces with the objects that lie behind them in a scene. In order to be done correctly, anti-aliasing or blending must be done after all underlying objects have been drawn. Since the ESV does not pre-sort all objects on a pixel basis before drawing, problems can appear. These rendering problems can be decreased by traversing and rendering primitives in certain groups and in an order that provides the best image. The groups are

- Opaque surfaces.
- Edges of opaque surfaces.
- Transparent surfaces and their edges.
- Polylines.

The above ordering may or may not be the best for a given image, depending on which primitives are in front and which are in back.

Note: Transparent surfaces are created by using a **GSE** element **PES_GSE_TRANSPARENCY** in the structure. It specifies a transparency value attribute between 0.0 and 1.0. If the transparency value is 0.0, transparency is turned off. If the transparency value is not 0.0, then it is turned on and surface primitives that follow in the structure are considered transparent. See the documentation on **GENERALIZED_STRUCTURE_ELEMENT**.

Two methods of grouping primitives are provided for rendering. Both have some speed degradation over the normal rendering method. The first is a "vanilla-PHIGS application" method, where we provide special HLHSR modes (see the manual page on HLHSR modes, not to be confused with HLHSR IDs!) that instruct the structure walker to do multiple traversals in order to send primitives to the graphics pipeline in the right order. The second is a "smart application" method, where the application intelligently orders primitives so that they are traversed in the correct order.

In the "smart application" method, the application must order the structures and the primitives in each structure so that all opaque fillareas, triangle_strips, *etc.* are traversed together. Also, transparent surfaces should be grouped together and polylines should be grouped together. The order of traversal depends on how the different groups relate to each other in terms of *z* (*i.e.*, which is in front and which in back). No single order may solve all problems. You will want to find an order that best fits your application. Generally, objects that are further away should be traversed first.

In addition, in front of each of these groups of primitives the "smart application" must place one of the following special HLHSR ID elements:

PES_HLHSR_ID_BEG_SURFACES	Begin rendering surfaces.
PES_HLHSR_ID_BEG_SURF_EDGES	Begin rendering edges of surfaces.
PES_HLHSR_ID_BEG_CTHRU	Begin rendering transparent surfaces.
PES_HLHSR_ID_BEG_LINES	Begin rendering lines.

These HLHSR IDs tell the structure walker and the graphics pipeline what kinds of primitives are coming next and how to deal with them. This "smart application" method of doing better renderings gets better performance at run time than the "vanilla-PHIGS" method alluded to above, but it requires intelligent structure building by the application.

Edges

Notice that there are different HLHSR IDs for doing edges of surfaces. In order to do surface edges with the "smart application" method, the application must create the structure so that surface primitives are traversed twice, the first time to draw the interior of the surface, the second time to draw the surface edges.

Note: The HLHSR mode for a workstation is set with the **SET HLHSR MODE** function.

SET HLHSR MODE

Name

SET HLHSR MODE - set the hidden line and hidden surface removal algorithm for a workstation

Syntax

void pset_hlhsr_mode (*ws, mode*)

Pint	***ws*;**	workstation identifier
Pint	***mode*;**	HLHSR mode

Required PHIGS Operating States

(PHOP, WSOP, *, *)

Description

The **SET HLHSR MODE** requests a certain Hidden Line and Hidden Surface Removal (HLHSR) mode for a workstation. The workstation's current HLHSR mode either sets the HLHSR algorithm to be used or it disables all HLHSR methods for a workstation. The current HLHSR identifier from the structure network is used in conjunction with the HLHSR mode on the workstation during traversal.

On the ESV workstation, the HLHSR mode enables the use of *z*-buffering, or causes special structure traversals to be used to create higher quality images:

CPK Renderings

This special mode turns on the high-quality, anti-aliased rendering of spheres and cylinders. This mode was developed to support molecular modeling applications. These renderings are done in software on the host and are not real time.

Multi-Pass Traversals

On the ESV workstation, some images that include the use of transparencies, anti-aliased edges on surfaces such as fillareas, or polylines that pass through or are coincident with surfaces, can exhibit rendering problems. These problems are a natural result of trying to blend edges or transparent surfaces with the objects that lie behind them in a scene. In order to be done correctly, anti-aliasing or blending must be done after all underlying objects have been drawn. Since the ESV does not pre-sort all objects on a pixel basis before drawing, problems can appear. These rendering problems can be decreased by traversing and rendering primitives in certain groups and in an order that provides the best image. The groups are

- Opaque surfaces.
- Edges of opaque surfaces.
- Transparent surfaces and their edges.
- Polylines.

The above ordering may or may not be the best for a given image, depending on which primitives are in front and which are in back. We have provided two methods of grouping primitives for rendering. Both have some speed degradation over the normal z-buffering HLHSR mode. The first is a "vanilla-PHIGS application" method, where we provide special HLHSR modes (see below) that instruct the structure walker to do multiple traversals in order to send primitives to the graphics pipeline in the desired order. The second is a "smart application" method, where the application intelligently orders primitives in structures so that they are traversed in the desired order.

Input Parameters

ws The identifier of the workstation whose HLHSR mode is being set.

mode The HLHSR mode value. Presently defined values are

PHIGS_HLHSR_MODE_NONE

Disable z-buffering. All HLHSR rendering algorithms will be disabled with this mode.

PHIGS_HLHSR_MODE_ZBUFF

Enable z-buffering. z-buffering is enabled and capable of being turned on by the correct HLHSR ID.

PES_HLHSR_MODE_HQ_CPK

High-Quality Sphere and Cylinder Rendering. This mode turns on high-quality, anti-aliased rendering of spheres and cylinders, and is important to the molecular modeling industry. The speed of this type of rendering is much slower than the regular HLHSR z-buffering mode. This mode has some restrictions. First, the only primitives that will be rendered under this mode are spheres and cylinders (see **GDP** primitives). Second, only orthographic viewing projections can be used, no perspective allowed! Third, the view table index can only be set once, at the top of a structure, and cannot be changed. Fourth, primitives rendered in this way cannot be picked.

PES_HLHSR_MODE_MULTIPASS_*xxx*

Rendering Correct Edges on Surfaces and Rendering Transparencies. This is a group of modes that instruct the structure walker to do multiple traversals in a particular order.There can be up to three traversals requested, one for doing surfaces, one for transparent surfaces, and one for polylines. The letters ***xxx*** represent various combinations of the letters "S" (for traversing surfaces), "T" (for traversing transparent surfaces), and "L" (for traversing polylines). While none of these modes are correct for all applications, the picture will be more correct if objects that are further away in *z* are rendered first so that anti-aliased primitives that are nearer in *z* can be anti-aliased to the color of the primitives that are behind them. Also, transparent objects should take part of their color from the objects that lie behind them and are better rendered after the objects that are further away. Because multiple traversals are done, this rendering method takes longer than the normal *z*-buffering mode (although hardware *z*-buffering is still turned on for these modes). While all possible combinations of traversals do not make sense and are not available, many are valid and are be listed below.

Note: Transparent surfaces are created by using a **GSE** element **PES_GSE_TRANSPARENCY** in the structure. It specifies a transparency value attribute between 0.0 and 1.0. If the transparency value is 0.0, transparency is turned off. If the transparency value is not 0.0, then it is turned on and surface primitives that follow in the structure are considered transparent. See the documentation for **GENERALIZED STRUCTURE ELEMENT.**

PES_HLHSR_MODE_MULTIPASS_ST

Rendering Surfaces and Transparent Objects Only. This mode causes the structure walker in the X Server to make two traversals, rendering opaque surfaces and their edges first, and transparent surfaces last. A traversal pass to render polylines is not done and they are left out of the picture.

PES_HLHSR_MODE_MULTIPASS_STL

Rendering Transparent Objects With Surfaces and Lines. This mode causes the structure walker in the X Server to make three traversals of the workstation, rendering opaque primitives such as fillareas and their edges first, transparent objects second, and polylines last.

PES_HLHSR_MODE_MULTIPASS_SL

Rendering Correct Edges On Fillareas And Polylines Over Fillareas. This mode causes the structure walker in the X Server to make two traversals of the structures posted to the workstation, rendering opaque surfaces first with their edges and polylines last. This makes possible the correct rendering of edges on fillareas and the best rendering of anti-aliased lines over fillareas. In this mode, surfaces that are transparent are not rendered.

PES_HLHSR_MODE_MULTIPASS_SLT

Rendering Transparent Objects With Surfaces and Lines. This mode causes the structure walker in the X Server to make three traversals of the workstation, rendering opaque surfaces and their edges first, polylines second, and transparent objects last. This mode should be used in preference to the **STL** mode above if most polylines and opaque surfaces in the picture are further away in *z* than transparent objects.

PES_HLHSR_MODE_MULTIPASS_T

Rendering Transparent Objects Only. This mode is very similar to the above modes, except that the traversal passes to render opaque surfaces and polylines are not done and they are left out of the picture. Transparent surfaces only are rendered.

PES_HLHSR_MODE_MULTIPASS_TS

Rendering Surfaces and Transparent Objects Only. This mode causes the structure walker to make two traversals, rendering transparent surfaces first with their edges and opaque surfaces second with their edges. The traversal pass to render polylines is not done and polylines are left out of the picture.

PES_HLHSR_MODE_MULTIPASS_TSL

Rendering Transparent Objects With Surfaces and Lines. This mode causes the structure walker in the X Server to make three traversals of the workstation, rendering transparent surfaces first, opaque surfaces second, and polylines last. This mode should be used in preference to the **STL** mode above if all transparent surfaces in the picture are further away in *z* than other objects.

PES_HLHSR_MODE_MULTIPASS_TL

Rendering Transparent Objects and Polylines Only. This mode is very similar to the above modes, except that the traversal pass to render non-transparent surfaces is not done and they are left out of the picture. Transparent surfaces are rendered first and opaque surfaces are rendered second.

PES_HLHSR_MODE_MULTIPASS_TLS

Rendering Transparent Objects With Surfaces and Lines. This mode causes the structure walker in the X Server to make three traversals of the workstation, rendering transparent surfaces first, polylines second, and opaque surfaces such as fillareas and their edges last. This mode should be used in preference to the **STL** mode above if all transparent surfaces in the picture are further away in *z* than polylines and other objects.

PES_HLHSR_MODE_MULTIPASS_LS

Rendering Opaque Objects and Polylines Only. This mode causes the structure walker to make two traversals of the workstation, rendering polylines first and opaque surfaces second. The pass to render transparent surfaces is not done and they are left out of the picture. This mode should be used in preference to the **TL** mode above if all polylines in the picture are further away in *z* than opaque objects.

PES_HLHSR_MODE_MULTIPASS_LST

Rendering Transparent Objects With Surfaces and Lines. This mode causes the structure walker in the X Server to make three traversals of the workstation, rendering polylines first, opaque surfaces second, and transparent surfaces last. This mode should be used in preference to other three-pass modes if polylines are generally furthest away in *z*, followed by opaque surfaces, and then by transparent ones.

PES_HLHSR_MODE_MULTIPASS_LT

Rendering Transparent Objects and Polylines Only. This mode causes polylines to be traversed first, followed by transparent surfaces. The pass to render opaque surfaces is not done and they are left out of the picture. This mode should be used in preference to the **TL** mode above if all polylines in the picture are further away in *z* than transparent objects.

PES_HLHSR_MODE_MULTIPASS_LTS

Rendering Transparent Objects With Surfaces and Lines. This mode causes the structure walker in the X Server to make three traversals of the workstation, rendering polylines first, transparent surfaces second, and opaque surfaces last. This mode should be used in preference to other three-pass modes if polylines are generally furthest away in *z*, followed by transparent surfaces, and then by opaque surfaces.

Note

To avoid the cost of multiple traversals when drawing fillarea edges, polylines over fillareas and transparent objects, the application can either use the standard *z*-buffering mode (and expect an occasional anomaly in the image), or it can use a "smart application" method where the structure is created in such a way that filled surfaces are traversed together, as are transparent objects and polylines. The application must then insert special HLHSR ID elements in the structure that provide special instructions to the structure walker.

This will cause edges, transparent objects and lines to be rendered more successfully in a single traversal under the usual **PHIGS_HLHSR_MODE_ZBUFF** mode. See **SET HLHSR IDENTIFIER** for details.

Execution

If the requested HLHSR mode value is supported on the specified workstation, then **SET HLHSR MODE** immediately sets the requested HLHSR mode entry in the PHIGS workstation state list to the specified mode. The effect of calling **SET HLHSR MODE** is not visible until the requested HLHSR mode replaces the current HLHSR mode. The time at which this occurs depends on the workstation's display update state.

This assignment is performed immediately and the HLHSR update state is set to Not Pending if any one of the following is true:

1) The workstation display update state allows update.

2) The workstation modification mode is any value other than No Immediate Visual Effect, and the dynamic modification accepted for HLHSR mode entry in the workstation description table is set to Immediate.

3) The display space empty status in the workstation state list is **EMPTY**.

Otherwise, the HLHSR update state is set to Pending and the requested HLHSR mode will not replace the current HLHSR mode until the next time the workstation is updated. The HLHSR update state will be set to Not Pending at that time.

PHIGS Input with ESV Devices

This section describes how to use PHIGS input to access the knob box (control dials), button box and Spaceball devices on an ESV workstation. This is not intended to be a PHIGS input tutorial.

Refer to the manual pages for **INITIALIZE CHOICE (3)**, **INITIALIZE LOCATOR (3)**, **INITIALIZE PICK (3)**, **INITIALIZE STRING (3)**, **INITIALIZE STROKE (3)**, and **INITIALIZE VALUATOR (3)** for a complete description of all the available PHIGS devices.

Initializing the Knob Box

The ESV knob box is a set of eight PHIGS valuator devices, numbers 11-18. The knobs are numbered from left to right and top to bottom. The ESV knob box implementation supports two PETs (Prompt and Echo Types):

- **PET 1** provides an echo of the current knob value on the label above the knob when the device is active. This value is a floating point number in the range specified by the **high** and **low** values in the **Pval_data** structure.
- **PET -1** allows the application to define the knob label in the **INITIALIZE VALUATOR** and **INITIALIZE VALUATOR 3** functions. This label will be displayed when the device is active. The **pets.pet_u1.label** field of the **Pval_data** structure is used to specify the label. The other fields of the structure are ignored by this device.

Initializing the Button Box

The ESV button box is a single PHIGS choice device number 3. The values returned range from 1-32. The buttons are numbered left to right and top to bottom. The ESV button box implementation supports three PETs:

- **PET 1** will echo the device by turning on the LED in the button when the button is pressed and turning off the LED when the button is released.
- **PET 2** will echo the device by turning on the button LEDs corresponding to the **PPR_ON** values specified in the prompts list in the **Pchoice_data** structure when the device is active. The prompts list can be from 1 to 32 in length. Values in the list beyond the 32nd will be ignored.
- **PET -1** will echo the device by turning on the button LED the first time a button is pressed and turning of the LED the next time the same button is pressed when the device is active. There is no data record associated with this PET.

Initializing Spaceball

The ESV Spaceball is a single PHIGS choice device number 2 for Spaceball buttons and a single PHIGS string device number 2 for Spaceball motion. The string is an encoded form of the six floating point numbers returned by Spaceball. These values can be decoded with a **sscanf** function call using the format **"%d,%d,%d,%d,%d,%d"** and six floating point variables to receive the values.

The ESV Spaceball implementation supports only PET for both the choice and string device:

- **PET 1** has no echo on either the choice or string device.

Input Processing

There are no special considerations for processing input from ESV devices. They all follow standard PHIGS input conventions with the exception that the ESV Spaceball string device returns six floating point values encoded as a string.

Knob Box Example

```
{
   Plimit       echo_area;
   Pval_data    val_data;
   float        delay = 2.0;
   Pint         dev, ws;
   Pin_class    class;
   Pfloat       val;
   int          done = 0;

   /* init echo area. This is required even thought it is ignored */
   echo_area.x_min = 0.0;
   echo_area.x_max = 1.0;
   echo_area.y_min = 0.0;
   echo_area.y_max = 1.0;

   /* initialize knobs 1,2 and 3 with PET -1 and labels */
   val_data.low = -1.0;
   val_data.high = 1.0;
   val_data.pets.pet_u1.format = NULL;
   val_data.pets.pet_u1.low_label = NULL;
   val_data.pets.pet_u1.high_label = NULL;
   val_data.pets.pet_u1.label = "Rotate X";
   pinit_val(1, 11, 0.0, -1, &echo_area, &val_data);
   val_data.pets.pet_u1.label = "Rotate y";
   pinit_val(1, 12, 0.0, -1, &echo_area, &val_data);
```

```
    val_data.pets.pet_u1.label = "Rotate z";
    pinit_val(1, 13, 0.0, -1, &echo_area, &val_data);

    /* set knobs 1, 2, and 3 in event mode */
    pset_val_mode( 1, 11, POP_EVENT, PSWITCH_ECHO);
    pset_val_mode( 1, 12, POP_EVENT, PSWITCH_ECHO);
    pset_val_mode( 1, 13, POP_EVENT, PSWITCH_ECHO);

    while(!done)
    {
       pawait_event( delay, &ws, &class, &dev );
       switch (class)
       {
         default:
         case PIN_NONE:
          break;
         case PIN_VAL:
          pget_val( &val );
          /* process knob events */
          break;
       }
    }
}
```

Button Box Example

```
{
    Pchoice_data cho_data;
    Plimit       echo_area;
    float        delay = 2.0;
    Pint         dev, ws;
    Pin_class    class;
    Pint         cho;
    Pin_status   status;
    int          done = 0;

    /* init echo area. This is required even thought it is ignored */
    echo_area.x_min = 0.0;
    echo_area.x_max = 1.0;
    echo_area.y_min = 0.0;
    echo_area.y_max = 1.0;
```

```
   /* initialize PET 2 with LEDs 1-9 on and the rest off */
   cho_data.pet_r2.num_prompts = 32;
   cho_data.pet_r2.prompts = (Ppr_switch *)calloc(32,
            sizeof(Ppr_switch));
   for (i = 0; i < 9; i++) {
      cho_data.pet_r2.prompts[i] = PPR_ON;
   }
   pinit_choice(1, 3, PIN_STATUS_OK, 1, 2, &echo_area, &cho_data);

   /* set buttons in event mode */
   pset_choice_mode( 1, 3, POP_EVENT, PSWITCH_ECHO);

   while (!done)
   {
      pawait_event( delay, &ws, &class, &dev );
      switch (class)
      {
        default:
        case PIN_NONE:
         break;
        case PIN_CHOICE:
         pget_choice( &status, &cho);
         switch (status)
         {
           case PIN_STATUS_OK:
            /* process button box events */
            break;
         }
         break;
      }
   }
}
```

Spaceball Example

```
{
   int          done = 0;
   float        delay = 2.0;
   Pint         dev, ws;
   Pin_class    class;
   char         str[100];
   Pint         cho;
   Pin_status   status;
```

```
   /* set space ball in event mode. it is both a chaice and string
    * device. There is no need to initialize
    */
   pset_choice_mode( 1, 2, POP_EVENT, PSWITCH_ECHO);
   pset_string_mode( 1, 2, POP_EVENT, PSWITCH_ECHO);

   while (!done)
   {
      pawait_event( delay, &ws, &class, &dev );
      switch (class)
      {
        default:
        case PIN_NONE:
         break;
        case PIN_STRING:
         {
            int     axis[6];

            pget_string(str);
            sscanf(str, "%d,%d,%d,%d,%d,%d",
            &axis[0],
            &axis[1],
            &axis[2],
            &axis[3],
            &axis[4],
            &axis[5]
            );
            /* process spaceball motion event */
            break;
         }
        case PIN_CHOICE:
         pget_choice( &status, &cho);
         switch (status)
         {
           case PIN_STATUS_OK:
            /* process spaceball button events */
            break;
         }
         break;
      }
   }
}
```

Function Numbers

The function numbers listed in the left-hand column are returned by error messages. The corresponding symbolic name is listed in the middle column, and the corresponding function name is listed in the right-hand column.

In the following list, functions shown in **bold** are supported by the ESV Workstation, and functions shown in *italics* are not currently supported by the ESV Workstation.

No.	Symbolic Name	Function Name
-5	**Pfn_openpex**	**OPEN PEX**
0	**Pfn_openphigs**	**OPEN PHIGS**
1	**Pfn_closephigs**	**CLOSE PHIGS**
2	**Pfn_openws**	**OPEN WORKSTATION**
3	**Pfn_closews**	**CLOSE WORKSTATION**
4	**Pfn_redrawallstruct**	**REDRAW ALL STRUCTURES**
5	**Pfn_updatews**	**UPDATE WORKSTATION**
6	**Pfn_setdisplayupdatest**	**SET DISPLAY UPDATE STATE**
7	**Pfn_message**	**MESSAGE**
8	**Pfn_polyline3**	**POLYLINE 3**
9	**Pfn_polyline**	**POLYLINE**
10	**Pfn_polymarker3**	**POLYMARKER 3**
11	**Pfn_polymarker**	**POLYMARKER**
12	**Pfn_text3**	**TEXT 3**
13	**Pfn_text**	**TEXT**
14	**Pfn_annotationtextrelative3**	**ANNOTATION TEXT RELATIVE 3**
15	**Pfn_annotationtextrelative**	**ANNOTATION TEXT RELATIVE**
16	**Pfn_fillarea3**	**FILL AREA 3**
17	**Pfn_fillarea**	**FILL AREA**
18	**Pfn_fillareaset3**	**FILL AREA SET 3**
19	**Pfn_fillareaset**	**FILL AREA SET**
20	*Pfn_cellarray3*	*CELL ARRAY 3*
21	*Pfn_cellarray*	*CELL ARRAY*
22	**Pfn_gdp3**	**GENERALIZED DRAWING PRIMITIVE 3**
23	**Pfn_gdp**	**GENERALIZED DRAWING PRIMITIVE**
24	**Pfn_setlineind**	**SET POLYLINE INDEX**
25	**Pfn_setmarkerind**	**SET POLYMARKER INDEX**

No.	Symbolic Name	Function Name
26	**Pfn_settextind**	**SET TEXT INDEX**
27	**Pfn_setintind**	**SET INTERIOR INDEX**
28	**Pfn_setedgeind**	**SET EDGE INDEX**
29	**Pfn_setlinetype**	**SET LINETYPE**
30	**Pfn_setlinewidth**	**SET LINEWIDTH SCALE FACTOR**
31	**Pfn_setlinecolourind**	**SET POLYLINE COLOUR INDEX**
32	**Pfn_setmarkertype**	**SET MARKER TYPE**
33	**Pfn_setmarkersize**	**SET MARKER SIZE SCALE FACTOR**
34	**Pfn_setmarkercolourind**	**SET POLYMARKER COLOUR INDEX**
35	**Pfn_settextfont**	**SET TEXT FONT**
36	**Pfn_settextprec**	**SET TEXT PRECISION**
37	**Pfn_setcharexpan**	**SET CHARACTER EXPANSION FACTOR**
38	**Pfn_setcharspace**	**SET CHARACTER SPACING**
39	**Pfn_settextcolourind**	**SET TEXT COLOUR INDEX**
40	**Pfn_setcharheight**	**SET CHARACTER HEIGHT**
41	**Pfn_setcharup**	**SET CHARACTER UP VECTOR**
42	**Pfn_settextpath**	**SET TEXT PATH**
43	**Pfn_settextalign**	**SET TEXT ALIGNMENT**
44	**Pfn_setannotationcharheight**	**SET ANNOTATION TEXT CHARACTER HEIGHT**
45	**Pfn_setannotationcharup**	**SET ANNOTATION TEXT CHARACTER UP VECTOR**
46	**Pfn_setannotationpath**	**SET ANNOTATION TEXT PATH**
47	**Pfn_setannotationalign**	**SET ANNOTATION TEXT ALIGNMENT**
48	**Pfn_setannotationstyle**	**SET ANNOTATION STYLE**
49	**Pfn_setintstyle**	**SET INTERIOR STYLE**
50	**Pfn_setintstyleind**	**SET INTERIOR STYLE INDEX**
51	**Pfn_setintcolourind**	**SET INTERIOR COLOUR INDEX**
51	**Pfn_setedgeflag**	**SET EDGE FLAG**
53	**Pfn_setedgetype**	**SET EDGETYPE**
54	**Pfn_setedgewidth**	**SET EDGEWIDTH SCALE FACTOR**
55	**Pfn_setedgecolourind**	**SET EDGE COLOUR INDEX**
56	*Pfn_setpatsize*	*SET PATTERN SIZE*

No.	Symbolic Name	Function Name
57	*Pfn_setpatrefptvectors*	*SET PATTERN REFERENCE POINT AND VECTORS*
58	*Pfn_setpatrefpt*	*SET PATTERN REFERENCE POINT*
59	**Pfn_addnameset**	**ADD NAMES TO SET**
60	**Pfn_removenameset**	**REMOVE NAMES FROM SET**
61	**Pfn_setindivasf**	**SET INDIVIDUAL ASF**
62	**Pfn_setlinerep**	**SET POLYLINE REPRESENTATION**
63	**Pfn_setmarkerrep**	**SET POLYMARKER REPRESENTATION**
64	**Pfn_settextrep**	**SET TEXT REPRESENTATION**
65	**Pfn_setintrep**	**SET INTERIOR REPRESENTATION**
66	**Pfn_setedgerep**	**SET EDGE REPRESENTATION**
67	*Pfn_setpatrep*	*SET PATTERN REPRESENTATION*
68	**Pfn_setcolourrep**	**SET COLOUR REPRESENTATION**
69	**Pfn_sethilightfilter**	**SET HIGHLIGHTING FILTER**
70	**Pfn_setinvisfilter**	**SET INVISIBILITY FILTER**
71	**Pfn_setcolourmodel**	**SET COLOUR MODEL**
72	**Pfn_sethlhsrid**	**SET HLHSR IDENTIFIER**
73	**Pfn_sethlhsrmode**	**SET HLHSR MODE**
74	**Pfn_setlocaltran3**	**SET LOCAL TRANSFORMATION 3**
75	**Pfn_setlocaltran**	**SET LOCAL TRANSFORMATION**
76	**Pfn_setglobaltran3**	**SET GLOBAL TRANSFORMATION 3**
77	**Pfn_setglobaltran**	**SET GLOBAL TRANSFORMATION**
78	*Pfn_setmodelclipvolume3*	*SET MODELLING CLIPPING VOLUME 3*
79	*Pfn_setmodelclipvolume*	*SET MODELLING CLIPPING VOLUME*
80	*Pfn_setmodelclipindicator*	*SET MODELLING CLIPPING INDICATOR*
81	*Pfn_restoremodelclipvolume*	*RESTORE MODELLING CLIPPING VOLUME*
82	**Pfn_setviewind**	**SET VIEW INDEX**
83	**Pfn_setviewrep3**	**SET VIEW REPRESENTATION 3**
84	**Pfn_setviewrep**	**SET VIEW REPRESENTATION**
85	**Pfn_setviewtraninputpri**	**SET VIEW TRANSFORMATION INPUT PRIORITY**
86	**Pfn_setwswindow3**	**SET WORKSTATION WINDOW 3**
87	**Pfn_setwswindow**	**SET WORKSTATION WINDOW**

No.	Symbolic Name	Function Name
88	Pfn_setwsviewport3	SET WORKSTATION VIEWPORT 3
89	Pfn_setwsviewport	SET WORKSTATION VIEWPORT
90	Pfn_openstruct	OPEN STRUCTURE
91	Pfn_closestruct	CLOSE STRUCTURE
92	Pfn_executestruct	EXECUTE STRUCTURE
93	Pfn_label	LABEL
94	Pfn_applicationdata	APPLICATION DATA
95	Pfn_gse	GENERALIZED STRUCTURE ELEMENT
96	Pfn_seteditmode	SET EDIT MODE
97	Pfn_copyallelemsstruct	COPY ALL ELEMENTS FROM STRUCTURE
98	Pfn_setelemptr	SET ELEMENT POINTER
99	Pfn_offsetelemptr	OFFSET ELEMENT POINTER
100	Pfn_setelemptrlabel	SET ELEMENT POINTER AT LABEL
101	Pfn_delelem	DELETE ELEMENT
102	Pfn_delelemrange	DELETE ELEMENT RANGE
103	Pfn_delelemslabels	DELETE ELEMENTS BETWEEN LABELS
104	Pfn_emptystruct	EMPTY STRUCTURE
105	Pfn_delstruct	DELETE STRUCTURE
106	Pfn_delstructnet	DELETE STRUCTURE NETWORK
107	Pfn_delallstruct	DELETE ALL STRUCTURES
108	Pfn_changestructid	CHANGE STRUCTURE IDENTIFIER
109	Pfn_changestructref	CHANGE STRUCTURE REFERENCES
110	Pfn_changestructidref	CHANGE STRUCTURE IDENTIFIER AND REFERENCES
111	Pfn_poststruct	POST STRUCTURE
112	Pfn_unpoststruct	UNPOST STRUCTURE
113	Pfn_unpostallstruct	UNPOST ALL STRUCTURES
114	Pfn_openarfile	OPEN ARCHIVE FILE
115	Pfn_closearfile	CLOSE ARCHIVE FILE
116	Pfn_arstruct	ARCHIVE STRUCTURES
117	Pfn_arstructnet	ARCHIVE STRUCTURE NETWORKS
118	Pfn_arallstruct	ARCHIVE ALL STRUCTURES

No.	Symbolic Name	Function Name
119	**Pfn_setconfres**	**SET CONFLICT RESOLUTION**
120	**Pfn_retrievestructids**	**RETRIEVE STRUCTURE IDENTIFIERS**
121	**Pfn_retrievestruct**	**RETRIEVE STRUCTURES**
122	**Pfn_retrievestructnet**	**RETRIEVE STRUCTURE NETWORKS**
123	**Pfn_retrieveallstruct**	**RETRIEVE ALL STRUCTURES**
124	**Pfn_retrieveancesstruct**	**RETRIEVE ANCESTORS OF STRUCTURE**
125	**Pfn_retrievedescstruct**	**RETRIEVE DESCENDANTS OF STRUCTURE**
126	**Pfn_delstructar**	**DELETE STRUCTURES FROM ARCHIVE**
127	**Pfn_delstructnetar**	**DELETE STRUCTURE NETWORKS FROM ARCHIVE**
128	**Pfn_delallstructar**	**DELETE ALL STRUCTURES FROM ARCHIVE**
129	**Pfn_setpickid**	**SET PICK IDENTIFIER**
130	**Pfn_setpickfilter**	**SET PICK FILTER**
131	**Pfn_initloc3**	**INITIALIZE LOCATOR 3**
132	**Pfn_initloc**	**INITIALIZE LOCATOR**
133	*Pfn_initstroke3*	*INITIALIZE STROKE 3*
134	*Pfn_initstroke*	*INITIALIZE STROKE*
135	**Pfn_initval3**	**INITIALIZE VALUATOR 3**
136	**Pfn_initval**	**INITIALIZE VALUATOR**
137	**Pfn_initchoice3**	**INITIALIZE CHOICE 3**
138	**Pfn_initchoice**	**INITIALIZE CHOICE**
139	**Pfn_initpick3**	**INITIALIZE PICK 3**
140	**Pfn_initpick**	**INITIALIZE PICK**
141	**Pfn_initstring3**	**INITIALIZE STRING 3**
142	**Pfn_initstring**	**INITIALIZE STRING**
143	**Pfn_setlocmode**	**SET LOCATOR MODE**
144	*Pfn_setstrokemode*	*SET STROKE MODE*
145	**Pfn_setvalmode**	**SET VALUATOR MODE**
146	**Pfn_setchoicemode**	**SET CHOICE MODE**
147	**Pfn_setpickmode**	**SET PICK MODE**
148	**Pfn_setstringmode**	**SET STRING MODE**

No.	Symbolic Name	Function Name
149	**Pfn_reqloc3**	**REQUEST LOCATOR 3**
150	**Pfn_reqloc**	**REQUEST LOCATOR**
151	*Pfn_reqstroke3*	*REQUEST STROKE 3*
152	*Pfn_reqstroke*	*REQUEST STROKE*
153	**Pfn_reqval**	**REQUEST VALUATOR**
154	**Pfn_reqchoice**	**REQUEST CHOICE**
155	**Pfn_reqpick**	**REQUEST PICK**
156	**Pfn_reqstring**	**REQUEST STRING**
157	**Pfn_sampleloc3**	**SAMPLE LOCATOR 3**
158	**Pfn_sampleloc**	**SAMPLE LOCATOR**
159	*Pfn_samplestroke3*	*SAMPLE STROKE 3*
160	*Pfn_samplestroke*	*SAMPLE STROKE*
161	**Pfn_sampleval**	**SAMPLE VALUATOR**
162	**Pfn_samplechoice**	**SAMPLE CHOICE**
163	**Pfn_samplepick**	**SAMPLE PICK**
164	**Pfn_samplestring**	**SAMPLE STRING**
165	**Pfn_awaitevent**	**AWAIT EVENT**
166	**Pfn_flushevents**	**FLUSH DEVICE EVENTS**
167	**Pfn_getloc3**	**GET LOCATOR 3**
168	**Pfn_getloc**	**GET LOCATOR**
169	*Pfn_getstroke3*	*GET STROKE 3*
170	*Pfn_getstroke*	*GET STROKE*
171	**Pfn_getval**	**GET VALUATOR**
172	**Pfn_getchoice**	**GET CHOICE**
173	**Pfn_getpick**	**GET PICK**
174	**Pfn_getstring**	**GET STRING**
175	*Pfn_writemf*	*WRITE ITEM TO METAFILE*
176	*Pfn_gettypemf*	*GET ITEM TYPE FROM METAFILE*
177	*Pfn_readmf*	*READ ITEM FROM METAFILE*
178	*Pfn_interpret*	*INTERPRET ITEM*
179	**Pfn_seterrorhandmode**	**SET ERROR HANDLING MODE**
180	**Pfn_escape**	**ESCAPE**
201	**Pfn_polylineset3data**	**POLYLINE SET 3 WITH DATA**

No.	Symbolic Name	Function Name
202	**Pfn_fillarea3data**	**FILL AREA 3 WITH DATA**
203	**Pfn_fillareaset3data**	**FILL AREA SET 3 WITH DATA**
205	**Pfn_tri3data**	**TRIANGLE STRIP 3 WITH DATA**
206	**Pfn_quad3data**	**QUADRILATERAL MESH 3 WITH DATA**
207	**Pfn_polyhedron3data**	**POLYHEDRON 3 WITH DATA**
208	*Pfn_nunibspcurv*	*NON-UNIFORM B-SPLINE CURVE*
210	*Pfn_nunibspsurf*	*NON-UNIFORM B-SPLINE SURFACE*
211	*Pfn_extcellarray3*	*EXTENDED CELL ARRAY 3*
212	**Pfn_compfillareasetgnorm**	**COMPUTE FILL AREA SET GEOMETRIC NORMAL**
213	**Pfn_setdcueind**	**SET DEPTH CUE INDEX**
216	**Pfn_setareaprop**	**SET AREA PROPERTIES**
217	**Pfn_setbackareaprop**	**SET BACK AREA PROPERTIES**
218	**Pfn_setlineshadmethod**	**SET POLYLINE SHADING METHOD**
220	**Pfn_setbackintstyle**	**SET BACK INTERIOR STYLE**
221	**Pfn_setbackintstyleind**	**SET BACK INTERIOR STYLE INDEX**
222	**Pfn_setintshadmethod**	**SET INTERIOR SHADING METHOD**
223	**Pfn_setbackintshadmethod**	**SET BACK INTERIOR SHADING METHOD**
224	**Pfn_setintreflecteq**	**SET INTERIOR REFLECTANCE EQUATION**
225	**Pfn_setbackintreflecteq**	**SET BACK INTERIOR REFLECTANCE EQUATION**
226	**Pfn_setlightsrcstate**	**SET LIGHT SOURCE STATE**
227	**Pfn_setfacedistgmode**	**SET FACE DISTINGUISHING MODE**
228	**Pfn_setfaceculImode**	**SET FACE CULLING MODE**
229	**Pfn_setlinecolour**	**SET POLYLINE COLOUR**
230	**Pfn_setmarkercolour**	**SET POLYMARKER COLOUR**
231	**Pfn_settextcolour**	**SET TEXT COLOUR**
232	**Pfn_setintcolour**	**SET INTERIOR COLOUR**
233	**Pfn_setbackintcolour**	**SET BACK INTERIOR COLOUR**
234	**Pfn_setedgecolour**	**SET EDGE COLOUR**
235	*Pfn_setcurveapprox*	*SET CURVE APPROXIMATION CRITERIA*

No.	Symbolic Name	Function Name
236	*Pfn_settrimcurvapprox*	*SET TRIMMING CURVE APPROXIMATION CRITERIA*
237	*Pfn_setsurfapprox*	*SET SURFACE APPROXIMATION CRITERIA*
239	**Pfn_setextlinerep**	**SET EXTENDED POLYLINE REPRESENTATION**
240	**Pfn_setextmarkerrep**	**SET EXTENDED POLYMARKER REPRESENTATION**
241	**Pfn_setexttextrep**	**SET EXTENDED TEXT REPRESENTATION**
242	**Pfn_setextedgerep**	**SET EXTENDED EDGE REPRESENTATION**
243	**Pfn_setgenintrep**	**SET EXTENDED INTERIOR REPRESENTATION**
245	*Pfn_setextpatrep*	*SET EXTENDED PATTERN REPRESENTATION*
246	**Pfn_setdcuerep**	**SET DEPTH CUE REPRESENTATION**
247	**Pfn_setlightsrcrep**	**SET LIGHT SOURCE REPRESENTATION**

Error Messages

The numbers listed in the left-hand column are returned by error messages. The corresponding symbolic name is listed in the middle column, and the corresponding error description is listed in the right-hand column.

Error	Symbolic Name	Description
-317	**PXBADIMPL**	X Bad Implementation Error.
-316	**PXBADLENGTH**	X Bad Length Error.
-315	**PXBADNAME**	X Bad Name Error.
-314	**PXBADIDCHOICE**	X Bad ID Choice Error.
-313	**PXBADGC**	X Bad GC Error.
-312	**PXBADCOLOR**	X Bad Colour Error.
-311	**PXBADALLOC**	X Bad Alloc Error.
-310	**PXBADACCESS**	X Bad Access Error.
-309	**PXBADDRAWABLE**	X Bad Drawable Error.
-308	**PXBADMATCH**	X Bad Match Error.
-307	**PXBADFONT**	X Bad Font Error.
-306	**PXBADCURSOR**	X Bad Cursor Error.
-305	**PXBADATOM**	X Bad Atom Error.
-304	**PXBADPIXMAP**	X Bad Pixmap Error.
-303	**PXBADWINDOW**	X Bad Window Error.
-302	**PXBADVALUE**	X Bad Value Error.
-301	**PXBADREQUEST**	X Bad Request Error.
-264	**PPEXOCE**	PEX output command error.
-263	**PPEXSE**	PEX structure error.
-262	**PPEXSCE**	PEX search context error.
-261	**PPEXRE**	PEX renderer error.
-260	**PPEXPCE**	PEX pipeline context error.
-259	**PPEXPME**	PEX pick measure error.
-258	**PPEXPWE**	PEX PHIGS workstation error.
-257	**PPEXFE**	PEX font error.
-256	**PPEXPE**	PEX path error.
-255	**PPEXNSE**	PEX name set error.
-254	**PPEXLTE**	PEX lookup table error.
-253	**PPEXLE**	PEX label error.

Error	Symbolic Name	Description
-252	**PPEXFPFE**	PEX floating point format error.
-251	**PPEXRSE**	PEX rendering state error.
-250	**PPEXCTE**	PEX colour type error.
-202	**PPXALLOC**	Ignoring function. An X allocation error has occurred.
-201	**PPEXNOPEX**	Ignoring function. The specified X Server does not support a compatible PEX extension.
-200	**PPEXNOXSRVR**	Ignoring function. Cannot connect to the designated or default server.
-167	**PEMAXCRWS**	Ignoring function. Opening this workstation would exceed the maximum number of simultaneously open canvas region workstations on a canvas.
-165	**PEBADNEDGE**	Ignoring function. The length of specified edge data lists is inconsistent with the length of corresponding vertices lists.
-164	**PEBADNVTX**	Ignoring function. The specified number of vertices or sets of vertices is less than 0.
-163	**PENOEFLAG**	Ignoring function. The specified edge flag is invalid.
-162	**PENOVFLAG**	Ignoring function. The specified vertex flag is invalid.
-161	**PENOFFLAG**	Ignoring function. The specified facet flag is invalid.
-160	**PENOFUNC**	Ignoring function. The specified function is not available on the specified workstation.
-159	**PERNOINFO**	Ignoring function. The requested information is not available.
-157	**PENOGDP**	Warning. The specified **GDP** is not available on one or more workstations in this implementation. The **GDP** will be processed by those workstations on which it is available.
-156	**PENOFONTCS**	Ignoring function. Specified font is not available for character set.
-155	**PEBADCHARSET**	Specified character set is invalid.
-153	**PELENGTHLZ**	List length is less than 0. 0 will be used.

Error	Symbolic Name	Description
-152	**PENOTIMPL**	Ignoring function. Not implemented.
-151	**PEBADNAME**	Ignoring function. Nameset or filter contains name outside supported range.
-150	**PEBADNPTS**	Ignoring function. The specified number of points or sets of points is less than 0.
-100	**PEWSTBOUND**	Ignoring function. Workstation type is a default type or bound to a workstation and cannot be modified.
-57	**PESHMEM**	Kernel not configured with shared-memory. IPC facility needed for PEX-SI communication.
-55	**PENOFONT**	Ignoring function. Cannot open PHIGS. Cannot open font files.
-54	**PENOSPFILE**	Ignoring function. Cannot locate SI support file.
-53	**PEBADFPATH**	Ignoring function. SI support file path invalid.
-52	**PEPATHTOOLONG**	Ignoring function. **PEXAPIDIR** path is too long.
-51	**PESRVRFILE**	Ignoring function. Cannot open PHIGS. Cannot locate SI file **phigsmon**.
-50	**PECOMM**	Communication error.
-6	**PENOTRAVMEM**	Could not allocate additional dynamic memory during structure traversal.
-2	PEEXEC	Ignoring function. Cannot open PHIGS. Cannot create server.
-1	PECOMCREAT	Ignoring function. Cannot open PHIGS. Cannot create communication channel.
0	**PNO_ERROR**	No error.
1	**PENOTCL**	Ignoring function. Function requires state (**PHCL,WSCL,STCL,ARCL**).
2	**PENOTPHOP**	Ignoring function. Function requires state (**PHOP,*,*,***).
3	**PENOTWSOP**	Ignoring function. Function requires state (**PHOP,WSOP,*,***).
4	**PENOTPHOPCL**	Ignoring function. Function requires state (**PHOP,WSCL,STCL,ARCL**).
5	**PENOTSTOP**	Ignoring function. Function requires state (**PHOP,*,STOP,***).

Error	Symbolic Name	Description
6	**PENOTSTCL**	Ignoring function. Function requires state (**PHOP**,*,**STCL**,*).
7	**PENOTAROP**	Ignoring function. Function requires state (**PHOP**,*,*,**AROP**).
50	**PECNIDINV**	Ignoring function. Connection identifier not recognized by the implementation.
51	**PENOTAVAIL**	Ignoring function. This information is not yet available for this generic workstation type. Open a workstation of this type and use the specific workstation type.
52	**PEWSTYPEINV**	Ignoring function. Workstation type not recognized by the implementation.
53	**PEWSIDINUSE**	Ignoring function. Workstation identifier already is in use.
54	**PEWSNOTOP**	Ignoring function. The specified workstation is not open.
55	**PENOWSOP**	Ignoring function. Workstation cannot be opened for an implementation dependent reason.
56	**PEWSNOTMO**	Ignoring function. Specified workstation is not of category **MO**.
57	**PEWSCATMI**	Ignoring function. Specified workstation is of category **MI**.
58	**PEWSNOTMI**	Ignoring function. Specified workstation is not of category **MI**.
59	**PEWSNOTOUT**	Ignoring function. The specified workstation does not have output capability (*i.e.*, the workstation category is neither **OUTPUT**, **OUTIN**, nor **MO**).
60	**PEWSNOTIN**	Ignoring function. Specified workstation is not of category **OUTIN**.
61	**PEWSNOTIO**	Ignoring function. Specified workstation is neither of category **INPUT** nor of category **OUTIN**.
62	**PEWSNOTOO**	Ignoring function. This information is not available for this **MO** workstation type.

Error	Symbolic Name	Description
63	**PEWSMAXOPN**	Ignoring function. Opening this workstation would exceed the maximum number of simultaneously open workstations.
64	**PEWSNOGDP**	Ignoring function. The specified workstation type is not able to generate the specified generalized drawing primitive.
100	**PEBINXLT1**	Ignoring function. The bundle index value is less than 1.
101	**PENOREP**	The specified representation has not been defined.
102	**PENOPREDEF**	Ignoring function. The specified representation has not be predefined on this workstation.
103	**PEWSMAXBNL**	Ignoring function. Setting this bundle table entry would exceed the maximum number of entries allowed in the workstation bundle table.
104	**PENOLINTP**	Ignoring function. The specified linetype is not available on the specified workstation.
105	**PENOMKRTP**	Ignoring function. The specified marker type is not available on the specified workstation.
106	**PENOTXTFP**	Ignoring function. The specified font is not available for the requested text precision on the specified workstation.
107	**PENOEDGTP**	Ignoring function. The specified edgetype is not available on the specified workstation.
108	**PENOISTYL**	Ignoring function. The specified interior style is not available on the workstation.
109	**PENOPAT**	Ignoring function. Interior style **PATTERN** is not supported on the workstation.
110	**PEBADCMOD**	Ignoring function. The specified colour model is not available on the workstation.
111	**PENOHLHSR**	Ignoring function. The specified **HLHSR** mode is not available on the specified workstation.
112	**PEPINXLT1**	Ignoring function. The pattern index value is less than 1.
113	**PECINXLZ**	Ignoring function. The colour index value is less than 0.
114	**PEBINDLZ**	Ignoring function. The view index value is less than 0.

Error	Symbolic Name	Description
115	**PEBINDL1**	Ignoring function. The view index value is less than 1.
116	**PEBADPAT**	Ignoring function. One of the dimensions of pattern colour array is less than 1.
117	**PECADIM**	Ignoring function. One of the dimensions of the colour index array is less than 0.
118	**PEBADCRNG**	Ignoring function. One of the components of the colour specification is out of range. The valid range is dependent upon the current colour mode.
119	**PEDCINDLZ**	Ignoring function. Depth cue index is less than 0.
120	**PEBADDCIND**	Ignoring function. Depth cue index is less than 1.
122	**PENOLINSHADE**	Ignoring function. The specified polyline shading method is not available on the workstation.
123	**PENOINTSHADE**	Ignoring function. The specified interior shading method is not available on the workstation.
124	**PENOREFEQN**	Ignoring function. The specified interior reflectance equation is not available on the workstation.
129	**PEBADLTSSRCIND**	Ignoring function. The light source index is less than 1.
130	**PEINVREFPL**	Ignoring function. Invalid reference planes. **DQMIN > DQMAX.**
131	**PENOLTSRCTYPE**	Ignoring function. The specified light source type is not available on the workstation.
132	**PEINVLTANG**	Ignoring function. The specified spot light spread angle is out of range.
133	**PEINVALLSSIND**	Ignoring function. One of the entries in the activation list or the deactivation list is less than 1.
135	**PEINVALLSS**	Ignoring function. The same entry exists in both the activation and the deactivation list.
150	**PEMAXVIEW**	Ignoring function. Setting this view table entry would exceed the maximum number of entries allowed in the workstation's view table.

Error	Symbolic Name	Description
151	**PEBADWIN**	Ignoring function. Invalid window. **XMIN** ≥ **XMAX**, **YMIN** ≥ **YMAX**, or **ZMIN** > **ZMAX**.
152	**PEBADVP**	Ignoring function. Invalid viewport. **XMIN** ≥ **XMAX**, **YMIN** ≥ **YMAX**, or **ZMIN** > **ZMAX**.
153	**PEBADBOX**	Ignoring function. Invalid view clipping limits. **XMIN** ≥ **XMAX**, **YMIN** ≥ **YMAX**, or **ZMIN** > **ZMAX**.
154	**PEBADVLIM**	Ignoring function. The view clipping limits are not within NPC range.
155	**PEBADPROVP**	Ignoring function. The projection viewport limits are not within NPC range.
156	**PEWINRNG**	Ignoring function. The workstation window limits are not within NPC range.
157	**PEVPRNG**	Ignoring function. The workstation viewport is not within display space.
158	**PEFREQBK**	Ignoring function. Front plane and back plane distances are equal when *z*-extent of the projection viewport is 0.
159	**PEBADVPN**	Ignoring function. The view plane normal vector has length 0.
160	**PEBADVUP**	Ignoring function. The view up vector has length 0.
161	**PEBADVIEW**	Ignoring function. The view up and view plane normal vectors are parallel thus the viewing coordinate system cannot be established.
162	**PEBADPRP**	Ignoring function. The projection reference point is between the front and back planes.
163	**PEPRPVP**	Ignoring function. The projection reference point cannot be positioned on the view plane.
164	**PEBADBACK**	Ignoring function. The back plane is in front of the front plane.
200	**PEIGNSTRUCT**	Warning. Ignoring structures that do not exist.
201	**PENOSTRUCT**	Ignoring function. The specified structure does not exist.
202	**PENOELEM**	Ignoring function. The specified element does not exist.
203	**PEBADSPATH**	Ignoring function. Specified starting path not found in CSS.

Error	Symbolic Name	Description
204	**PECEILRNG**	Ignoring function. Specified search ceiling index out of range.
205	**PENOLABEL**	Ignoring function. The label does not exist in the open structure between the element pointer and the end of the structure.
206	**PENOLABELS**	Ignoring function. One or both of the labels does not exist in the open structure between the element pointer and the end of the structure.
207	**PEPATHDEPNEG**	Ignoring function. The specified path depth is less than 0.
208	**PEDISPRIRNG**	Ignoring function. The display priority is out of range.
250	**PENOINDEV**	Ignoring function. The specified device is not available on the specified workstation.
251	**PENOTREQUEST**	Ignoring function. The function requires the input device to be in **REQUEST** mode.
252	**PENOTSAMPLE**	Ignoring function. The function requires the input device to be in **SAMPLE** mode.
253	**PEBADPET**	Warning. The specified prompt/echo type is not available on the specified workstation. Prompt/echo type 1 will be used in its place.
254	**PEBADECHO**	Ignoring function. Invalid echo area/volume. **XMIN** ≥ **XMAX**, **YMIN** ≥ **YMAX**, or **ZMIN** > **ZMAX**.
255	**PENOPETWS**	Ignoring function. One of the echo area/volume boundary points is outside the range of the device.
256	**PEINQOVFL**	Warning. One input queue has overflowed.
257	**PENOQOVFL**	Ignoring function. Input queue has not overflowed.
258	**PEINQOVFLWSCL**	Warning. Input queue has overflowed, but associated workstation has been closed.
259	**PEBADCLASS**	Ignoring function. The input device class of the current input report does not match the class being requested.
260	**PEBADDATA**	Ignoring function. One of the fields within the input device data record is in error.
261	**PEBADIVAL**	Ignoring function. Initial value is invalid.

Error	Symbolic Name	Description
262	**PEPTSGTBUF**	Ignoring function. Number of points in the initial stroke is greater than the buffer size.
263	**PELENGTBUF**	Ignoring function. Length of the initial string is greater than the buffer size.
300	**PERESERVE**	Ignoring function. Item type is not allowed for user items.
301	**PEBDLNGTH**	Ignoring function. Item length is invalid.
302	**PENOITEM**	Ignoring function. No item is left in Metafile input.
303	**PEITMINV**	Ignoring function. Metafile item is invalid.
304	**PEBATITM**	Ignoring function. Item type is unknown.
305	**PEBADCNTS**	Ignoring function. Content of item data record is invalid for the specified item type.
306	**PEBDMXDR**	Ignoring function. Maximum item data record length is invalid.
307	**PEINTERPT**	Ignoring function. User item cannot be interpreted.
350	**PEESCAPE**	Warning. The specified escape is not available on one or more workstations in this implementation. The escape will be processed by those workstations on which it is available.
351	**PEESCDAT**	Ignoring function. One of the fields within the escape data record is in error.
400	**PENOAROPN**	Ignoring function. The archive file cannot be opened.
401	**PEMAXAR**	Ignoring function. Opening this archive file would exceed the maximum number of simultaneously open archive files.
402	**PEARIDINUSE**	Ignoring function. Archive file identifier already in use.
403	**PEBADARFILE**	Ignoring function. The archive file is not a PHIGS archive file.
404	**PENOTOPNAR**	Ignoring function. The specified archive file is not open.
405	**PECONFLICT**	Ignoring function. Name conflict occurred while conflict resolution flag has value **ABANDON**.

Error	Symbolic Name	Description
406	**PEARFULL**	Warning. The archive file is full. Any structures that were archived were archived in total.
407	**PESTRUCTAR**	Warning. Some of the specified structures do not exist on the archive file.
408	**PEARSTRUCT**	Warning. Some of the specified structures do not exist on the archive file. PHIGS will create empty structures in their places.
450	**PEBADERRFILE**	Ignoring function. The specified error file is invalid.
500	**PESMALLORDER**	Ignoring function. The specified order is less than 1.
501	**PECTLPOINTS**	Ignoring function. Not enough control points for specified order.
502	**PEBADORDER**	Ignoring function. The specified order is inconsistent with number of knots and control points.
503	**PEKNOTDECR**	Ignoring function. The knot sequence is not non-decreasing.
504	**PEINVALVIND**	Ignoring function. One or more of the vertex indices is out of range.
505	**PEDEGENFAS**	Warning. The fill area is degenerate.
506	**PEPARAMRANGE**	Ignoring function. Parameter range is inconsistent with the knots.
900	**PEOVFLPH**	Storage overflow has occurred in PHIGS.
901	**PEOVFLCSS**	Storage overflow has occurred in CSS.
902	**PEIOREAD**	Input/Output error has occurred while reading.
903	**PEIOWRITE**	Input/Output error has occurred while writing.
904	**PESENDWS**	Input/Output error has occurred while sending data to a workstation.
905	**PERECVWS**	Input/Output error has occurred while receiving data from a workstation.
906	**PELIBMAN**	Input/Output error has occurred during program library management.
907	**PERDWSDT**	Input/Output error has occurred while reading workstation description table.
908	**PEARITH**	Arithmetic error has occurred.

Error	Symbolic Name	Description
2200	**PEBUFSPAC**	Buffer overflow in input or inquiry function.
2201	**PEOUTRANG**	Start index out of range.
2000	**PEFTN2000**	Ignoring function. Enumeration type out of range.
2001	**PEFTN2001**	Ignoring function. Output parameter size insufficient.
2002	**PEFTN2002**	Ignoring function. List or set element not available.
2003	**PEFTN2003**	Ignoring function. Invalid data record.
2004	**PEFTN2004**	Ignoring function. Input parameter size out of range.
2005	**PEFTN2005**	Ignoring function. Invalid list of point lists.
2006	**PEFTN2006**	Ignoring function. Invalid list of filters.

PHIGS Tables

PHIGS Description Table

Data Type Abbreviations

I	Integer
E	Enumeration Type
L(type)	List of Values of a Given Type
MCV	Modelling Clipping Volume
P3	3D Point
R	Real
SET(NM)	Set of Eligible Names
V2/V3	2D/3D Vector
W	Workstation Type
n/s	Not Supported

Description Table Entry	Data Type	Default or Initial Value
number of available workstation types	I	1
list of available workstation types	L(W)	See table 1-1
maximum number of simultaneously open workstations	I	14
maximum number of simultaneously open archive files	I	0
number of available names for name sets	I	64
number of available character sets	I	1
character set	I	0
maximum length of normal filter list for ISS	I	n/s
maximum length of inverted filter list for ISS	I	n/s
polyline index	I	1
linetype	I	1
linewidth scale factor	R	1.0
polyline colour index	I	1
linetype ASF	E	INDIVIDUAL
linewidth scale factor ASF	E	INDIVIDUAL
polyline colour index ASF	E	INDIVIDUAL

Description Table Entry	Data Type	Default or Initial Value
polymarker index	I	1
marker type	I	3
marker size scale factor	R	1.0
polymarker colour index	I	1
marker type ASF	E	INDIVIDUAL
marker size scale factor ASF	E	INDIVIDUAL
polymarker colour index ASF	E	INDIVIDUAL
text index	I	1
text font	I	1 (Monospaced Roman Simplex)
text precision	E	STROKE
character expansion factor	R	1.0
character spacing	R	0.0
text colour index	I	1
text font ASF	E	INDIVIDUAL
text precision ASF	E	INDIVIDUAL
character expansion factor ASF	E	INDIVIDUAL
character spacing ASF	E	INDIVIDUAL
text colour index ASF	E	INDIVIDUAL
character height	R	0.01
character up vector	V2	(0.0,1.0)
character width	R	n/s
character base vector	V2	n/s
text path	E	RIGHT
text alignment (horizontal & vertical)	2xE	(NORMAL, NORMAL)
annotation text character height	R	0.01
annotation text character up vector	V2	(0.0,1.0)
annotation text character width	R	n/s
annotation text character base vector	V2	n/s

Description Table Entry	Data Type	Default or Initial Value
annotation text path	E	RIGHT
annotation text alignment (horizontal & vertical)	2xE	(NORMAL, NORMAL)
annotation style	I	1 (unconnected)
interior index	I	1
interior style	E	HOLLOW
interior style index	I	1
interior colour index	I	1
interior style ASF	E	INDIVIDUAL
interior style index ASF	E	INDIVIDUAL
interior colour index ASF	E	INDIVIDUAL
edge index	I	1
edge flag	E	OFF
edgetype	I	1
edgewidth scale factor	R	1.0
edge colour index	I	1
edge flag ASF	E	INDIVIDUAL
edgetype ASF	E	INDIVIDUAL
edgewidth scale factor ASF	E	INDIVIDUAL
edge colour index ASF	E	INDIVIDUAL
pattern size	2xR	n/s
pattern reference point	P3	n/s
pattern reference vectors	2xV3	n/s
pick identifier	I	0
view index	I	0
HLHSR identifier	I	0
name set	SET(NM)	no classes (empty set)
global modelling transformation	4x4xR	Identity
local modelling transformation	4x4xR	Identity

Description Table Entry	Data Type	Default or Initial Value
modelling clipping volume	MCV	n/s
modelling clipping indicator	E	NOCLIP
number of available generalized structure elements	I	0
maximum number of distinct planes in modelling clipping volumes	I	0
number of available modelling clipping operators	I	0
list of available modelling clipping operators		empty

PHIGS PLUS Description Table

Data Type Abbreviations

I	Integer
E	Enumeration Type
GCOLR	General Colour
L(type)	List of Values of a Given Type
MCV	Modelling Clipping Volume
P3	3D Point
R	Real
SET(NM)	Set of Eligible Names
V2/V3	2D/3D Vector
W	Workstation Type
n/s	Not Supported

Description Table Entry	Data Type	Default or Initial Value
polyline colour	GCOLR	(RGB,WHITE)
polyline shading method	I	1(NONE)
polyline shading method ASF	E	INDIVIDUAL
polymarker colour	GCOLR	(RGB,WHITE)
text colour	GCOLR	(RGB,WHITE)
face distinguishing mode	E	NONE
face culling mode	E	NONE
interior colour	GCOLR	(RGB,WHITE)
interior shading method	I	1(NONE)
ambient reflection coefficient	R	1.0
diffuse reflection coefficient	R	1.0
specular reflection coefficient	R	1.0
specular colour	GCOLR	(RGB,WHITE)
specular exponent	R	0.0
reflectance characteristics	I	1(NONE)
interior shading method ASF	E	INDIVIDUAL
reflectance properties ASF	E	INDIVIDUAL
reflectance characteristics ASF	E	INDIVIDUAL

Description Table Entry	Data Type	Default or Initial Value
back interior style	E	HOLLOW
back interior style index	I	1
back interior colour	GCOLR	(RGB,WHITE)
back interior shading method	I	1(NONE)
back ambient reflection coefficient	R	1.0
back diffuse reflection coefficient	R	1.0
back specular reflection coefficient	R	1.0
back specular colour	GCOLR	(RGB,WHITE)
back specular exponent	R	0.0
back reflectance characteristics	I	1(NONE)
back interior style ASF	E	INDIVIDUAL
back interior style index ASF	E	INDIVIDUAL
back interior colour ASF	E	INDIVIDUAL
back interior shading method ASF	E	INDIVIDUAL
back reflectance properties ASF	E	INDIVIDUAL
back reflectance characteristics ASF	E	INDIVIDUAL
light source state	L(I)	empty
edge colour	GCOLR	(RGB,WHITE)
curve approximation criteria type	I	0 or n/s
curve approximation criteria value	R	0 or n/s
curve approximation criteria ASF	E	INDIVIDUAL
surface approximation criteria type	I	n/s
surface approximation criteria value	2xR	n/s
surface approximation criteria ASF	E	n/s
rendering colour model	I	n/s
depth cue index	I	0
colour mapping index	I	n/s

PHIGS Workstation Description Table

Data Type Abbreviations

B	Bounding Range
CC	Chromaticity Coefficient
D	Data Record
E	Enumeration Type
FP	Font/Precision Pair
I	Integer
L(type)	List of values of a given type
P3	3D Point
R	Real
n/s	Not Supported

Workstation Description Table Entry	Data Type	Initial Value
workstation type	W	See table 1-1
workstation category	E	See table 1-1
device coordinate units	E	OTHER
maximum display space size:		
in device coordinates	3xR	(1.0,1.0,1.0)
in device address units	3xI	(1280,1024, 24 *z*-buffer planes)
number of available HLHSR identifiers	I	2
list of available HLHSR identifiers	L(I)	NONE,ZBUFF
number of available HLHSR modes	I	2
list of available HLHSR modes	L(I)	NONE,ZBUFF
number of predefined view indices (representations)	I	6
table of predefined view representations:		
view orientation matrix	4x4xR	Identity
view mapping matrix	4x4xR	Identity
view clipping limits	3xB	(0,1,0,1,0,1)
x-y clipping indicator	E	CLIP
back clipping indicator	E	CLIP
front clipping indicator	E	CLIP

Workstation Description Table Entry	Data Type	Initial Value
workstation classification	E	RASTER
dynamic modification accepted for:		
view representation	E	IRG
polyline bundle representation	E	IRG
polymarker bundle representation	E	IRG
text bundle representation	E	IRG
interior bundle representation	E	IRG
edge bundle representation	E	IRG
pattern representation	E	IRG
colour representation	E	IRG
workstation transformation	E	IRG
highlighting filter	E	IRG
invisibility filter	E	IRG
HLHSR mode	E	IRG
default value for deferral state:		
deferral mode	E	ASAP
modification mode	E	NIVE
number of available linetypes	L(I)	See table 1-2
number of available linewidths	I	1
nominal linewidth	R	1.0 (pixel)
minimum linewidth	R	1.0
maximum linewidth	R	1.0
number of predefined polyline indices (bundles)	I	5
table of predefined polyline bundles		See table 1-3
number of available marker types	I	8
list of available marker types	L(I)	See table 1-4
number of available marker sizes	I	0 (continuous)
nominal marker size	R	1.0 (9 pixels)
minimum marker size	R	0
maximum marker size	R	unlimited

Workstation Description Table Entry	Data Type	Initial Value
number of predefined polymarker indices (bundles)	I	5
table of predefined polymarker bundles		See table 1-5
number of text font and precision pairs	I	2
list of text font and precision pairs	L(FP)	See table 1-6
number of available character expansion factors	I	0 (continuous)
minimum character expansion factor	R	0.0
maximum character expansion factor	R	unlimited
number of available character heights	I	0 (continuous)
minimum character height	R	0.0
maximum character height	R	unlimited
number of predefined text indices (bundles)	I	6
table of predefined text bundles		See table 1-7
number of available annotation styles	I	2
list of available annotation styles	L(I)	See table 1-12
number of available interior styles	I	3
list of available interior styles	L(E)	See table 1-8
number of available hatch styles	I	0
list of available hatch styles	L(I)	empty
number of predefined interior indices (bundles)	I	5
table of predefined interior bundles		See table 1-9
number of available edgetypes	I	4
list of available edgetypes	L(I)	See table 1-2
number of available edgewidths	I	1
nominal edgewidth	R	1.0 (pixel)
minimum edgewidth	R	1.0
maximum edgewidth	R	1.0
number of predefined edge indices (bundles)	I	5
table of predefined edge bundles		See table 1-10
number of predefined pattern indices (representations)	I	0

Workstation Description Table Entry	Data Type	Initial Value
table of predefined pattern representations		empty
number of available colour models	I	2
list of available colour models	L(I)	1 (RGB)
default colour model	I	1 (RGB)
colour available	E	COLOUR
number of predefined colour indices (representations)	I	8
table of predefined colour representations		See table 1-11
number of available generalized drawing primitives 3 (**GDP3**)	I	0
list of available generalized drawing primitives 3 (**GDP3**)		empty
number of available generalized drawing primitives (**GDP**)	I	0
list of available generalized drawing primitives (**GDP**)		empty
number of display priorities supported	I	0 (unlimited)
maximum number of polyline bundle table entries	I	20
maximum number of polymarker bundle table entries	I	20
maximum number of text bundle table entries	I	20
maximum number of interior bundle table entries	I	20
maximum number of edge bundle table entries	I	20
maximum number of pattern table entries	I	0
maximum number of colour indices	I	256
maximum number of view indices	I	20
dynamic modification accepted for:		
structure content modification	E	IRG
post structure	E	IRG
unpost structure	E	IRG
delete structure	E	IRG
reference modification	E	IRG

Workstation Description Table Entry	Data Type	Initial Value
number of logical devices of class LOCATOR	I	0
number of logical input devices of class STROKE	I	0
number of logical input devices of class VALUATOR	I	0
number of logical devices of class CHOICE	I	0
number of logical input devices of class PICK	I	0
number of logical input devices of class STRING	I	0

PHIGS PLUS Workstation Description Table

Data Type Abbreviations

B	Bounding Range
CC	Chromaticity Coefficient
D	Data Record
E	Enumeration Type
FP	Font/Precision Pair
I	Integer
L(type)	List of values of a given type
P3	3D Point
R	Real
n/s	Not Supported

Workstation Description Table Entry	Data Type	Initial Value
number of available directly specifiable colour models	I	1 (RGB)
list of available directly specifiable color models	I	1 (RGB)
number of available rendering colour models	I	1 (RGB)
list of available rendering colour models	I	1 (RGB)
dynamic modification accepted for:		
data mapping representation	E	IRG
reflectance representation	E	IRG
parametric surface representation	E	IRG
light source representation	E	IRG
depth cue representation	E	IRG
colour mapping representation	E	IRG
table of predefined polyline bundles		See table 1-3
table of predefined polymarker bundles		See table 1-5
table of predefined text bundles		See table 1-7
table of predefined interior bundles		See table 1-9
table of predefined edge bundles		See table 1-10
maximum number of data mapping bundle table entries	I	n/s
number of predefined data mapping bundles	I	n/s

Workstation Description Table Entry	Data Type	Initial Value
maximum number of reflectance bundle table entries	I	20
number of predefined reflectance bundles	I	1
for every entry:		
reflectance index	I	1
reflectance characteristics	I	1
ambient reflection coefficient	R	1.0
diffuse reflection coefficient	R	1.0
specular reflection coefficient	R	1.0
specular colour	GCOLR	1
specular exponent	R	1.0
maximum number of parametric surface bundle table entries	I	n/s
number of predefined parametric surface bundles	I	n/s
number of predefined pattern representations	I	0
maximum number of light source table entries	I	12
number of predefined light source table indices	I	1
table of predefined light sources		See table 1-18
maximum number of depth cue table entries	I	6
number of predefined depth cue indices	I	2
table of predefined depth cue representations		See table 1-17
maximum number of colour mapping table entries	I	n/s
number of predefined colour mapping table entries	I	0 (n/s)
table of predefined colour mappings		n/s
number of available polyline shading models	I	2
list of available polyline shading methods		See table 1-13
number of available interior styles		See table 1-8
number of available data mapping methods	I	n/s
list of available data mapping methods	L(I)	n/s
number of available interior shading methods	I	2

Workstation Description Table Entry	Data Type	Initial Value
list of available interior shading methods	L(I)	See table 1-14
number of available reflectance characteristics values	I	4
list of available reflectance characteristics		See table 1-16
maximum non-uniform B-spline curve order	I	n/s
maximum trimming curve order	I	0 (n/s)
number of available curve approximation criteria types	I	0
list of available curve approximation criteria types	L(I)	n/s
maximum non-uniform b-spline surface order	I	0 (n/s)
number of available surface approximation criteria types	I	0
list of available surface approximation criteria types	L(I)	n/s
number of available trimming curve approximation criteria types	I	0
list of available trimming curve approximation criteria types	L(I)	n/s
number of available parametric surface characteristics types	I	n/s
list of available parametric surface characteristics types	L(I)	n/s
number of available light source types	I	4
list of available light source types		See table 1-15
maximum number of simultaneously active non-ambient light sources	I	12
number of available colour mapping methods	I	n/s
list of available colour mapping methods	L(I)	n/s
number of available true colours	I	2^{24}
maximum number of pseudo colour entries	I	n/s

Table 1-1. Workstation type and category

Type	C Name	Category
X drawable	**phigs_ws_type_x_drawable**	**OUTPUT**
X tool	**phigs_ws_type_x_tool**	**OUTIN**

Table 1-2. Available line and/or edge types

Type	C Name	Meaning
1	**PLINE_SOLID**	Solid
2	**PLINE_DASH**	Dashed
3	**PLINE_DOT**	Dotted
4	**PLINE_DOT_DASH**	Dot-dashed

Table 1-3. Predefined extended† polyline bundle table

Bundle Index	Linetype	Line Width Scale Factor	Colour Index‡	Shading Method†	Approx. Type†	Approx. Value†
1	Solid	1.0	1	None	N/A	N/A

†PHIGS PLUS extension.

‡ Predefined Extended Polymarker Bundle entries use colour model INDIRECT.

Table 1-4. Available marker types

Value	C Name	Meaning
1	**PMARKER_DOT**	Point
2	**PMARKER_PLUS**	Plus
3	**PMARKER_ASTERISK**	Asterisk
4	**PMARKER_CIRCLE**	Circle
5	**PMARKER_CROSS**	Cross
0	**PES_MARKER_DEF_STAR**	Asterisk
-1	**PES_MARKER_DIAMOND**	Diamond
-2	**PES_MARKER_FAST_DOT**	Fast Dot
-3	**PES_MARKER_TRIANGLE**	Triangle
-4	**PES_MARKER_SQUARE**	Square
-5	**PES_MARKER_INV_TRIANGLE**	Inverted Triangle
-6	**PES_MARKER_OCTAGON**	Octagon

Table 1-5. Predefined polymarker bundle table

Bundle Index	Marker Type	Marker Size Scale Factor	Colour Index†
1	Asterisk	1.0	1

†All Predefined Extended Polymarker Bundle entries (PHIGS PLUS extension) use colour model INDIRECT.

Table 1-6. Available text fonts and precisions

Font Number	C Name	Precisions Supported
1	**PFONT_MONO**	STROKE
2	**PFONT_COMPLEX**	STROKE

Table 1-7. Predefined extended text bundle table

Bundle Index	Font Number	Text Precision	Expansion Factor	Character Spacing	Colour Index†
1	1	STROKE	1.0	0.0	1

†All Predefined Extended Polymarker Bundle entries (PHIGS PLUS extension) use colour model INDIRECT.

Table 1-8. Available interior styles

C Name	Meaning
PSTYLE_HOLLOW	Hollow
PSTYLE_SOLID	Solid-filled
PSTYLE_EMPTY	Empty

Table 1-9. Predefined fill area interior bundle table

Bundle Index	Interior Style	Interior Style Index	Colour Index	Reflectance Equation†	Shading Method
1	Hollow	1	1	None	None

†All Predefined Extended Edge Bundle entries (PHIGS PLUS extension) use colour model INDIRECT; have back attribute values identical to the front; and have the following area properties:

Ambient Coefficient	Diffuse Coefficient	Specular Coefficient	Specular Colour	Specular Exponent	Transparency Coefficient
1.0	1.0	1.0	(RGB,1.0,1.0,1.0)	0.0	0.0

Table 1-10. Predefined edge bundle table

Bundle Index	Edge Flag	Edgetype	Edgewidth Scale Factor	Colour Index†
1	OFF	Solid	1.0	1

†All Predefined Extended Edge Bundle entries (PHIGS PLUS extension) use colour model INDIRECT.

Table 1-11. Predefined colour table

Colour Index	Red	Green	Blue	Description
0	0.0	0.0	0.0	Black
1	1.0	1.0	1.0	White
2	1.0	0.0	0.0	Red
3	0.0	1.0	0.0	Green
4	0.0	0.0	1.0	Blue
5	1.0	1.0	0.0	Yellow
6	0.0	1.0	1.0	Cyan
7	1.0	0.0	1.0	Magenta

Table 1-12. Available annotation styles

Value	C Name	Meaning
1	**PANNO_STYLE_UNCONNECTED**	Unconnected
2	**PANNO_STYLE_LEAD_LINE**	Lead Line

Table 1-13. Polyline shading methods

Value	C Name	Meaning
1	**PSD_NONE**	No Shading
2	**PSD_COLOUR**	Colour Shading

Table 1-14. Available interior shading methods

Value	C Name	Meaning
1	**PSD_NONE**	No Shading
2	**PSD_COLOUR**	Colour Shading

Table 1-15. Available light source types

Value	C Name	Meaning
1	**PLIGHT_AMBIENT**	Ambient Light Source
2	**PLIGHT_DIRECTIONAL**	Directional Light Source
3	**PLIGHT_POSITIONAL**	Positional Light Source
4	**PLIGHT_SPOT**	Spot Light Source

Table 1-16. Available reflectance characteristics

Value	C Name	Meaning
1	**PREFL_NONE**	No Reflectance Calculation Performed
2	**PREFL_AMBIENT**	Use Ambient Term
3	**PREFL_AMB_DIFF**	Use Ambient and Diffuse Terms
4	**PREFL_AMB_DIFF_SPEC**	Use Ambient, Diffuse, and Specular Terms

Table 1-17. Predefined depth cue table

Depth Cue Index	Depth Cue Mode	Depth Cue Reference Planes	Depth Cue Scale Factors	Depth Cue Colour
0	SUPPRESSED	(1.0,0.0)	(1.0,1.0)	(INDIRECT,0)
1	ALLOWED	(1.0,0.0)	(1.0,0.02)	(INDIRECT,0)

Table 1-18. Predefined light sources

Index	Type	Data Record
1	DIRECTIONAL	(RGB,1.0,1.0,1.0),0.0,0.0,1.0

C and FORTRAN Bindings

The following tables list the C and FORTRAN bindings for the PHIGS and PHIGS PLUS/PEX functions.

In the PHIGS table, an asterisk (*) means the function is a C binding function, and a dagger (†) means the function is a FORTRAN binding function. The C and FORTRAN bindings both split the **INQUIRE HLHSR FACILITIES** function into the following two functions:

INQUIRE HLHSR IDENTIFIER FACILITIES

INQUIRE HLHSR MODE FACILITIES

In the PHIGS PLUS table, the C binding names are from the ISO PHIGS PLUS working draft. Since the FORTRAN bindings are not yet defined as a standard, the names listed in this table are our attempt to anticipate what they will be. The functions shown in **bold** typeface are defined correctly in PEX-SI beta and will probably remain the same in the release to MIT. The functions shown in *italics* are those whose function and/or C binding name differ in the PEX-SI beta but are expected to be corrected in the release to MIT. The functions shown in regular font indicate those implemented in PEX-SI via an older function name that is not in the PHIGS PLUS Dp and will probably remain in the release to MIT.

PHIGS Function	C Binding	FORTRAN Binding
ADD NAMES TO SET	padd_names_set	pads
ANNOTATION TEXT RELATIVE	panno_text_rel	patr
ANNOTATION TEXT RELATIVE 3	panno_text_rel3	patr3
APPLICATION DATA	pappl_data	pap
ARCHIVE ALL STRUCTURES	par_all_structs	parast
ARCHIVE STRUCTURE NETWORKS	par_struct_nets	parsn
ARCHIVE STRUCTURES	par_structs	parst
AWAIT EVENT	pawait_event	pwait
BUILD TRANSFORMATION MATRIX	pbuild_tran_matrix	pbltm
BUILD TRANSFORMATION MATRIX 3	pbuild_tran_matrix3	pbltm3
CELL ARRAY	pcell_array	pca
CELL ARRAY 3	pcell_array3	pca3
CHANGE STRUCTURE IDENTIFIER	pchange_struct_id	pcstid
CHANGE STRUCTURE IDENTIFIER AND REFERENCES	pchange_struct_id_refs	pcstir
CHANGE STRUCTURE REFERENCES	pchange_struct_refs	pcstrf
CLOSE ARCHIVE FILE	pclose_ar_file	pclarf
CLOSE PHIGS	pclose_phigs	pclph
CLOSE STRUCTURE	pclose_struct	pclst
CLOSE WORKSTATION	pclose_ws	pclwk
COMPOSE MATRIX	pcompose_matrix	pcom
COMPOSE MATRIX 3	pcompose_matrix3	pcom3
COMPOSE TRANSFORMATION MATRIX	pcompose_tran_matrix	pcotm
COMPOSE TRANSFORMATION MATRIX 3	pcompose_tran_matrix3	pcotm3
COPY ALL ELEMENTS FROM STRUCTURE	pcopy_all_elems_struct	pcelst
CREATE STORE *	pcreate_store	N/S
DELETE ALL STRUCTURES	pdel_all_structs	pdas
DELETE ALL STRUCTURES FROM ARCHIVE	pdel_all_structs_ar	pdasar
DELETE ELEMENT	pdel_elem	pdel
DELETE ELEMENT RANGE	pdel_elem_range	pdelra
DELETE ELEMENTS BETWEEN LABELS	pdel_elems_labels	pdellb
DELETE STORE *	pdel_store	N/S
DELETE STRUCTURE	pdel_struct	pdst
DELETE STRUCTURE NETWORK	pdel_struct_net	pdsn
DELETE STRUCTURE NETWORKS FROM ARCHIVE	pdel_struct_nets_ar	pdsnar
DELETE STRUCTURES FROM ARCHIVE	pdel_structs_ar	pdstar

PHIGS Function (continued)	C Binding	FORTRAN Binding
ELEMENT SEARCH	pelem_search	pels
EMERGENCY CLOSE PHIGS	pemergency_close_phigs	peclph
EMPTY STRUCTURE	pempty_struct	pemst
ERROR HANDLING	perr_hand	perhnd
ERROR LOGGING	perr_log	perlog
ESCAPE	pescape	pesc
EVALUATE VIEW MAPPING MATRIX	peval_view_map_matrix	pevmm
EVALUATE VIEW MAPPING MATRIX 3	peval_view_map_matrix3	pevmm3
EVALUATE VIEW ORIENTATION MATRIX	peval_view_ori_matrix	pevom
EVALUATE VIEW ORIENTATION MATRIX 3	peval_view_ori_matrix3	pevom3
EXECUTE STRUCTURE	pexec_struct	pexst
FILL AREA	pfill_area	pfa
FILL AREA 3	pfill_area3	pfa3
FILL AREA SET	pfill_area_set	pfas
FILL AREA SET 3	pfill_area_set3	pfas3
FLUSH DEVICE EVENTS	pflush_events	pflush
GENERALIZED DRAWING PRIMITIVE	pgdp	pgdp
GENERALIZED DRAWING PRIMITIVE 3	pgdp3	pgdp3
GENERALIZED STRUCTURE ELEMENT	pgse	pgse
GET CHOICE	pget_choice	pgtch
GET ITEM TYPE FROM METAFILE	pget_item_type	pgtitm
GET LOCATOR	pget_loc	pgtlc
GET LOCATOR 3	pget_loc3	pgtlc3
GET PICK	pget_pick	pgtpk
GET STRING	pget_string	pgtst
GET STROKE	pget_stroke	pgtsk
GET STROKE 3	pget_stroke3	pgtsk3
GET VALUATOR	pget_val	pgtvl
INCREMENTAL SPATIAL SEARCH	pincr_spa_search	piss
INCREMENTAL SPATIAL SEARCH 3	pincr_spa_search3	piss3
INITIALIZE CHOICE	pinit_choice	pinch
INITIALIZE CHOICE 3	pinit_choice3	pinch3
INITIALIZE LOCATOR	pinit_loc	pinlc
INITIALIZE LOCATOR 3	pinit_loc3	pinlc3
INITIALIZE PICK	pinit_pick	pinpk

PHIGS Function (continued)	C Binding	FORTRAN Binding
INITIALIZE PICK 3	pinit_pick3	pinpk3
INITIALIZE STRING	pinit_string	pinst
INITIALIZE STRING 3	pinit_string3	pinst3
INITIALIZE STROKE	pinit_stroke	pinsk
INITIALIZE STROKE 3	pinit_stroke3	pinsk3
INITIALIZE VALUATOR	pinit_val	pinvl
INITIALIZE VALUATOR 3	pinit_val3	pinvl3
INQUIRE ALL CONFLICTING STRUCTURES	pinq_all_conf_structs	pqcst
INQUIRE ANNOTATION FACILITIES	pinq_anno_facs	pqanf
INQUIRE ARCHIVE FILES	pinq_ar_files	pqarf
INQUIRE ARCHIVE STATE VALUE	pinq_ar_st	pqars
INQUIRE CHOICE DEVICE STATE	pinq_choice_st	pqchs
INQUIRE CHOICE DEVICE STATE 3	pinq_choice_st3	pqchs3
INQUIRE COLOUR FACILITIES	pinq_colr_facs	pqcf
INQUIRE COLOUR MODEL	pinq_colr_model	pqcmd
INQUIRE COLOUR MODEL FACILITIES	pinq_colr_model_facs	pqcmdf
INQUIRE COLOUR REPRESENTATION	pinq_colr_rep	pqcr
INQUIRE CONFLICT RESOLUTION	pinq_conf_res	pqcnrs
INQUIRE CONFLICTING STRUCTURES IN NETWORK	pinq_conf_structs_net	pqcstn
INQUIRE CURRENT ELEMENT CONTENT	pinq_cur_elem_content	pqceco
INQUIRE CURRENT ELEMENT TYPE AND SIZE	pinq_cur_elem_type_size	pqcets
INQUIRE DEFAULT CHOICE DEVICE DATA	pinq_def_choice_data	pqdch
INQUIRE DEFAULT CHOICE DEVICE DATA 3	pinq_def_choice_data3	pqdch3
INQUIRE DEFAULT DISPLAY UPDATE STATE	pinq_def_disp_upd_st	pqddus
INQUIRE DEFAULT LOCATOR DEVICE DATA	pinq_def_loc_data	pqdlc
INQUIRE DEFAULT LOCATOR DEVICE DATA 3	pinq_def_loc_data3	pqdlc3
INQUIRE DEFAULT PICK DEVICE DATA	pinq_def_pick_data	pqdpk
INQUIRE DEFAULT PICK DEVICE DATA 3	pinq_def_pick_data3	pqdpk3
INQUIRE DEFAULT STRING DEVICE DATA	pinq_def_string_data	pqdst
INQUIRE DEFAULT STRING DEVICE DATA 3	pinq_def_string_data3	pqdst3
INQUIRE DEFAULT STROKE DEVICE DATA	pinq_def_stroke_data	pqdsk
INQUIRE DEFAULT STROKE DEVICE DATA 3	pinq_def_stroke_data3	pqdsk3
INQUIRE DEFAULT VALUATOR DEVICE DATA	pinq_def_val_data	pqdvl
INQUIRE DEFAULT VALUATOR DEVICE DATA 3	pinq_def_val_data3	pqdvl3
INQUIRE DISPLAY SPACE SIZE	pinq_disp_space_size	pqdsp

PHIGS Function (continued)	C Binding	FORTRAN Binding
INQUIRE DISPLAY SPACE SIZE 3	pinq_disp_space_size3	pqdsp3
INQUIRE DISPLAY UPDATE STATE	pinq_disp_upd_st	pqdus
INQUIRE DYNAMICS OF STRUCTURES	pinq_dyns_structs	pqdstr
INQUIRE DYNAMICS OF WORKSTATION ATTRIBUTES	pinq_dyns_ws_attrs	pqdswa
INQUIRE EDGE FACILITIES	pinq_edge_facs	pqedf
INQUIRE EDGE REPRESENTATION	pinq_edge_rep	pqedr
INQUIRE EDIT MODE	pinq_edit_mode	pqedm
INQUIRE ELEMENT CONTENT	pinq_elem_content	pqeco
INQUIRE ELEMENT POINTER	pinq_elem_ptr	pqep
INQUIRE ELEMENT TYPE AND SIZE	pinq_elem_type_size	pqets
INQUIRE ERROR HANDLING MODE	pinq_err_hand_mode	pqerhm
INQUIRE GENERALIZED DRAWING PRIMITIVE	pinq_gdp	pqgdp
INQUIRE GENERALIZED DRAWING PRIMITIVE 3	pinq_gdp3	pqgdp3
INQUIRE GENERALIZED STRUCTURE ELEMENT FACILITIES	pinq_gse_facs	pqgsef
INQUIRE HIGHLIGHTING FILTER	pinq_highl_filter	pqhlft
INQUIRE HLHSR FACILITIES	N/S	N/S
INQUIRE HLHSR IDENTIFIER FACILITIES	pinq_hlhsr_id_facs	pqhrif
INQUIRE HLHSR MODE	pinq_hlhsr_mode	pqhrm
INQUIRE HLHSR MODE FACILITIES	pinq_hlhsr_mode_facs	pqhrmf
INQUIRE INPUT QUEUE OVERFLOW	pinq_in_overf	pqiqov
INQUIRE INTERIOR FACILITIES	pinq_int_facs	pqif
INQUIRE INTERIOR REPRESENTATION	pinq_int_rep	pqir
INQUIRE INVISIBILITY FILTER	pinq_invis_filter	pqivft
INQUIRE LIST OF AVAILABLE GENERALIZED DRAWING PRIMITIVES	pinq_list_avail_gdp	pqegdp
INQUIRE LIST OF AVAILABLE GENERALIZED DRAWING PRIMITIVES 3	pinq_list_avail_gdp3	pqegd3
INQUIRE LIST OF AVAILABLE GENERALIZED STRUCTURE ELEMENTS	pinq_list_avail_gse	pqegse
INQUIRE LIST OF AVAILABLE WORKSTATION TYPES	pinq_list_avail_ws_types	pqewk
INQUIRE LIST OF COLOUR INDICES	pinq_list_colr_inds	pqeci
INQUIRE LIST OF EDGE INDICES	pinq_list_edge_inds	pqeedi
INQUIRE LIST OF INTERIOR INDICES	pinq_list_int_inds	pqeii
INQUIRE LIST OF PATTERN INDICES	pinq_list_pat_inds	pqepai
INQUIRE LIST OF POLYLINE INDICES	pinq_list_line_inds	pqepli
INQUIRE LIST OF POLYMARKER INDICES	pinq_list_marker_inds	pqepmi
INQUIRE LIST OF TEXT INDICES	pinq_list_text_inds	pqetxi
INQUIRE LIST OF VIEW INDICES	pinq_list_view_inds	pqevwi

PHIGS Function (continued)	C Binding	FORTRAN Binding
INQUIRE LOCATOR DEVICE STATE	pinq_loc_st	pqlcs
INQUIRE LOCATOR DEVICE STATE 3	pinq_loc_st3	pqlcs3
INQUIRE MODELLING CLIPPING FACILITIES	pinq_model_clip_facs	pqmclf
INQUIRE MORE SIMULTANEOUS EVENTS	pinq_more_simult_events	pqsim
INQUIRE NUMBER OF AVAILABLE LOGICAL INPUT DEVICES	pinq_num_avail_in	pqli
INQUIRE NUMBER OF DISPLAY PROIRITIES SUPPORTED	pinq_num_disp_pris	pqdp
INQUIRE OPEN STRUCTURE	pinq_open_struct	pqopst
INQUIRE PATHS TO ANCESTORS	pinq_paths_ances	pqpan
INQUIRE PATHS TO DESCENDANTS	pinq_paths_descs	pqpde
INQUIRE PATTERN FACILITIES	pinq_pat_facs	pqpaf
INQUIRE PATTERN REPRESENTATION	pinq_pat_rep	pqpar
INQUIRE PHIGS FACILITIES	pinq_phigs_facs	pqphf
INQUIRE PICK DEVICE STATE	pinq_pick_st	pqpks
INQUIRE PICK DEVICE STATE 3	pinq_pick_st3	pqpks3
INQUIRE POLYLINE FACILITIES	pinq_line_facs	pqplf
INQUIRE POLYLINE REPRESENTATION	pinq_line_rep	pqplr
INQUIRE POLYMARKER FACILITIES	pinq_marker_facs	pqpmf
INQUIRE POLYMARKER REPRESENTATION	pinq_marker_rep	pqpmr
INQUIRE POSTED STRUCTURES	pinq_posted_structs	pqpost
INQUIRE PREDEFINED COLOUR REPRESENTATION	pinq_pred_colr_rep	pqpcr
INQUIRE PREDEFINED EDGE REPRESENTATION	pinq_pred_edge_rep	pqpedr
INQUIRE PREDEFINED INTERIOR REPRESENTATION	pinq_pred_int_rep	pqpir
INQUIRE PREDEFINED PATTERN REPRESENTATION	pinq_pred_pat_rep	pqppar
INQUIRE PREDEFINED POLYLINE REPRESENTATION	pinq_pred_line_rep	pqpplr
INQUIRE PREDEFINED POLYMARKER REPRESENTATION	pinq_pred_marker_rep	pqppmr
INQUIRE PREDEFINED TEXT REPRESENTATION	pinq_pred_text_rep	pqptxr
INQUIRE PREDEFINED VIEW REPRESENTATION	pinq_pred_view_rep	pqpvwr
INQUIRE SET OF OPEN WORKSTATIONS	pinq_set_open_wss	pqopwk
INQUIRE SET OF WORKSTATIONS TO WHICH POSTED	pinq_wss_posted	pqwkpo
INQUIRE STRING DEVICE STATE	pinq_string_st	pqsts
INQUIRE STRING DEVICE STATE 3	pinq_string_st3	pqsts3
INQUIRE STROKE DEVICE STATE	pinq_stroke_st	pqsks
INQUIRE STROKE DEVICE STATE 3	pinq_stroke_st3	pqss3
INQUIRE STRUCTURE IDENTIFIERS	pinq_struct_ids	pqsid
INQUIRE STRUCTURE STATE VALUE	pinq_struct_st	pqstrs

PHIGS Function (continued)	C Binding	FORTRAN Binding
INQUIRE STRUCTURE STATUS	pinq_struct_status	pqstst
INQUIRE SYSTEM STATE VALUE	pinq_sys_st	pqsys
INQUIRE TEXT EXTENT	pinq_text_extent	pqtxx
INQUIRE TEXT FACILITIES	pinq_text_facs	pqtxf
INQUIRE TEXT REPRESENTATION	pinq_text_rep	pqtxr
INQUIRE VALUATOR DEVICE STATE	pinq_val_st	pqvls
INQUIRE VALUATOR DEVICE STATE 3	pinq_val_st3	pqvls3
INQUIRE VIEW FACILITIES	pinq_view_facs	pqvwf
INQUIRE VIEW REPRESENTATION	pinq_view_rep	pqvwr
INQUIRE WORKSTATION CATEGORY	pinq_ws_cat	pqwkca
INQUIRE WORKSTATION CLASSIFICATION	pinq_ws_class	pqwkcl
INQUIRE WORKSTATION CONNECTION AND TYPE	pinq_ws_conn_type	pqwkc
INQUIRE WORKSTATION STATE TABLE LENGTHS	pinq_ws_st_table	pqwksl
INQUIRE WORKSTATION STATE VALUE	pinq_ws_st	pqwkst
INQUIRE WORKSTATION TRANSFORMATION	pinq_ws_tran	pqwkt
INQUIRE WORKSTATION TRANSFORMATION 3	pinq_ws_tran3	pqwkt3
INTERPRET ITEM	pinterpret_item	pitm
LABEL	plabel	plb
MESSAGE	pmessage	pmsg
OFFSET ELEMENT POINTER	poffset_elem_ptr	posep
OPEN ARCHIVE FILE	popen_ar_file	poparf
OPEN PEX	popen_xphigs	poppx
OPEN PHIGS	popen_phigs	popph
OPEN STRUCTURE	popen_struct	popst
OPEN WORKSTATION	popen_ws	popwk
PACK DATA RECORD †	N/A	pprec
POLYLINE	ppolyline	ppl
POLYLINE 3	ppolyline3	ppl3
POLYMARKER	ppolymarker	ppm
POLYMARKER 3	ppolymarker3	ppm3
POST STRUCTURE	ppost_struct	ppost
READ ITEM FROM METAFILE	pread_item	prditm
REDRAW ALL STRUCTURES	predraw_all_structs	prst
REMOVE NAMES FROM SET	premove_names_set	pres
REQUEST CHOICE	preq_choice	prqch

PHIGS Function (continued)	C Binding	FORTRAN Binding
REQUEST LOCATOR	preq_loc	prqlc
REQUEST LOCATOR 3	preq_loc3	prqlc3
REQUEST PICK	preq_pick	prqpk
REQUEST STRING	preq_string	prqst
REQUEST STROKE	preq_stroke	prqsk
REQUEST STROKE 3	preq_stroke3	prqsk3
REQUEST VALUATOR	preq_val	prqvl
RESTORE MODELLING CLIPPING VOLUME	prestore_model_clip_vol	prmcv
RETRIEVE ALL STRUCTURES	pret_all_structs	prast
RETRIEVE PATHS TO ANCESTORS	pret_paths_ances	prepan
RETRIEVE PATHS TO DESCENDANTS	pret_paths_descs	prepde
RETRIEVE STRUCTURE IDENTIFIERS	pret_struct_ids	prsid
RETRIEVE STRUCTURE NETWORKS	pret_struct_nets	presn
RETRIEVE STRUCTURES	pret_structs	prest
ROTATE	protate	pro
ROTATE X	protate_x	prox
ROTATE Y	protate_y	proy
ROTATE Z	protate_z	proz
SAMPLE CHOICE	psample_choice	psmch
SAMPLE LOCATOR	psample_loc	psmlc
SAMPLE LOCATOR 3	psample_loc3	psmlc3
SAMPLE PICK	psample_pick	psmpk
SAMPLE STRING	psample_string	psmst
SAMPLE STROKE	psample_stroke	psmsk
SAMPLE STROKE 3	psample_stroke3	psmsk3
SAMPLE VALUATOR	psample_val	psmvl
SCALE	pscale	psc
SCALE 3	pscale3	psc3
SET ANNOTATION STYLE	pset_anno_style	psans
SET ANNOTATION TEXT ALIGNMENT	pset_anno_align	psatal
SET ANNOTATION TEXT CHARACTER HEIGHT	pset_anno_char_ht	psatch
SET ANNOTATION TEXT CHARACTER UP VECTOR	pset_anno_char_up_vec	psatcu
SET ANNOTATION TEXT PATH	pset_anno_path	psatp
SET CHARACTER EXPANSION FACTOR	pset_char_expan	pschxp
SET CHARACTER HEIGHT	pset_char_ht	pschh

PHIGS Function (continued)	C Binding	FORTRAN Binding
SET CHARACTER SPACING	pset_char_space	pschsp
SET CHARACTER UP VECTOR	pset_char_up_vec	pschup
SET CHOICE MODE	pset_choice_mode	pschm
SET COLOUR MODEL	pset_colr_model	pscmd
SET COLOUR REPRESENTATION	pset_colr_rep	pscr
SET CONFLICT RESOLUTION	pset_conf_res	pscnrs
SET DISPLAY UPDATE STATE	pset_disp_upd_st	psdus
SET EDGE COLOUR INDEX	pset_edge_colr_ind	psedci
SET EDGE FLAG	pset_edge-flag	psedfg
SET EDGE INDEX	pset_edge_ind	psedi
SET EDGE REPRESENTATION	pset_edge_rep	psedr
SET EDGETYPE	pset_edgetype	psedt
SET EDGEWIDTH SCALE FACTOR	pset_edgewidth	psewsc
SET EDIT MODE	pset_edit_mode	psedm
SET ELEMENT POINTER	pset_elem_ptr	psep
SET ELEMENT POINTER AT LABEL	pset_elem_ptr_label	pseplb
SET ERROR HANDLING *	pset_err_hand	N/S
SET ERROR HANDLING MODE	pset_err_hand_mode	pserhm
SET GLOBAL TRANSFORMATION	pset_global_tran	psgmt
SET GLOBAL TRANSFORMATION 3	pset_global_tran3	psgmt3
SET HIGHLIGHTING FILTER	pset_highl_filter	pshlft
SET HLHSR IDENTIFIER	pset_hlhsr_id	pshrid
SET HLHSR MODE	pset_hlhsr_mode	pshrm
SET INDIVIDUAL ASF	pset_indiv_asf	psiasf
SET INTERIOR COLOUR INDEX	pset_int_colr_ind	psici
SET INTERIOR INDEX	pset_int_ind	psii
SET INTERIOR REPRESENTATION	pset_int_rep	psir
SET INTERIOR STYLE	pset_int_style	psis
SET INTERIOR STYLE INDEX	pset_int_style_ind	psisi
SET INVISIBILITY FILTER	pset_invis_filter	psivft
SET LINETYPE	pset_linetype	psln
SET LINEWIDTH SCALE FACTOR	pset_linewidth	pslwsc
SET LOCAL TRANSFORMATION	pset_local_tran	pslmt
SET LOCAL TRANSFORMATION 3	pset_local_tran3	pslmt3
SET LOCATOR MODE	pset_loc_mode	pslcm

PHIGS Function (continued)	C Binding	FORTRAN Binding
SET MARKER SIZE SCALE FACTOR	pset_marker_size	psmksc
SET MARKER TYPE	pset_marker_type	psmk
SET MODELLING CLIPPING INDICATOR	pset_model_clip_ind	psmcli
SET MODELLING CLIPPING VOLUME	pset_model_clip_vol	psmcv
SET MODELLING CLIPPING VOLUME 3	pset_model_clip_vol3	psmcv3
SET PATTERN REFERENCE POINT	pset_pat_ref_point	psparf
SET PATTERN REFERENCE POINT AND VECTORS	pset_pat_ref_point_vecs	psprpv
SET PATTERN REPRESENTATION	pset_pat_rep	pspar
SET PATTERN SIZE	pset_pat_size	pspa
SET PICK FILTER	pset_pick_filter	pspkft
SET PICK IDENTIFIER	pset_pick_id	pspkid
SET PICK MODE	pset_pick_mode	pspkm
SET POLYLINE COLOUR INDEX	pset_line_colr_ind	pslci
SET POLYLINE INDEX	pset_line_ind	psli
SET POLYLINE REPRESENTATION	pset_line_rep	pslir
SET POLYMARKER COLOUR INDEX	pset_marker_colr_ind	pspmci
SET POLYMARKER INDEX	pset_marker_ind	pspmi
SET POLYMARKER REPRESENTATION	pset_marker_rep	pspmr
SET STRING MODE	pset_string_mode	psstm
SET STROKE MODE	pset_stroke_mode	psskm
SET TEXT ALIGNMENT	pset_text_align	pstxal
SET TEXT COLOUR INDEX	pset_text_colr_ind	pstxci
SET TEXT FONT	pset_text_font	pstxfn
SET TEXT INDEX	pset_text_ind	pstxi
SET TEXT PATH	pset_text_path	pstxp
SET TEXT PRECISION	pset_text_prec	pstxpr
SET TEXT REPRESENTATION	pset_text_rep	pstxr
SET VALUATOR MODE	pset_val_mode	psvlm
SET VIEW INDEX	pset_view_ind	psvwi
SET VIEW REPRESENTATION	pset_view_rep	psvwr
SET VIEW REPRESENTATION 3	pset_view_rep3	psvwr3
SET VIEW TRANSFORMATION INPUT PRIORITY	pset_view_tran_in_pri	psvtip
SET WORKSTATION VIEWPORT	pset_ws_vp	pswkv
SET WORKSTATION VIEWPORT 3	pset_ws_vp3	pswkv3
SET WORKSTATION WINDOW	pset_ws_win	pswkw

PHIGS Function (continued)	C Binding	FORTRAN Binding
SET WORKSTATION WINDOW 3	**pset_ws_win3**	**pswkw3**
TEXT	**ptext**	**ptx**
TEXT 3	**ptext3**	**ptx3**
TRANSFORM POINT	**ptran_point**	**ptp**
TRANSFORM POINT 3	**ptran_point3**	**ptp3**
TRANSLATE	**ptranslate**	**ptr**
TRANSLATE 3	**ptranslate3**	**ptr3**
UNPACK DATA RECORD †	**N/A**	**purec**
UNPOST ALL STRUCTURES	**punpost_all_structs**	**pupast**
UNPOST STRUCTURE	**punpost_struct**	**pupost**
UPDATE WORKSTATION	**pupd_ws**	**puwk**
WRITE ITEM TO METAFILE	**pwrite_item**	**pwitm**

PHIGS PLUS/PEX Binding	C Binding	FORTRAN Binding
CELL ARRAY 3 PLUS	pcell_array3_plus	pca3p
COMPUTE FILL AREA SET GEOMETRIC NORMAL	pcomp_fill_area_set_gnorm	pcfasn
FILL AREA SET 3 WITH DATA	pfill_area_set3_data	pfas3d
INQUIRE CURVE AND SURFACE FACILITIES	pinq_curv_surf_facs	pqcsf
INQUIRE DEPTH CUE FACILITIES	pinq_dcue_facs	pqdcf
INQUIRE DEPTH CUE REPRESENTATION	pinq_dcue_rep	pqdcr
INQUIRE DIRECT COLOUR MODEL FACILITIES	pinq_direct_colr_model_facs	pqdcmf
INQUIRE DYNAMICS OF WORKSTATION PLUS	pinq_dyns_ws_plus	pqdwkp
INQUIRE EDGE REPRESENTATION PLUS	pinq_edge_rep_plus	pqerp
INQUIRE INTERIOR FACILITIES PLUS	pinq_int_facs_plus	pqifp
INQUIRE INTERIOR REPRESENTATION PLUS	pinq_int_rep_plus	pqirp
INQUIRE LIGHT SOURCE FACILITIES	pinq_light_src_facs	pqlsf
INQUIRE LIGHT SOURCE REPRESENTATION	pinq_light_src_rep	pqlsr
INQUIRE LIST OF DEPTH CUE INDICES	pinq_list_dcue_inds	pqedci
INQUIRE LIST OF LIGHT SOURCE INDICES	pinq_list_light_src_inds	pqelsi
INQUIRE PATTERN REPRESENTATION PLUS	pinq_pat_rep_plus	pqparp
INQUIRE POLYLINE FACILITIES PLUS	pinq_line_facs_plus	pqlfp
INQUIRE POLYLINE REPRESENTATION PLUS	pinq_line_rep_plus	pqlrp
INQUIRE POLYMARKER REPRESENTATION PLUS	pinq_marker_rep_plus	pqmrp
INQUIRE PREDEFINED DEPTH CUE REPRESENTATION	pinq_pred_dcue_rep	pqpdcr
INQUIRE PREDEFINED EDGE REPRESENTATION PLUS	pinq_pred_edge_rep_plus	pqperp
INQUIRE PREDEFINED INTERIOR REPRESENTATION PLUS	pinq_pred_int_rep_plus	pqpirp
INQUIRE PREDEFINED LIGHT SOURCE REPRESENTATION	pinq_pred_light_src_rep	pqplsr
INQUIRE PREDEFINED PATTERN REPRESENTATION PLUS	pinq_pred_pat_rep_plus	pqpprp
INQUIRE PREDEFINED POLYLINE REPRESENTATION PLUS	pinq_pred_line_rep_plus	pqplrp
INQUIRE PREDEFINED POLYMARKER REPRESENTATION PLUS	pinq_pred_marker_rep_plus	pqpmrp
INQUIRE PREDEFINED TEXT REPRESENTATION PLUS	pinq_pred_text_rep_plus	pqptrp
INQUIRE RENDERING COLOUR MODEL FACILITIES	pinq_rend_colr_model_facs	pqrcmf
INQUIRE TEXT REPRESENTATION PLUS	pinq_text_rep_plus	pqtxrp
INQUIRE WORKSTATION STATE TABLE LENGTHS PLUS	pinq_ws_st_table_plus	pqwslp
NON-UNIFORM B-SPLINE CURVE	pnuni_bsp_curv	pnubsc
NON-UNIFORM B-SPLINE SURFACE	pnuni_bsp_surf	pnubss
POLYLINE SET 3 WITH DATA	ppolyline_set3_data	ppls3d
QUADRILATERAL MESH 3 WITH DATA	pquad_mesh3_data	pme3d
SET REFLECTANCE PROPERTIES	pset_refl_prop	psap

PHIGS PLUS/PEX Binding (continued)	C Binding	FORTRAN Binding
SET BACK REFLECTANCE PROPERTIES	pset_back_refl_prop	psbap
SET BACK INTERIOR COLOUR	pset_back_int_colr	psbic
SET BACK INTERIOR REFLECTANCE EQUATION	pset_back_refl_eqn	psbirfe
SET BACK INTERIOR SHADING METHOD	pset_back_int_shad_meth	psbism
SET BACK INTERIOR STYLE	pset_back_int_style	psbis
SET BACK INTERIOR STYLE INDEX	pset_back_int_style_ind	psbisi
SET CURVE APPROXIMATION CRITERIA	pset_curve_approx	pscac
SET DEPTH CUE INDEX	pset_dcue_ind	psdci
SET DEPTH CUE REPRESENTATION	pset_dcue_rep	psdcr
SET EDGE COLOUR	pset_edge_colr	psec
SET EDGE REPRESENTATION PLUS	pset_edge_rep_plus	pserp
SET FACE CULLING MODE	pset_face_cull_mode	psfcm
SET FACE DISTINGUISHING MODE	pset_face_disting_mode	psfdm
SET INTERIOR COLOUR	pset_int_colr	psic
SET REFLECTANCE EQUATION	pset_refl_eqn	psirfe
SET INTERIOR REPRESENTATION PLUS	pset_int_rep_plus	psirp
SET INTERIOR SHADING METHOD	pset_int_shad_meth	psism
SET LIGHT SOURCE REPRESENTATION	pset_light_src_rep	psisr
SET LIGHT SOURCE STATE	pset_light_src_state	psiss
SET OF FILL AREA SET 3 WITH DATA	pset_of_fill_area_set3_data	psfa3d
SET PARAMETRIC SURFACE CHARACTERISTICS	pset_param_surf_chars	pspsc
SET PATTERN REPRESENTATION PLUS	pset_pat_rep_plus	psprp
SET POLYLINE COLOUR	pset_line_colr	psic
SET POLYLINE REPRESENTATION PLUS	pset_line_rep_plus	psirp
SET POLYLINE SHADING METHOD	pset_line_shad_meth	psism
SET POLYMARKER COLOUR	pset_marker_colr	psmc
SET POLYMARKER REPRESENTATION PLUS	pset_marker_rep_plus	psmrp
SET RENDERING COLOUR MODEL	pset_rend_colr_model	psrcm
SET SURFACE APPROXIMATION CRITERIA	pset_surf_approx	pssac
SET TEXT COLOUR	pset_text_colr	pstxc
SET TEXT REPRESENTATION PLUS	pset_text_rep_plus	pstxrp
TRIANGLE STRIP 3 WITH DATA	ptri_strip3_data	pts3d

Example Programs

The example programs are provided as working ES/PEX programs. The programs may be found in the following directory on the distribution tape: **/usr/people/fstest/demo/pexexamples.**

The programs demonstrate the minimal amount of coding necessary to perform the functions stated within each program. Care has been taken to demonstrate a "proper" coding style to ensure that these programs may be used with other, larger models. There are many methods used in the example programs that are strictly a reflection of the authors' style. There is no "one way" to write ES/PEX programs; however, these example programs represent one way to demonstrate the basic functionality.

There are four programs implemented in two different ways. One way uses the Xlib calls and the other way uses the Motif Graphical User Interface. **example1.c** through **example4.c** use the Xlib calls for the user interface. **motif1.c** through **motif4.c** use the Motif Graphical User Interface.

Both sets of programs link with the same PHIGS procedures (**phigs1.c** through **phigs4.c**). Therefore, the PHIGS functionality is identical; only the user interface is different. This is done to demonstrate the different programming environments between Xlib calls and Motif calls.

The PHIGS functionality in the example programs progresses from a very simple program (**1**), which displays two objects and exits, to a fully interactive program (**4**) which demonstrates the use of the dials device along with picking on 3D objects.

Users should become familiar with what the example programs do before "pasting" them into other applications. Many hours of frustration may be avoided by taking time to understand the reasoning behind the implementation of these example programs.

Makefile

```
CCOPT = -systype bsd43
CDEBUG = -g

DEFNS = -Wf,-XNp50000 -Wf,-XNd50000

CFLAGS = $(CDEBUG) $(CCOPT) $(DEFNS) -I/bsd43/usr/include/X11R3
LDFLAGS = $(CDEBUG) $(CCOPT)
FIN =  -lm -lc /usr/lib/libc.a


# ------------------------------------------------------------------------
# Variables specific to clients created by this Makefile - edit as needed

OBJS1 = motif1.o
LIBS1 = -lPEXapi -lXm -lXt -lX11
OBJS2 = phigs2.o motif2.o
LIBS2 = -lPEXapi -lXm -lXt -lXinput -lXext -lX11
OBJS3 = phigs3.o motif3.o
LIBS3 = -lPEXapi -lXm -lXt -lXinput -lXext -lX11
OBJS4 = phigs4.o motif4.o
LIBS4 = -lPEXapi -lXm -lXt -lXinput -lXpick -lXext -lX11

XOBJS1 = example1.o
XLIBS1 = -lPEXapi -lX11
XOBJS2 = phigs2.o example2.o
XLIBS2 = -lPEXapi -lX11
XOBJS3 = phigs3.o example3.o
XLIBS3 = -lPEXapi -lXinput -lXext -lX11
XOBJS4 = phigs4.o example4.o
XLIBS4 = -lPEXapi -lXinput -lXpick -lXext -lX11

ALL   = motif1 motif2 motif3 motif4 example1 example2 example3 example4

# ------------------------------------------------------------------------
# Commands specific to clients created by this Makefile - edit as needed

all: $(ALL)

clean:
   rm -f *.o $(ALL)

motif1: $(OBJS1)
   $(CC) $(LDFLAGS) -o motif1 $(OBJS1) $(LIBS1) $(FIN)
```

```
motif2: $(OBJS2)
   $(CC) $(LDFLAGS) -o motif2 $(OBJS2) $(LIBS2) $(FIN)

motif3: $(OBJS3)
   $(CC) $(LDFLAGS) -o motif3 $(OBJS3) $(LIBS3) $(FIN)

motif4: $(OBJS4)
   $(CC) $(LDFLAGS) -o motif4 $(OBJS4) $(LIBS4) $(FIN)

example1: $(XOBJS1)
     $(CC) $(LDFLAGS) -o example1 $(XOBJS1) $(XLIBS1) $(FIN)

example2: $(XOBJS2)
     $(CC) $(LDFLAGS) -o example2 $(XOBJS2) $(XLIBS2) $(FIN)

example3: $(XOBJS3)
     $(CC) $(LDFLAGS) -o example3 $(XOBJS3) $(XLIBS3) $(FIN)

example4: $(XOBJS4)
     $(CC) $(LDFLAGS) -o example4 $(XOBJS4) $(XLIBS4) $(FIN)
```

example1.c

```
/*
                example1.c

        This program defines and displays two squares on the open
        PEX workstation. One square is a 2d polyline object and the
        other is a 2d fillarea object.
        This program shows the minimum PEX calls required to display
        simple objects. System defaults are used and there is no
        event handling.

        Author: James Buckmiller  May 1990.
        Modified: J. Buckmiller Mar 1991.  Approved C binding

        Copyright (C) 1990, Evans & Sutherland
*/

#include <X11/Xlib.h>              /* Include Xlib definitions        */
#include <X11/Xatom.h>
#include <X11/phigs/phigs.h>       /* Include Phigs/Phigs+ extensions */

#define POLYSQUARE     1           /* Define Structure name constants */
#define FILLSQUARE     2
#define DISPLAY_STRUCT 3
#define WS             1           /* Workstation ID number           */
#define WINPOSX        100
#define WINPOSY        100
#define WINWIDTH       600
#define WINHEIGHT      600

Window myWin;
Display *dpy;

main()
{
  char *display = NULL;

  if (!(dpy = XOpenDisplay(display)))    /* Attempt to open the display */
   {
    perror("Cannot open display\n");
    exit(-1);
   }
```

```
                    /* Create a simple, unmapped input/output window */
  myWin = XCreateSimpleWindow(dpy, RootWindow(dpy, DefaultScreen(dpy)),
          WINPOSX, WINPOSY, WINWIDTH, WINHEIGHT, 0,NULL,NULL);
                    /* Change the window name property */
  XChangeProperty(dpy, myWin, XA_WM_NAME, XA_STRING, 8,
        PropModeReplace, "Example 1:   PHIGS Squares", 26);
                   /* Map the window for display */
  XMapWindow(dpy, myWin);

                   /* Begin PHIGS calls          */
  StartPhigs(dpy, myWin);

}

/*
   StartPhigs

   This routine is the top level routine that calls all supporting
   routines in the logical order of a usual phigs routine i.e.
   open PEX, setup the workstation parameters and define the phigs
   structure.
*/

StartPhigs(dpy, win)          /* Routine to start phigs calls */
Display *dpy;
Window win;
{
  OpenPex(dpy);               /* Open PEX                              */
  SetupWorkstation(dpy, win); /* Setup PHIGS Workstation parameters */
  Make_squares();             /* Create Phigs structures               */
  sleep(5);                   /* Display structures for 5 seconds      */
  pclose_ws(WS);              /* Close workstation                     */
  pclose_phigs();             /* Close PHIGS                           */
}

/*
   OpenPex

   This routine Opens PEX on the display that was passed
   as an argument.
*/

OpenPex(dpy)
Display *dpy;
```

```
{
  Pxphigs_info  xinfo;
  unsigned long  infomask;

  xinfo.display = dpy;
  xinfo.rmdb    = NULL;
  xinfo.appl_id.name = NULL;
  xinfo.appl_id.class = NULL;
  xinfo.args.argc_p    = NULL;
  xinfo.args.argv      = NULL;
  xinfo.flags.no_monitor = 1;
  xinfo.flags.force_client_SS = 0;

  infomask = PXPHIGS_INFO_DISPLAY | PXPHIGS_INFO_FLAGS_NO_MON;

  /* Open Pex */
  popen_xphigs((char*)NULL, PDEF_MEM_SIZE, infomask, &xinfo);
}

/*
    SetupWorkstation

    This routine opens a PHIGS workstation and sets the structure edit mode
    to insert elements. The update state is also set to PWAIT causing the
    workstation display to be updated only upon request.
*/

SetupWorkstation(dpy, win)
Display *dpy;
Window   win;
{
  Pconnid_x_drawable   conn;

  conn.display     = dpy;
  conn.drawable_id = win;
  popen_ws(WS, (Pconnid *) (&conn), phigs_ws_type_x_drawable);/*Open WS*/

  pset_edit_mode(PEDIT_INSERT);     /* Set edit mode to insert elements */
                                    /* Set update state to WAIT */
  pset_disp_upd_st(WS, PDEFER_WAIT, PMODE_NIVE);
}

/*
   Make_squares
```

```
   Make_squares defines a polyline square and a fillarea square in 2
   dimensions. Structures POLYSQUARE and FILLSQUARE are defined to contain
   these data elements along with color and style attributes to be applied
   to the data elements. A higher level structure DISPLAY_STRUCT is defined
   to include both the POLYSQUARE and FILLSQUARE structures and is then
   posted to the open workstation to be displayed.
*/

Make_squares()
{

 static Ppoint     line_points[]=    /*  Define points for line drawing */
   { {  0.5, 0.5} , {  0.9, 0.5} ,
     {  0.9, 0.9} , {  0.5, 0.9} ,
     {  0.5, 0.5} };

 static Ppoint    fill_points[]=    /*  Define points for filled drawing */
   { {  0.1, 0.1} , {  0.5, 0.1} ,
     {  0.5, 0.5} , {  0.1, 0.5} };

 Ppoint_list Line_list, Fill_list;

 popen_struct(POLYSQUARE);        /* Open line drawing structure          */
   pset_line_colr_ind(2);         /* Assign default index color 2 to lines*/
   Line_list.num_points = 5;      /* Fill in number of points in list     */
   Line_list.points = line_points; /* Pointer to point array              */
   ppolyline(&Line_list);         /* Create a polyline element            */
 pclose_struct();                 /* Close line drawing structure         */

 popen_struct(FILLSQUARE);        /* Open filled drawing structure        */
   pset_int_style(PSTYLE_SOLID);/* Set interior style to be solid         */
   pset_int_colr_ind(3);          /* Assign default index color 3 to lines*/
   Fill_list.num_points = 4;      /* Fill in number of points in list     */
   Fill_list.points = fill_points; /* Pointer to point array              */
   pfill_area(&Fill_list);        /* Create a fill area element           */
 pclose_struct();                 /* Close filled drawing structure       */

 popen_struct( DISPLAY_STRUCT ); /* Open the top level display structure */
   pexec_struct(POLYSQUARE);     /* Include the line drawn square        */
   pexec_struct(FILLSQUARE);     /* Include the filled square            */
 pclose_struct();

 ppost_struct(WS, DISPLAY_STRUCT, 1.0);   /* Post DISPLAY_STRUCT, prio 1 */
 pupd_ws(WS, PUPD_PERFORM);              /* Update the workstation      */
}
```

motif1.c

```
/*
        m o t i f 1 . c

This program creates a drawing surface via a top-level shell widget. Two
PEX objects are then drawn on this surface and the program displays the
image for 5 seconds before exiting.

      Author: Rich Thomson
       Date: Thursday, June 12th, 1990
       Modified J. Buckmiller    Mar 1991.    Approved C binding
      Copyright (C) 1990, Evans & Sutherland Computer Corporation
*/

#include <X11/Xlib.h>              /* Include Xlib definitions        */
#include <X11/phigs/phigs.h>       /* Include Phigs/Phigs+ extensions */
#include <X11/Intrinsic.h>         /* Toolkit intrinsics              */
#include <Xm/Xm.h>/* resource names                    */

#define POLYSQUARE      1          /* Define Structure name constants */
#define FILLSQUARE      2
#define DISPLAY_STRUCT  3
#define WS              1          /* Workstation ID number           */
#define WINPOSX         100
#define WINPOSY         100
#define WINWIDTH        600
#define WINHEIGHT       600

/*
main

The main routine creates the widget hierarchy for the program, realizes the
hierarchy and then calls StartPhigs. The widget hierarchy used here is:
  motif1 (class topLevelShell)
The display connection (dpy) and window ID (drawWindow) of the top-level
shell widget are available after the widget hierarchy has been realized.
*/
main(argc, argv)
      int argc;
      char *argv[];
{
   register int n;
   Arg args[10];
   Widget topLevel;
                                   /* create topLevelShell widget */
```

```
    topLevel = XtInitialize("motif1", "Example", NULL, 0, &argc, argv);

    n = 0;
    XtSetArg(args[n], XmNheight, WINHEIGHT); n++;
    XtSetArg(args[n], XmNwidth, WINWIDTH); n++;
    XtSetArg(args[n], XmNx, WINPOSX); n++;
    XtSetArg(args[n], XmNy, WINPOSY); n++;
    XtSetArg(args[n], XmNtitle, "Example 1"); n++;
    XtSetValues(topLevel, args, n);/* set position and size resources */

    XtRealizeWidget(topLevel);    /* realize (and map) topLevel widget */

    StartPhigs(XtDisplay(topLevel), XtWindow(topLevel));
                          /* Begin PHIGS calls */
}

/*
StartPhigs

This routine is the top level routine that calls all supporting routines
in the logical order of a usual phigs routine i.e., open PEX, setup the
workstation parameters and define the phigs structure.
*/

StartPhigs(dpy, win)           /* Routine to start phigs calls */
Display *dpy;
Window win;
{
    OpenPex(dpy);                /* Open PEX                              */
    SetupWorkstation(dpy, win); /* Setup PHIGS Workstation parameters */
    Make_squares();              /* Create Phigs structures               */
    sleep(5);                    /* Display structures for 5 seconds      */
    pclose_ws(WS);               /* Close workstation                     */
    pclose_phigs();              /* Close PHIGS                           */
}

/*
OpenPex

This routine Opens PEX on the display that was passed as an argument.
*/
OpenPex(dpy)
Display *dpy;

{
```

```
    Pxphigs_info  xinfo;
    unsigned long  infomask;

    xinfo.display = dpy;
    xinfo.rmdb    = NULL;
    xinfo.appl_id.name = NULL;
    xinfo.appl_id.class = NULL;
    xinfo.args.argc_p    = NULL;
    xinfo.args.argv      = NULL;
    xinfo.flags.no_monitor = 1;
    xinfo.flags.force_client_SS = 0;

    infomask = PXPHIGS_INFO_DISPLAY | PXPHIGS_INFO_FLAGS_NO_MON;

                        /* Open Pex */
    popen_xphigs((char*)NULL, PDEF_MEM_SIZE, infomask, &xinfo);

}

/*
SetupWorkstation

This routine opens a PHIGS workstation and sets the structure edit mode to
insert elements. The update state is also set to PWAIT causing the
workstation display to be updated only upon request.
*/

SetupWorkstation(dpy, win)
Display *dpy;
Window   win;
{
    Pconnid_x_drawable   conn;

    conn.display     = dpy;
    conn.drawable_id = win;
                                    /* Open WS */
    popen_ws(WS, (Pconnid *) (&conn), phigs_ws_type_x_drawable);

    pset_edit_mode(PEDIT_INSERT);  /* Set edit mode to insert elements */
    pset_disp_upd_st(WS, PDEFER_WAIT, PMODE_NIVE);
                                    /* Set update state to WAIT */
}
/*
Make_squares
Make_squares defines a polyline square and a fillarea square in 2
```

```
dimensions. Structures POLYSQUARE and FILLSQUARE are defined to contain
these data elements along with color and style attributes to be applied to
the data elements. A higher level structure DISPLAY_STRUCT is defined to
include both the POLYSQUARE and FILLSQUARE structures and is then posted
to the open workstation to be displayed.
*/
Make_squares()
{

   static Ppoint     line_points[]=    /*  Define points for line drawing */
   { {  0.5, 0.5} , {  0.9, 0.5} ,
     {  0.9, 0.9} , {  0.5, 0.9} ,
     {  0.5, 0.5} };

   static Ppoint     fill_points[]=   /*  Define points for filled drawing */
   { {  0.1, 0.1} , {  0.5, 0.1} ,
     {  0.5, 0.5} , {  0.1, 0.5} };

   Ppoint_list Line_list, Fill_list;

   popen_struct(POLYSQUARE);        /* Open line drawing structure        */
     pset_line_colr_ind(2);        /* Assign default index color 2 to lines*/
     Line_list.num_points = 5;     /* Fill in number of points in list    */
     Line_list.points = line_points;/* Pointer to point array             */
     ppolyline(&Line_list);        /* Create a polyline element          */
   pclose_struct();                /* Close line drawing structure       */

   popen_struct(FILLSQUARE);        /* Open filled drawing structure      */
     pset_int_style(PSTYLE_SOLID);/* Set interior style to be solid      */
     pset_int_colr_ind(3);        /* Assign default index color 3 to lines*/
     Fill_list.num_points = 4;     /* Fill in number of points in list    */
     Fill_list.points = fill_points; /* Pointer to point array           */
     pfill_area(&Fill_list);       /* Create a fill area element         */
   pclose_struct();                /* Close filled drawing structure     */

   popen_struct( DISPLAY_STRUCT ); /* Open the top level display struct. */
     pexec_struct(POLYSQUARE);     /* Include the line drawn square      */
     pexec_struct(FILLSQUARE);     /* Include the filled square          */
   pclose_struct();

   ppost_struct(WS, DISPLAY_STRUCT, 1.0); /* Post DISPLAY_STRUCT, prio 1 */

   pupd_ws(WS, PUPD_PERFORM);              /* Update the workstation     */
}
```

example2.c

```
/*

            example2.c

This program defines and displays two boxes on the open  PEX workstation.
One box is a 3d polyline object and the other is a 3d fillarea object. This
program incorporates a PHIGS view representation. The view is set to
encompass -1.5 to 1.5 in model space coordinates with a parallel projection
matrix. This program shows command line arguments to handle Xlib
environment calls as well as an event loop. To exit this program press any
keyboard key.

       Author: James Buckmiller  May 1990.
       Modified:  J. Buckmiller  Mar 1991.  Approved C binding

       Copyright (C) 1990, Evans & Sutherland

*/
#include <X11/Xlib.h>
#include <X11/Xatom.h>
#include <X11/Xproto.h>
#include <X11/extensions/XInput.h>
#include <X11/phigs/phigs.h>       /* PHIGS extensions to X */
#include <X11/keysymdef.h>

#include "header2.h"               /* local includes */

Window myWin;
Display *dpy;
char *ProgramName;
Pint PEX_error;                    /* for PEX error number handling */

main(argc, argv)
int       argc;
char    *argv[];
{
   int i;
   char *geom = NULL;
   char *display = NULL;
   int winposx, winposy, winwidth, winheight;

   ProgramName = argv[0];

   winposx = 100;                  /* default window geometry */
```

```
winposy = 100;
winwidth  = 600;
winheight = 600;

 for (i=1; i < argc; i++)                    /* Parse the command line */
 {
      char *arg = argv[i];

      if (arg[0] == '-')
      {
         switch (arg[1])
          {
             case 'd':                              /* -display host:dpy */
                 if (++i >= argc) usage ();
                   display = argv[i];
                 continue;
             case 'g':                             /* -geometry host:dpy */
                 if (++i >= argc) usage ();
                   geom = argv[i];
                 continue;
             default:
                 usage ();
          }
      }
 }

 if (!(dpy = XOpenDisplay(display)))     /* Attempt to open the display */
   {
    perror("Cannot open display\n");
    exit(-1);
   }

 if (geom)
  {           /* Generate position and size from the geometry string */
    (void) XParseGeometry(geom, &winposx, &winposy, &winwidth,
         &winheight);
  }
                  /* Create a simple, unmapped input/output window */
 myWin = XCreateSimpleWindow(dpy, RootWindow(dpy, DefaultScreen(dpy)),
                   winposx, winposy, winwidth, winheight, 0,NULL,NULL);

                 /* Change the window name property */
 XChangeProperty(dpy, myWin, XA_WM_NAME, XA_STRING, 8,
             PropModeReplace, "Example 2:  Press any key to exit", 33);
```

```
                    /* Map the window for display */
    XMapWindow(dpy, myWin);

                    /* Begin PHIGS calls          */
    StartPhigs(dpy, myWin);

}


/*
usage

This routine prints out command line argument information if the user
supplied arguments are incorrect.
*/


usage ()
{
   fprintf (stderr, "usage:  %s [-options ...]\n\n", ProgramName);
   fprintf (stderr, "where options include:\n");
   fprintf (stderr, "    -display host:dpy        X server to use\n");
   fprintf (stderr, "    -geometry geom           geometry of window\n");
   fprintf (stderr, "\n");
   exit (1);
}


/*
Get_events

This routine is the event handler of X events that are generated
by the user. Only KeyPress and Expose events are handled at this time.
*/

Get_events(dpy)
Display  *dpy;
{
   XEvent report;
   int done = 0;

  /* setup the event mask to return keyboard and expose events */

   XSelectInput(dpy, myWin, KeyPressMask | ExposureMask );
```

```
  while(!done)                              /* Loop to get events          */
   {
     XNextEvent(dpy, &report);              /* Get the Event               */
     switch(report.type)
       {
       case KeyPress:                       /* Keypress on keyboard = exit */
         done = 1;
         break;
       case Expose:                         /* Expose events = redraw      */
         predraw_all_structs(1, PFLAG_ALWAYS);
         break;
       default:
         break;
       }
   }
}
```

motif2.c

```
/*
        m o t i f 2 . c

This program expands on motif1 by using a slightly more elaborate widget
hierarchy to display the drawing area and a push button.  The push button
allows the user to specify when the program should exit.  Otherwise, it is
the same as motif1.

   Author: Rich Thomson
   Date: Thursday, June 12th, 1990
   Modified: J. Buckmiller   Mar 1991.   Approved C binding

   Copyright (C) 1990, Evans & Sutherland Computer Corporation
*/

#include <X11/Xlib.h>
#include <X11/phigs/phigs.h>        /* PHIGS extensions to X */

#include <X11/Intrinsic.h>/* Toolkit intrinsics */
#include <Xm/Xm.h>/* Motif declarations */
#include <Xm/RowColumn.h>/* row column widget declarations */
#include <Xm/DrawingA.h>/* drawing area widget declarations */
#include <Xm/PushBG.h>/* push button gadget declarations */

#include "header2.h"

char *ProgramName;
Pint PEX_error;    /* for PEX error number handling */
Boolean done;      /* end flag */

/*
quit_CB

The callback procedure for the quit pushbutton widget.  It simply sets the
event processing exit flag to True, which will cause Get_events to stop
processing events.
*/
void quit_CB(quitButton, client_data, call_data)
     Widget quitButton;
     caddr_t client_data;
     XmAnyCallbackStruct *call_data;
{
  if (call_data->reason == XmCR_ACTIVATE)
```

```
    done = True;
}

/*
drawArea_CB

The callback procedure for the drawing area widget.  It redraws all the
structures on the workstation.
*/
void drawArea_CB(drawArea, client_data, call_data)
     Widget drawArea;
     caddr_t client_data;
     XmDrawingAreaCallbackStruct *call_data;
{
   if (call_data->reason == XmCR_EXPOSE)
      predraw_all_structs(WS, PFLAG_ALWAYS);
}

/*
main

The main routine creates the widget hierarchy for the program and then calls
StartPhigs. StartPhigs will then call Get_events to initiate event
processing.

The widget hierarchy used here is:

motif2 (class topLevelShell)
   |
   +-- rowcol (class RowColumn)
         |
         +--- drawArea (class DrawingArea)
         |
         +--- quit (class PushButtonGadget)

The row column widget is used for organizing its child widgets into a
columnar layout. The drawing area widget is used for PEX operations and the
PEX workstation is opened on its window. A push button is used to supply a
quit operation.

Any necessary resources for the widgets are specified here in the program,
which override any user defaults or command-line options.  Note that this
is not very friendly to the user who may want to change the font of the
push buttons. A friendlier way is to provide an application defaults file
which the user may override with user defaults or command-line arguments.
For simplicity, I have set the arguments here directly.
```

The display connection (dpy) and window ID (drawWindow) of the drawing area widget are available after the widget hierarchy has been realized.

BEWARE!! BEWARE!!

The drawing area widget in Motif 1.0 has a bug in that it ignores height and width resources supplied at creation time. A workaround I've found is to set the margins of the drawing area to be half the desired height and width. Since the margins specify the boundary between the drawing area widget's border and any children of the drawing area widget (we have none here), the drawing area widget will be sized to contain its children plus twice the margins in each direction. Hence to get a 600x600 drawing area widget, you can set the margins to 300. Other workarounds suggested involve creating the drawing area widget as a child of other widgets, but where there are no children of the drawing area, I prefer setting the margins.

```
*/
main(argc, argv)
int       argc;
char   *argv[];
{
   Arg args[10];
   register int n;
   Widget topLevel, rowColumn, quitButton, drawArea;
   XFontStruct *buttonFont;

   ProgramName = argv[0];
                  /* create topLevelShell */
   topLevel = XtInitialize(ProgramName, "Example", NULL, 0, &argc, argv);

   n = 0;         /* set the window title */
   XtSetArg(args[n], XmNtitle, "Example 2"); n++;
   XtSetValues(topLevel, args, n);

   rowColumn =    /* create row column for layout */
      XtCreateManagedWidget("rowcol", xmRowColumnWidgetClass, topLevel,
            NULL, 0);

   n = 0;         /* create drawing area for PEX */
   XtSetArg(args[n], XmNmarginWidth, 300); n++;
   XtSetArg(args[n], XmNmarginHeight, 300); n++;
   drawArea = XtCreateManagedWidget("drawArea", xmDrawingAreaWidgetClass,
               rowColumn, args, n);
                  /* add an expose callback */
   XtAddCallback(drawArea, XmNexposeCallback, drawArea_CB, NULL);
   n = 0;         /* create a quit button */
   XtSetArg(args[n], XmNlabelString,
```

```
      XmStringCreate("Click here to quit the program.",
           XmSTRING_DEFAULT_CHARSET)); n++;
   buttonFont =          /* look for Helvetica Bold font */
      XLoadQueryFont(XtDisplay(topLevel), "-*-Helvetica-Bold-R-Normal--
           14*");
   if (buttonFont)       /* if we found it, set fontList */
    {                    /* resource on the button */
      XtSetArg(args[n], XmNfontList,
   XmFontListCreate(buttonFont, XmSTRING_DEFAULT_CHARSET));
   n++;
   }
   quitButton =
      XtCreateManagedWidget("quit", xmPushButtonGadgetClass, rowColumn,
           args, n);
                         /* add an activation callback */
   XtAddCallback(quitButton, XmNactivateCallback, quit_CB, NULL);

   XtRealizeWidget(topLevel);/* realize widget hierarchy */

                         /* Begin PHIGS calls */
   StartPhigs(XtDisplay(drawArea), XtWindow(drawArea));
}

/*
Get_events

This routine handles event processing. It simply obtains events from the
toolkit via XtNextEvent and dispatches them to the widgets via
XtDispatchEvent. This continues until the boolean done becomes True. This
happens when the quit button's activation callback is invoked.
*/

Get_events(dpy)
Display  *dpy;
{
   XEvent event;

   done = False;

   do {
      XtNextEvent(&event);
      XtDispatchEvent(&event);
   } while (!done);
}
```

phigs2.c

```
/*
        phigs2.c

This file contains the PHIGS specific calls for example program 2.

       Author: James Buckmiller May 1990.
       Modified: J. Buckmiller Mar 1991.  Approved C binding

       Copyright (C) 1990, Evans & Sutherland

*/
#include <X11/Xlib.h>
#include <X11/phigs/phigs.h>

#include "header2.h"              /* Local includes */

/*
Error_Check

This routine checks the global variable used to store error codes returned
from PEX. If the error code is non-zero, it prints out a diagnostic message
and dies.
*/

void Error_Check(File, Line, Routine)
     char *File, *Routine;
     int Line;
{
  if (PEX_error)
    {
      fprintf(stderr, "(file %s; line %d):\n", File, Line);
      fprintf(stderr, "\t?unexpected PEX error %d in routine %s\n",
              PEX_error, Routine);
      exit(1);
    }
}
/*
StartPhigs

This routine is the top level routine that calls all supporting routines
in the logical order of a usual phigs routine i.e.,open PEX, setup the
workstation parameters, define the phigs structure and then go into the
event loop.
*/
```

```
StartPhigs(dpy, win)        /* Routine to start phigs calls */
Display *dpy;
Window win;
{
   OpenPex(dpy);                /* Open PEX                                */
   SetupWorkstation(dpy, win); /* Setup PHIGS workstation parameters    */
   Make_boxes();                /* Create Phigs structures                 */
   Get_events(dpy);             /* Event loop                              */
   Cleanup();                  /* Cleanup phigs structures close workstation */
}

/*
OpenPex

This routine Opens PEX on the display that was passed as an argument.
*/

OpenPex(dpy)
Display *dpy;
{

   Pxphigs_info  xinfo;
   unsigned long  infomask;

   xinfo.display = dpy;
   xinfo.rmdb    = NULL;
   xinfo.appl_id.name = NULL;
   xinfo.appl_id.class = NULL;
   xinfo.args.argc_p    = NULL;
   xinfo.args.argv      = NULL;
   xinfo.flags.no_monitor = 1;
   xinfo.flags.force_client_SS = 0;

   infomask = PXPHIGS_INFO_DISPLAY | PXPHIGS_INFO_FLAGS_NO_MON;

                        /* Open Pex */
   popen_xphigs((char*)NULL, PDEF_MEM_SIZE, infomask, &xinfo);

}

/*
SetupWorkstation

This routine opens a PHIGS workstation and sets up a PHIGS Viewport. The
structure edit mode is set to insert elements and the display update state
```

```
is set to PWAIT. The event mask for X input is set to select KeyPress and
Expose events.
*/

SetupWorkstation(dpy, win)          /* Open an Xwindow  workstation */
Display *dpy;
Window   win;
{
   Pconnid_x_drawable  conn;
   Pview_rep3          vrep;        /* Declare viewporting variables */
   Pview_map3          map;
   Ppoint3             vrp, cntr;
   Pvec3               vup;
   Pvec3               vpn;

   conn.display     = dpy;
   conn.drawable_id = win;

                                    /* Open WS */
   popen_ws(WS, (Pconnid *) (&conn), phigs_ws_type_x_drawable);

                                    /* Setup viewport parameters */

   map.proj_type = PTYPE_PARAL;                    /* Set projection type */
   map.vp.x_min = 0.0; map.vp.x_max = 1.0;         /* Set viewport limits */
   map.vp.y_min = 0.0; map.vp.y_max = 1.0;
   map.vp.z_min = 0.0; map.vp.z_max = 1.0;

   map.win.x_min= -1.5; map.win.x_max= 1.5;      /* Set window limits    */
   map.win.y_min= -1.5; map.win.y_max= 1.5;

   map.back_plane  = -2.0;    /* Set the front and back clipping planes */
   map.front_plane =  1.0;
   map.view_plane  =  0.0;    /* Set the location of the view plane */

   map.proj_ref_point.x =  1.0;
                       /* Set projection Reference point to be offset */
   map.proj_ref_point.y = -1.5;
                       /* from the VRP (eye) in VRC space to give an  */
   map.proj_ref_point.z =  3.0;/* angle view of the objects.*/

   vrep.xy_clip    = PIND_NO_CLIP;    /* Turn Viewport clipping off */
   vrep.back_clip  = PIND_NO_CLIP;  /* not to be confused with the  */
   vrep.front_clip = PIND_NO_CLIP;    /* clipping at the viewplanes!*/
```

```
    vrep.clip_limit = map.vp;   /* Set Viewport clipping volume = viewport */
                                    /* Setup View Reference Coordinates */

    vrp.x = 0.0; vrp.y = 0.0; vrp.z = 1.0;          /* Set View ref point   */
                                    /* Set view up vector    */
    vup.delta_x = 0.0; vup.delta_y = 1.0; vup.delta_z = 0.0;
                                    /* Set view plane normal*/
    vpn.delta_x = 0.0; vpn.delta_y = 0.0; vpn.delta_z = 1.0;

    peval_view_ori_matrix3(&vrp, &vpn, &vup,    /* Evaluate orient matrix */
                  &PEX_error, vrep.ori_matrix);

    SAFE_PEX("peval_view_ori_matrix3");         /* Check for error status */

    peval_view_map_matrix3( &map, &PEX_error,      /* Evaluate map matrix */
                              vrep.map_matrix);

    SAFE_PEX("peval_view_map_matrix3");         /* Check for error status */

    pset_view_rep3( WS, VIEW, &vrep );     /* Set the view representation  */


    pset_edit_mode(PEDIT_INSERT);     /* Set edit mode to insert elements */
               pset_disp_upd_st(WS, PDEFER_WAIT, PMODE_NIVE);
                                    /* Set update state to WAIT */
    pset_hlhsr_mode(1, PHIGS_HLHSR_MODE_ZBUFF); /* Enable WS Z buffering  */

}


/*
Make_boxes

Make_boxes  defines a polyline cube and a fillarea cube in 3
dimensions. Structures POLYBOX and FILLBOX are defined to contain
these data elements along with color and style attributes to be applied
to the data elements. A higher level structure DISPLAY_STRUCT is defined
to include both the POLYBOX and FILLBOX structures and is then
posted to the open workstation to be displayed. The workstation is then
updated, causing the objects to be displayed.
*/

Make_boxes()
{
```

```
                        /* Define polyline cube vectors */

static Ppoint3    line_points1[]=     /* Define points for front face */
{{  0.5, 0.5, 0.5} , { 1.0, 0.5, 0.5} ,
 {  1.0, 1.0, 0.5} , { 0.5, 1.0, 0.5} ,
 {  0.5, 0.5, 0.5}};

static Ppoint3    line_points2[]=      /* Define points for back face */
{{  0.5, 0.5, 0.0} , { 1.0, 0.5, 0.0} ,
 {  1.0, 1.0, 0.0} , { 0.5, 1.0, 0.0} ,
 {  0.5, 0.5, 0.0}};

static Ppoint3    line_points3[]=       /* Define connecting line     */
{{  0.5, 0.5, 0.5} , { 0.5, 0.5, 0.0}};

static Ppoint3    line_points4[]=       /* Define connecting line     */
{{  1.0, 0.5, 0.5} , { 1.0, 0.5, 0.0}};

static Ppoint3    line_points5[]=       /* Define connecting line     */
{{  1.0, 1.0, 0.5} , { 1.0, 1.0, 0.0}};

static Ppoint3    line_points6[]=       /* Define connecting line     */
{{  0.5, 1.0, 0.5} , { 0.5, 1.0, 0.0}};


                                /* Define solid cube faces */

static Ppoint3    fill_points1[]=     /* Define points for front face */
{{  -0.5, -0.5, 0.5} , {  0.0, -0.5, 0.5} ,
 {   0.0,  0.0, 0.5} , { -0.5,  0.0, 0.5}};

static Ppoint3    fill_points2[]=      /* Define points for back face */
{{  -0.5, -0.5, 0.0} , {  0.0, -0.5, 0.0} ,
 {   0.0,  0.0, 0.0} , { -0.5,  0.0, 0.0}};

static Ppoint3    fill_points3[]=     /* Define points for right face */
{{  0.0, -0.5, 0.5} , { 0.0, -0.5, 0.0} ,
 {  0.0,  0.0, 0.0} , { 0.0,  0.0, 0.5}};

static Ppoint3    fill_points4[]=      /* Define points for left face */
{{  -0.5, -0.5, 0.5} , { -0.5, -0.5, 0.0} ,
 {  -0.5,  0.0, 0.0} , { -0.5,  0.0, 0.5}};

static Ppoint3    fill_points5[]=    /* Define points for bottom face */
```

```
{{  -0.5, -0.5, 0.5} , {  0.0, -0.5, 0.5} ,
 {   0.0, -0.5, 0.0} , { -0.5, -0.5, 0.0}};

static Ppoint3    fill_points6[]=       /* Define points for top face */
{{  -0.5, 0.0, 0.5} , {  0.0, 0.0, 0.5} ,
 {   0.0, 0.0, 0.0} , { -0.5, 0.0, 0.0}};

Ppoint_list3 Line_list[5], Fill_list[5];


popen_struct(POLYBOX);          /* Open line drawing structure        */
   pset_line_colr_ind(2);       /* Assign default index color 2 to lines*/

   Line_list[0].num_points = 5; /* Fill in number of points in list   */
   Line_list[0].points = line_points1;   /* Pointer to point array   */
   ppolyline3(&Line_list[0]);   /* Create a polyline element          */

pset_line_colr_ind(3);          /* Assign default index color 3 to lines*/

   Line_list[1].num_points = 5;
   Line_list[1].points = line_points2;
   ppolyline3(&Line_list[1]);

pset_line_colr_ind(4);          /* Assign default index color 4 to lines*/
   Line_list[2].num_points = 2;
   Line_list[2].points = line_points3;
   ppolyline3(&Line_list[2]);

pset_line_colr_ind(5);          /* Assign default index color 5 to lines*/
   Line_list[3].num_points = 2;
   Line_list[3].points = line_points4;
   ppolyline3(&Line_list[3]);

pset_line_colr_ind(6);          /* Assign default index color 6 to lines*/
   Line_list[4].num_points = 2;
   Line_list[4].points = line_points5;
   ppolyline3(&Line_list[4]);

pset_line_colr_ind(7);          /* Assign default index color 7 to lines*/
   Line_list[5].num_points = 2;
   Line_list[5].points = line_points6;
   ppolyline3(&Line_list[5]);

pclose_struct();                /* Close line drawing structure       */
```

```
popen_struct(FILLBOX);          /* Open filled drawing structure       */
   pset_int_style(PSTYLE_SOLID); /* Set interior style to be solid     */
   pset_int_colr_ind(7);        /* Assign default index color 7 to face */

Fill_list[0].num_points = 4;     /* Fill in number of points in list  */
Fill_list[0].points = fill_points1; /* Pointer to point array         */
pfill_area3(&Fill_list[0]);      /* Create a fill area element        */

pset_int_colr_ind(3);          /* Assign default index color 3 to face */
   Fill_list[1].num_points = 4;
   Fill_list[1].points = fill_points2;
   pfill_area3(&Fill_list[1]);

pset_int_colr_ind(2);          /* Assign default index color 2 to face */
   Fill_list[2].num_points = 4;
   Fill_list[2].points = fill_points3;
   pfill_area3(&Fill_list[2]);

pset_int_colr_ind(4);          /* Assign default index color 4 to face */
   Fill_list[3].num_points = 4;
   Fill_list[3].points = fill_points4;
   pfill_area3(&Fill_list[3]);

pset_int_colr_ind(5);          /* Assign default index color 5 to face */
   Fill_list[4].num_points = 4;
   Fill_list[4].points = fill_points5;
   pfill_area3(&Fill_list[4]);

pset_int_colr_ind(6);          /* Assign default index color 6 to face */
   Fill_list[5].num_points = 4;
   Fill_list[5].points = fill_points6;
pfill_area3(&Fill_list[5]);

pclose_struct();                /* Close filled drawing structure      */

popen_struct(DISPLAY_STRUCT); /* Open the top level display structure */
   pset_view_ind(VIEW);         /* Set the view index to be used       */
   pset_hlhsr_id(PHIGS_HLHSR_ID_ON);      /* Turn on Z buffering       */
   pexec_struct(FILLBOX);      /* Include the filled square           */
   pexec_struct(POLYBOX);      /* Include the line drawn square       */
   pset_hlhsr_id(PHIGS_HLHSR_ID_OFF);     /* Turn off Z buffering      */
pclose_struct();                /* Close the structure                */

ppost_struct(WS, DISPLAY_STRUCT, 1.0); /* Post DISPLAY_STRUCT, prio 1 */
```

```
    pupd_ws(WS, PUPD_PERFORM);              /* Update the workstation     */

}


Cleanup()                               /* Cleanup routine when done    */
{
    punpost_all_structs(WS);             /* Unpost all structures on ws  */
    pdel_all_structs();                  /* Delete all structures */
    pclose_ws(WS);                       /* Close workstation     */
    pclose_phigs();                      /* Close PHIGS           */
}
```

header2.h

```
/*

        header2.h

This file contains header information for the example 2 programs.

       Author: James Buckmiller May 1990.

       Copyright (C) 1990, Evans & Sutherland

*/

#define WS              1
#define POLYBOX         1     /* define some constants to be used later */
#define FILLBOX         2
#define DISPLAY_STRUCT  3
#define VIEW            4
#define SAFE_PEX(routine) Error_Check(__FILE__, __LINE__, routine)

extern Window myWin;
extern Display *dpy;
extern char *ProgramName;

extern Pint PEX_error;/* for PEX error number handling */
```

example3.c

```
/*

        example3.c


This program defines and displays two boxes on the open PEX workstation.
One box is a 3d polyline object and the other is a 3d fillarea object. This
program incorperates a PHIGS view representation. The view is set to
encompass -1.5 to 1.5 in model space coordinates with a parallel projection
matrix. This program incorporates dials input to update the transformation
matrices that are in the display structure. This program shows command line
arguments to handle Xlib environment calls as well as an event loop. To
reset the picture press the r key on the keyboard. To exit the client press
the e key on the keyboard.


      Author: James Buckmiller  May 1990.
      Modified: J. Buckmiller Mar 1991.   Approved C binding

      Copyright (C) 1990, Evans & Sutherland

*/
#include <X11/Xlib.h>
#include <X11/Xproto.h>
#include <X11/Xatom.h>
#include <X11/extensions/XInput.h>
#include <X11/phigs/phigs.h>
#include <X11/keysymdef.h>

#include "header3.h"                    /* local includes */

Window myWin;                           /* drawing window */
Display *dpy;                           /* X11 display connection */
char *ProgramName;
Pint PEX_error;                         /* For PEX error numbers */


main(argc, argv)
int       argc;
char   *argv[];
{
   int i;
   char *geom = NULL;
   char *display = NULL;
```

```
int winposx, winposy, winwidth, winheight;

ProgramName = argv[0];

winposx = 100;                                  /* default window geometry */
winposy = 100;
winwidth  = 600;
winheight = 600;

for (i=1; i < argc; i++)                    /* Parse the command line */
{
     char *arg = argv[i];

     if (arg[0] == '-')
      {
         switch (arg[1])
          {
             case 'd':                              /* -display host:dpy */
                 if (++i >= argc) usage ();
                   display = argv[i];
                 continue;
             case 'g':                              /* -geometry host:dpy */
                 if (++i >= argc) usage ();
                   geom = argv[i];
                 continue;
             default:
                 usage ();
          }
      }
 }

if (!(dpy = XOpenDisplay(display))) /* Attempt to open the display */
{
   perror("Cannot open display\n");
   exit(-1);
}

if (geom)
{        /* Generate position and size from the geometry string */
(void) XParseGeometry(geom, &winposx, &winposy, &winwidth, &winheight);
  }
              /* Create a simple, unmapped input/output window */
myWin = XCreateSimpleWindow(dpy, RootWindow(dpy, DefaultScreen(dpy)),
                    winposx, winposy, winwidth, winheight, 0,NULL,NULL);
```

```
                /* Change the window name property */
    XChangeProperty(dpy, myWin, XA_WM_NAME, XA_STRING, 8,
            PropModeReplace, "Example 3:   r Key = Reset  e Key = Exit", 40);

                /* Map the window for display */
    XMapWindow(dpy, myWin);

                /* Begin PHIGS calls          */
    StartPhigs(dpy, myWin);

}

/*
usage

This routine prints out command line argument information if the user
supplied arguments are incorrect.
*/

usage ()
{
    fprintf (stderr, "usage:  %s [-options ...]\n\n", ProgramName);
    fprintf (stderr, "where options include:\n");
    fprintf (stderr, "    -display host:dpy        X server to use\n");
    fprintf (stderr, "    -geometry geom           geometry of window\n");
    fprintf (stderr, "\n");
    exit (1);
}

/*
Get_events

This routine is the event handling routine. First the usual X
events are trapped. If an expose event occurs then a PHIGS
redrawallstructures is called. If a Keyboard event occurs
the keysym is reviewed to see if it is an "e" for exit or an
"r" for a reset of the display. If the event that is generated
is not a usual X event then it is check to be an extension
event for dials events. If dials event then the object is
transformed by rotates and translates in X,Y,Z and scale.
*/

Get_events(dpy)
Display  *dpy;
{
```

```
XDeviceInfo *devices = NULL;
XDevice *dials = NULL;
XEventClass DeviceMotionClass[100];
XID dials_id;
XEvent report;

XKeyPressedEvent *pev;
KeySym key;
char buf[20];

unsigned long event_mask;
XButtonPressedEvent *bdown;
XButtonReleasedEvent *bup;
Atom dials_atom = 0;
int knob_totals[MAXDIALS];
XStringFeedbackControl strfc;
int done=0;
KeySym ledstring[MAXDIALS][CHARS_PER_DIAL], blankled[CHARS_PER_DIAL];

Pmatrix3 composite;
static Pmatrix3 currmatrix[] =
        {{1,0,0,0},{0,1,0,0},{0,0,1,0},{0,0,0,1}};

Pvec3 scale, trans;
int ndevices = 0, i, j, EventCount = 0, DeviceMotion = -1;
int rotx=0, roty=0, rotz=0;
Ppoint3 cntr;

                                                /* Dial Labels */
static char textstring[MAXDIALS][CHARS_PER_DIAL] =
                                { " X ROT  ", " Y ROT  ", " Z ROT  ",
                                  " SCALE  ", " X TRAN ", " Y TRAN ",
                                  " Z TRAN ", "        " };

cntr.x = 0.0;               /* Define center point of transformations */
cntr.y = 0.0;
cntr.z = 0.0;
scale.delta_x = 1.0;         /* Define initial scale to be 1          */
scale.delta_y = 1.0;
scale.delta_z = 1.0;
trans.delta_x = 0.0;         /* Define initial translation to be 0    */
trans.delta_y = 0.0;
trans.delta_z = 0.0;
```

```
                                /* Get the atom ID for the Knob box */
   dials_atom = XInternAtom(dpy, "KNOB_BOX", True);
                                /* Get list of Input devices        */
   devices = XListInputDevices(dpy, &ndevices);
               /* Find the dials device in the list and open the device */

   for (i = 0; i < ndevices; i++, devices++)
      if ((devices->type == dials_atom) &&
            (devices->use == IsXExtensionDevice))
         {
         dials_id = devices->id;
         dials = XOpenDevice(dpy, dials_id); /* Get Xdevice structure */
         break;                              /* for the dials         */
         }

if (!dials)
   {
     fprintf(stderr, "?dials box not found in X extension device list.\n");
   exit(1);
   }

                                  /* Get event class values for dials     */
DeviceMotionNotify(dials, DeviceMotion, DeviceMotionClass[EventCount]);
EventCount++;

                               /* Tell server to pass on Extension events */
XSelectExtensionEvent(dpy, myWin, DeviceMotionClass, EventCount);
                                  /* Set the event mask for the window    */
XSelectInput(dpy, myWin, EnterWindowMask | LeaveWindowMask
                 | KeyPressMask | ExposureMask);


for (i = 0; i < MAXDIALS; i++)          /* Load the keysym arrays */
   for (j = 0; j < CHARS_PER_DIAL; j++)
      {
      ledstring[i][j] = (KeySym) textstring[i][j];   /* Dial labels  */
      blankled[j] = SPACEKEYSYM;                     /* Blank labels */
      }

strfc.class = StringFeedbackClass;
strfc.length = sizeof(XStringFeedbackControl);
strfc.num_keysyms = CHARS_PER_DIAL;

for (i=0; i<MAXDIALS; i++)              /* Set the dial labels */
```

```
    {
    strfc.id = i;
    strfc.syms_to_display = ledstring[i];
    XChangeFeedbackControl(dpy, dials, DvString, &strfc);
    }

while (!done)
{
    XNextEvent(dpy, &report);       /* Get next event from event queue */
    if (report.type < LASTEvent)    /* Check if extension event or not */
    {
       switch (report.type)
       {
       case EnterNotify:               /* Turn on the dial labels */
          for (i=0; i<MAXDIALS; i++)  /* when the cursor enters  */
          {                          /* the window.             */
          strfc.id = i;
          strfc.syms_to_display = ledstring[i];
          XChangeFeedbackControl(dpy, dials, DvString, &strfc);
          }
          break;

       case KeyPress:                          /* Trap keyboard events   */
                                               /* and perform function.  */
          pev = (XKeyPressedEvent *) &report;
          XLookupString(pev, buf, sizeof(buf), &key, NULL);
          if(buf[0] == 'r')      /* reset picture */
             {
             make_identity(currmatrix); /* Currmatrix = identity matrix */
             Make_boxes();                     /* Rebuild the structures*/
                                       /* Reset xformation parameters */
             scale.delta_x = scale.delta_y = scale.delta_z = 1.0;
             trans.delta_x = trans.delta_y = trans.delta_z = 0.0;
             rotx = roty = rotz = 0;
             }
          if(buf[0] == 'e')      /* Exit  application */
             done = 1;
          break;

       case Expose:                          /* expose events = redraw */
             predraw_all_structs(WS, PFLAG_ALWAYS);
          break;

       case LeaveNotify:         /* Blank LEDs when cursor leaves window */
```

```
        for (i=0; i<MAXDIALS; i++)
        {
           strfc.id = i;
           strfc.syms_to_display = blankled;
           XChangeFeedbackControl(dpy, dials, DvString, &strfc);
        }
        break;

     }
  }

                        /* else it's an extension event */
else if (report.type == DeviceMotion)          /* Dials input */
  {
  XDeviceMotionEvent *dm = (XDeviceMotionEvent *) &report;
  for (i=0; i<MAXDIALS; i++)       /* Initialize knob values array */
     knob_totals[i] = 0;

/*-------------------------------------------------------------------------
The following piece of code goes out to the event queue and scoops
off all dial motion events that are found on the queue with the
XCheckTypedEvent call.These events are then accumulated for each
axis and then processed with the accumulated values.
The reason for doing this is to increase the performance of the
system. If an update of the workstation display is performed
for every dial event that occurs the display will get behind thus
causing a lag time between when the dials stop sending events and
the system finishes unpiling the event queue.
For this application grabbing all dial events off the queue works
well however, one must beware that if an application allows the
dials to be redefined with some other event (function key or pick
menu) this method may not be the way to get the dial events since
there may be keypress or pick events intermixed with the dials
events. To get around this problem one may wish to use the
XPeekEvent routine to look ahead one event to be sure that it is
the same event class as the ones being accumulated.
-------------------------------------------------------------------------*/

  do                      /* Collapse the events before processing */
     for (i=0; i < dm->axes_count; i++)
        knob_totals [dm->first_axis+i] += dm->axis_data[i];
                        /* Gather all dial events from the event queue */
  while (XCheckTypedEvent (dpy, DeviceMotion, dm));

                        /* Process DeviceMotion events  */
```

```
if (knob_totals[0])          /* dial 1 input check rot x */
   rotx += knob_totals[0];

if (knob_totals[1])          /* Dial 2 input check rot y */
   roty += knob_totals[1];

if (knob_totals[2])          /* Dial 3 input check rot z */
   rotz += knob_totals[2];

if (knob_totals[3])          /* Dial 4 input check uniform scale */
   {
      scale.delta_x += knob_totals[3] * DIALSCALE;
      scale.delta_y += knob_totals[3] * DIALSCALE;
      scale.delta_z += knob_totals[3] * DIALSCALE;
   }

if (knob_totals[4])          /* Dial 5 input check trans x */
   trans.delta_x += knob_totals[4] * DIALSCALE;

if (knob_totals[5])          /* Dial 6 input check trans y */
   trans.delta_y += knob_totals[5] * DIALSCALE;

if (knob_totals[6])          /* Dial 7 input check trans z */
   trans.delta_z += knob_totals[6] * DIALSCALE;

                                  /* Build the transformation matrix */
pbuild_tran_matrix3(&cntr, &trans, DEG_TO_RAD(rotx), DEG_TO_RAD(roty)
               ,DEG_TO_RAD(rotz), &scale, &PEX_error
               ,currmatrix);
SAFE_PEX("pbuild_tran_matrix3");   /* Check for error status */
   popen_struct(DISPLAY_STRUCT);    /* Open structure for editing */
   {
      pset_elem_ptr(0);               /* Reset element pointer      */
      pset_elem_ptr_label(TRANS);     /* Find transformation label  */
      poffset_elem_ptr(1);           /* Point at matrix            */
                        /* replace matrix */
      pset_local_tran3(currmatrix, PTYPE_PRECONCAT);
           }
   pclose_struct();                   /* Close structure     */
   predraw_all_structs(WS, PFLAG_ALWAYS); /* Redraw the structure */

  }
 }
}
```

motif3.c

```
/*
        m o t i f 3 . c

This program expands on motif2 to display true 3D objects and handling of
events from the dials box. The objects displayed are a 3D solid cube
created using FILL AREA 3 structure elements and a 3D wire-frame cube
created using POLYLINE 3 structure elements. This program incorporates a
3D PHIGS view representation to view the objects. The view implements a
parallel projection whose view volume extends from -1.0 to 1.5 on all
three axes of model space.

The dials box controls viewing of the objects via a transformation matrix
in the structure, allowing an arbitrary translation, rotation and scale
about the origin. Setting the labels on the dials box only when the
core pointer is inside the drawing window aids in feedback to the user
as to when dial events are processed.

In addition to the widgets in motif2, this program adds a reset button to
reset the viewing transformation matrix to its original state (useful
when you've lost the object due to translations and/or scaling).

   Author: Rich Thomson
   Date: Thursday, June 12th, 1990
   Modified: J. Buckmiller Mar 1991.   Approved C binding

   Copyright (C) 1990, Evans & Sutherland Computer Corporation
*/

#include <X11/Xlib.h>
#include <X11/Xatom.h>
#include <X11/extensions/XInput.h>
#include <X11/phigs/phigs.h>

#include <X11/Intrinsic.h>/* toolkit intrinsics */
#include <Xm/RowColumn.h>/* row column widget */
#include <Xm/DrawingA.h>/* drawing area widget */
#include <Xm/PushBG.h>/* push button gadget */

#include "header3.h"

Display *dpy;/* X11 display connection */
char *ProgramName;
int DeviceMotion;/* device motion event type */
Window drawWindow;/* drawing window */
```

```
Pint PEX_error;                          /* For PEX error numbers */

static XDevice *dials = NULL;      /* dials device */
static Pmatrix3 currmatrix = {{1,0,0,0},{0,1,0,0},{0,0,1,0},{0,0,0,1}};
static Boolean done = False;
static Pvec3 scale = { 1.0, 1.0, 1.0 };   /* scale factors */
static Pvec3 trans = { 0.0, 0.0, 0.0 };   /* translation vector */
static Pint rotx = 0, roty = 0, rotz = 0; /* axis rotation amounts */

/*
quit_CB

The callback procedure for the quit pushbutton widget. It simply sets the
event processing exit flag to True, which will cause Get_events to
stop processing events.
*/
void quit_CB(quitButton, client_data, call_data)
     Widget quitButton;
     caddr_t client_data;
     XmAnyCallbackStruct *call_data;
{
  if (call_data->reason == XmCR_ACTIVATE)
    done = True;
}

/*
reset_CB

The callback procedure for the reset button.  It re-initializes the
parameters that define the transformation matrix corresponding to
translate, rotate and scale operations and recreates the initially posted
structures.
*/

void reset_CB(resetButton, client_data, call_data)
     Widget resetButton;
     caddr_t client_data;
     XmAnyCallbackStruct *call_data;
{
  if (call_data->reason == XmCR_ACTIVATE)
    {
      Make_boxes();              /* recreate initial structures */
```

```
                                /* reset xformation parameters */
        scale.delta_x = scale.delta_y = scale.delta_z = 1.0;
        trans.delta_x = trans.delta_y = trans.delta_z = 0.0;
        rotx = roty = rotz = 0;
      }
}

/*
drawArea_CB

The callback procedure for the drawing area widget. It redraws all
the structures on the workstation.
*/
void drawArea_CB(drawArea, client_data, call_data)
      Widget drawArea;
      caddr_t client_data;
      XmDrawingAreaCallbackStruct *call_data;
{
  if (call_data->reason == XmCR_EXPOSE)
    predraw_all_structs(WS, PFLAG_ALWAYS);
}

/*
knob_labels

This array holds the KeySym's that contain the knob labels. It is used by
the enter and leave window handlers to blank out the labels when the
pointer is not in the drawing area window.

Dial labels are CHARS_PER_DIAL KeySyms per dial. Ascii characters can be
converted to KeySyms by C type casting.
*/

static KeySym knob_labels[MAXDIALS][CHARS_PER_DIAL];

/*
enter_handler

This event handler restores the knob labels to our labels when the
pointer moves into the drawing window. Conditionally labelling the dials
in this way gives extra feedback to the user that the dials are active
only when the mouse is inside the appropriate window.
*/
```

```
void enter_handler(widget, client_data, event, continue_to_dispatch)
     Widget widget;
     caddr_t client_data;
     XEvent *event;
     Boolean *continue_to_dispatch;
{
  register int i;
  XStringFeedbackControl strfc;

  strfc.class = StringFeedbackClass;/* initialize the feedback struct */
  strfc.length = sizeof(XStringFeedbackControl);
  strfc.num_keysyms = CHARS_PER_DIAL;

  for (i = 0; i < MAXDIALS; i++)/* for each dial */
    {
      strfc.id = i;/* number of dial to set feedback on */
      strfc.syms_to_display = knob_labels[i];
            /* keysyms containing feedback */
      XChangeFeedbackControl(dpy, dials, DvString, &strfc);
    }
}

/*
leave_handler

This event handler blanks the knob labels when the pointer leaves the
drawing window.
*/
void leave_handler(widget, client_data, event, continue_to_dispatch)
     Widget widget;
     caddr_t client_data;
     XEvent *event;
     Boolean *continue_to_dispatch;
{
   static XStringFeedbackControl strfc;
   static KeySym blanks[CHARS_PER_DIAL];
   static Boolean initialized = False;
   register int i;

  if (!initialized)/* initialize variables the first */
                  /* time we're called  */
    {
      for (i = 0; i < CHARS_PER_DIAL; i++)
   blanks[i] = (KeySym) ' ';    /* prepare a blank keysym array */
```

```
        strfc.class = StringFeedbackClass;
              /* initialize the feedback struct */
        strfc.length = sizeof(XStringFeedbackControl);
        strfc.num_keysyms = CHARS_PER_DIAL;
        strfc.syms_to_display = blanks;

        initialized = True;          /* remember we've been initialized */
      }

    for (i = 0; i < MAXDIALS; i++)/* for each dial */
      {
        strfc.id = i;                /* indicate which dial to change */
                                     /* change it */
        XChangeFeedbackControl(dpy, dials, DvString, &strfc);
      }
}

/*
open_knob

This routine opens the knob box on the given display and selects device
motion extension events on the given window. Extension events are
selected by first invoking the appropriate macro on an XEventClass
structure (in this case DeviceMotionNotify) and then calling
XSelectExtensionEvent.
*/

void open_knob()
{
                    /* dial labels as ascii strings */
    static char textstring[MAXDIALS][CHARS_PER_DIAL] = {
      " X ROT  ", " Y ROT  ", " Z ROT  "," SCALE  ",
     " X TRAN ", " Y TRAN ", " Z TRAN ", "        " /* eighth label is blank */
      };
    register int knob, i;
    int ndevices;/* number of extension devices */
    XDeviceInfo *devices = NULL;/* extension device info list */
    Atom dials_atom = XInternAtom(dpy, "KNOB_BOX", True);
                    /* intern device name into an atom */
    XID dials_id; /* device ID for dials box */
    XEventClass eventClass[1];

    devices = XListInputDevices(dpy, &ndevices);  /* get list of devices */
```

```
  for (i = 0; i < ndevices; i++, devices++)
    if ((devices->type == dials_atom) && (devices->use ==
            IsXExtensionDevice))
      {           /* did we find the dial box? */
     dials_id = devices->id;/* yes, remember its device ID */
     dials = XOpenDevice(dpy, dials_id);     /* and open it */
     break;                          /* we only want the first one... */
      }

  if (!dials)/* couldn't open or find dials */
    {
      fprintf(stderr, "?couldn't open dials box.\n");
      exit(1);
    }
                  /* select device motion events */
  DeviceMotionNotify(dials, DeviceMotion, eventClass[0]);
  XSelectExtensionEvent(dpy, drawWindow, eventClass, 1);

  for (knob = 0; knob < MAXDIALS; knob++)
            /* convert ascii labels to KeySyms */
    for (i = 0; i < CHARS_PER_DIAL; i++)
      knob_labels[knob][i] = (KeySym) textstring[knob][i];
}

/*
main
```

The main routine creates the widget hierarchy for the program, opens the knob box and then calls StartPhigs. StartPhigs will then call Get_events to initiate event processing.
The widget hierarchy used here is:

```
  motif3 (class topLevelShell)
    |
    +-- rowcol (class RowColumn)
          |
          +--- drawArea (class DrawingArea)
          |
          +--- reset (class PushButtonGadget)
          |
          +--- quit (class PushButtonGadget)
```

The row column widget is used for organizing its child widgets into a columnar layout. The drawing area widget is used for PEX operations and the PEX workstation is opened on its window. The two push buttons are used to supply reset and quit operations.

Any necessary resources for the widgets are specified here in the program, which override any user defaults or command-line options. Note that this is not very friendly to the user who may want to change the font of the push buttons. A friendlier way is to provide an application defaults file which the user may override with user defaults or command-line arguments. For simplicity, I have set the arguments here directly.

The display connection (dpy) and window ID (drawWindow) of the drawing area widget are available after the widget hierarchy has been realized.

BEWARE!! BEWARE!!
The drawing area widget in Motif 1.0 has a bug in that it ignores height and width resources supplied at creation time. A workaround I've found is to set the margins of the drawing area to be half the desired height and width. Since the margins specify the boundary between the drawing area widget's border and any children of the drawing area widget (we have none here), the drawing area widget will be sized to contain its children plus twice the margins in each direction. Hence to get a 600x600 drawing area widget, you can set the margins to 300. Other workarounds suggested involve creating the drawing area widget as a child of other widgets, but where there are no children of the drawing area, I prefer setting the margins.

*/

```
main(argc, argv)
int       argc;
char   *argv[];
{
  Arg args[10];/* arg. array for widget resources */
  register int n;/* resource count */
  Widget topLevel, rowColumn, quitButton, resetButton, drawArea;
  XFontStruct *buttonFont;/* font obtained from XLoadQueryFont */
  XmFontList fontList;/* for setting widget's fontList */

  ProgramName = argv[0];
                /* create topLevelShell */
  topLevel = XtInitialize(ProgramName, "Example", NULL, 0, &argc, argv);

  n = 0;
  XtSetArg(args[n], XmNtitle, "Example 3"); n++;
  XtSetValues(topLevel, args, n);

  buttonFont =/* find the font we want for buttons */
    XLoadQueryFont(XtDisplay(topLevel), "-*-Helvetica-Bold-R-Normal--
          14*");
```

```
if (buttonFont)
  fontList = XmFontListCreate(buttonFont, XmSTRING_DEFAULT_CHARSET);

rowColumn =/* create the row column for layout */
  XtCreateManagedWidget("rowcol", xmRowColumnWidgetClass, topLevel,
          NULL, 0);

n = 0;     /* create the drawing area for PEX */
XtSetArg(args[n], XmNmarginWidth, 300); n++;    /* size appropriately */
XtSetArg(args[n], XmNmarginHeight, 300); n++;
drawArea = XtCreateManagedWidget("drawArea", xmDrawingAreaWidgetClass,
              rowColumn, args, n);
                 /* add an expose callback */
XtAddCallback(drawArea, XmNexposeCallback, drawArea_CB, NULL);

                 /* add event handlers to handle */
                 /* blanking of knob labels */
XtAddEventHandler(drawArea, EnterWindowMask, False, enter_handler,
          NULL);
XtAddEventHandler(drawArea, LeaveWindowMask, False, leave_handler,
          NULL);

n = 0;                  /* create the reset button */
XtSetArg(args[n], XmNlabelString,
    XmStringCreate("Click here to reset the picture.",
         XmSTRING_DEFAULT_CHARSET)); n++;
if (buttonFont)
  {
    XtSetArg(args[n], XmNfontList, fontList); n++;
  }
resetButton = XtCreateManagedWidget("reset", xmPushButtonGadgetClass,
                rowColumn, args, n);
                      /* add it's activation callback */
XtAddCallback(resetButton, XmNactivateCallback, reset_CB, NULL);

n = 0;     /* create the quit button */
XtSetArg(args[n], XmNalignment, XmALIGNMENT_CENTER);
XtSetArg(args[n], XmNlabelString,
    XmStringCreate("Click here to quit the program.",
         XmSTRING_DEFAULT_CHARSET)); n++;
if (buttonFont)
  {
    XtSetArg(args[n], XmNfontList, fontList); n++;
  }
quitButton =
```

```
    XtCreateManagedWidget("quit", xmPushButtonGadgetClass, rowColumn,
            args, n);
                /* add it's activation callback */
  XtAddCallback(quitButton, XmNactivateCallback, quit_CB, NULL);

  XtRealizeWidget(topLevel);/* realize widget hierarchy */

  dpy = XtDisplay(drawArea);/* obtain the display connection */
  drawWindow = XtWindow(drawArea);  /* and the drawing area's window ID */

  open_knob(dpy);/* open the knob box */

  StartPhigs(dpy, drawWindow);/* Begin PHIGS calls */
}

/* Get_events

This routine processes events requested by the program. XtNextEvent
obtains the next event from the input queue and places it in report. The
type of the event is then examined to determine if it is an extension
event or a regular X event. The constant LASTEvent (defined in X.h) is
bigger than the event type of any X event and can be used to
differentiate extension events from normal X events.

Regular events are handled by the toolkit dispatch mechanism via
XtDispatchEvent. Extension events (DeviceMotion) are handled on a
case-by-case basis.

When a DeviceMotion event is encountered, all device motion events are
removed from the event queue and accumulated into knob_totals, since the
dials box reports relative changes. Event explosion is a very real
possibility since every device motion event requires 2 XEvent structures
(only 6 axes' worth of data fit in a single XEvent) and the sample rate
of the dials box is high. Since this program is only concerned with
cumulative changes in the dials values, it is safe to condense the device
motion events via XCheckTypedEvent. Since XCheckTypedEvent can remove
events that are not at the head of the event queue, it may not be
appropriate for situations where the semantics of a device motion event
can be changed by another event (for instance, a key or button press).

*/

Get_events(dpy)
     Display  *dpy;

{
  XEvent report;
```

```
int i;
static Ppoint3 cntr = { 0.0, 0.0, 0.0 }; /* center at origin */

pset_edit_mode(PEDIT_REPLACE);   /* Set edit mode to replace elements */

while (!done)
  {
    XtNextEvent(&report);

    if (report.type < LASTEvent)
 XtDispatchEvent(&report);
  else if (report.type == DeviceMotion) /* must be device motion event */
 {
  XDeviceMotionEvent *dm = (XDeviceMotionEvent *) &report;
  int knob_totals[8];

  for (i = 0; i < MAXDIALS; i++)
        /* Initialize knob values array */
    knob_totals[i] = 0;

  do            /* Compress motion events */
    for (i = 0; i < dm->axes_count; i++)
      knob_totals [dm->first_axis+i] += dm->axis_data[i];
  while (XCheckTypedEvent(dpy, DeviceMotion, dm));

                /* Process device motion events */
  if (knob_totals[0])           /* dial 1 input check rot x */
    rotx += knob_totals[0];

  if (knob_totals[1])           /* Dial 2 input check rot y */
    roty += knob_totals[1];

  if (knob_totals[2])           /* Dial 3 input check rot z */
    rotz += knob_totals[2];

  if (knob_totals[3])       /* Dial 4 input check uniform scale */
    {
      scale.delta_x += knob_totals[3] * DIALSCALE;
      scale.delta_y += knob_totals[3] * DIALSCALE;
      scale.delta_z += knob_totals[3] * DIALSCALE;
    }

  if (knob_totals[4])           /* Dial 5 input check trans x */
    trans.delta_x += knob_totals[4] * DIALSCALE;
```

```
    if (knob_totals[5])          /* Dial 6 input check trans y */
      trans.delta_y += knob_totals[5] * DIALSCALE;

    if (knob_totals[6])          /* Dial 7 input check trans z */
      trans.delta_z += knob_totals[6] * DIALSCALE;

              /* Build the transformation matrix */
        pbuild_tran_matrix3(&cntr, &trans, DEG_TO_RAD(rotx),
          DEG_TO_RAD(roty)
                          ,DEG_TO_RAD(rotz), &scale, &PEX_error
                          ,currmatrix);
        SAFE_PEX("pbuild_tran_matrix3");    /* Check for error status */

              /* Combine old and new matrices */

        popen_struct(DISPLAY_STRUCT);    /* Open structure for editing */
    {
          pset_elem_ptr(0);            /* Reset element pointer       */
          pset_elem_ptr_label(TRANS);   /* Find transformation label  */
          poffset_elem_ptr(1);         /* Point at matrix             */
                          /* replace matrix */
          pset_local_tran3(currmatrix, PTYPE_PRECONCAT);
     }
       pclose_struct();                  /* Close structure           */
      predraw_all_structs(WS, PFLAG_ALWAYS); /* Redraw the structure   */

      }
    }
}
```

phigs3.c

```
/*
   phigs3.c

This program contains the phigs specific setup for example program 3.

   Author: James Buckmiller May 1990
   Modified: J. Buckmiller Mar 1991     Approved C binding

   Copyright (C) 1990, Evans & Sutherland

*/

#include <X11/Xlib.h>
#include <X11/phigs/phigs.h>

#include "header3.h"         /* local includes */



/*
Error_Check

This routine checks the global variable used to store error codes
returned from PEX. If the error code is non-zero, it prints out a
diagnostic message and dies.
*/

void Error_Check(File, Line, Routine)
     char *File, *Routine;
     int Line;
{
  if (PEX_error)
    {
      fprintf(stderr, "(file %s; line %d):\n", File, Line);
      fprintf(stderr, "\t?unexpected PEX error %d in routine %s\n",
              PEX_error, Routine);
      exit(1);
    }
}
```

```
/*
StartPhigs

This routine is the top level routine that calls all supporting
routines in the logical order of a usual phigs routine ie
open PEX, setup the workstation parameters, define the phigs
structure and then go into the event loop.
*/

StartPhigs(dpy, win)        /* Routine to start phigs calls */
Display *dpy;
Window win;
{
   OpenPex(dpy);              /* Open PEX                                    */
   SetupWorkstation(dpy, win);/* Setup PHIGS workstation parameters          */
   Make_boxes();              /* Create Phigs structures                     */
   Get_events(dpy);           /* Event loop                                  */
   Cleanup();               /* Cleanup phigs structures close workstation */
}


/*
OpenPex

This routine Opens PEX on the display that was passed as an argument.
*/

OpenPex(dpy)
Display *dpy;
{
  Pxphigs_info  xinfo;
  unsigned long  infomask;

  xinfo.display = dpy;
  xinfo.rmdb    = NULL;
  xinfo.appl_id.name = NULL;
  xinfo.appl_id.class = NULL;
  xinfo.args.argc_p    = NULL;
  xinfo.args.argv      = NULL;
  xinfo.flags.no_monitor = 1;
  xinfo.flags.force_client_SS = 0;

  infomask = PXPHIGS_INFO_DISPLAY | PXPHIGS_INFO_FLAGS_NO_MON;
```

```
  /* Open Pex */
  popen_xphigs((char*)NULL, PDEF_MEM_SIZE, infomask, &xinfo);

}



/*
SetupWorkstation

This routine opens a PHIGS workstation and sets up a Viewport.
Z buffering is enabled by calling psethlhsrmode. The structure
edit mode is set to insert elements and the display update state
is set to PWAIT.
*/

SetupWorkstation(dpy, win)
Display *dpy;
Window   win;
{
  Pconnid_x_drawable  conn;
  Pview_rep3          vrep;              /* Declare vieporting variables */
  Pview_map3          map;
  Ppoint3             vrp, cntr;
  Pvec3               vup;
  Pvec3               vpn;

  conn.display     = dpy;
  conn.drawable_id = win;

  popen_ws(WS, (Pconnid *) (&conn), phigs_ws_type_x_drawable);/* Open WS */

  /* Setup viewport parameters */

   map.proj_type = PTYPE_PARAL;                    /* Set projection type */
   map.vp.x_min = 0.0; map.vp.x_max = 1.0;         /* Set viewport limits */
   map.vp.y_min = 0.0; map.vp.y_max = 1.0;
   map.vp.z_min = 0.0; map.vp.z_max = 1.0;

   map.win.x_min= -1.5; map.win.x_max= 1.5;      /* Set window limits    */
   map.win.y_min= -1.5; map.win.y_max= 1.5;

   map.back_plane  = -2.0;    /* Set the front and back clipping planes */
   map.front_plane =  1.0;
   map.view_plane  =  0.0;    /* Set the location of the view plane     */
```

```
    map.proj_ref_point.x = 0.0;          /* Set projection Reference point */
    map.proj_ref_point.y = 0.0;           /* in VRC space                  */
    map.proj_ref_point.z = 3.0;

    vrep.xy_clip    = PIND_NO_CLIP;  /* Turn Viewport clipping off         */
    vrep.back_clip  = PIND_NO_CLIP; /* not to be confused with the         */
    vrep.front_clip = PIND_NO_CLIP;  /* clipping at the viewplanes!        */

   vrep.clip_limit = map.vp; /* Set Viewport clipping volume = viewport */

   /* Setup View Reference Coordinates */

   vrp.x = 0.0; vrp.y = 0.0; vrp.z = 1.0;     /* Set View ref point        */
   /* Set view up vector   */
   vup.delta_x = 0.0; vup.delta_y = 1.0; vup.delta_z = 0.0;
   /* Set view plane normal*/
   vpn.delta_x = 0.0; vpn.delta_y = 0.0; vpn.delta_z = 1.0;

   peval_view_ori_matrix3(&vrp, &vpn, &vup,      /* Evaluate orient matrix */
                  &PEX_error, vrep.ori_matrix);

   SAFE_PEX("peval_view_ori_matrix3");           /* Check for error status */

   peval_view_map_matrix3( &map, &PEX_error,      /* Evaluate map matrix    */
                              vrep.map_matrix);

   SAFE_PEX("peval_view_map_matrix3");           /* Check for error status */

    pset_view_rep3( WS, VIEW, &vrep );     /* Set the view representation  */

    pset_edit_mode(PEDIT_INSERT);      /* Set edit mode to insert elements */
    pset_disp_upd_st(WS, PDEFER_WAIT, PMODE_NIVE);
                                    /* Set update state to WAIT */
                                    /* Enable WS z buffering */
    pset_hlhsr_mode(WS, PHIGS_HLHSR_MODE_ZBUFF);
}

/* Make_boxes
```

Make_boxes unposts and deletes any old structures that are in structure memory (for reset purposes), sets edit mode to insert elements and then defines a polyline cube and a fillarea cube in 3 dimensions. Structures POLYBOX and FILLBOX are defined to contain these data elements along with color and style attributes to be applied to the data elements. A higher level structure DISPLAY_STRUCT is defined to include both the POLYBOX and

```
FILLBOX structures and is then posted to the open workstation to be
displayed. The workstation is then updated to display the objects. The edit
mode is set to Replace elements in preparation of structure transfomation
updates caused by dial input.
*/

Make_boxes()
{

 Pmatrix3 identity;

                        /* Define polyline cube vectors */

 static Ppoint3    line_points1[]=       /* Define points for front face */
   {{  0.5, 0.5, 0.5} , { 1.0, 0.5, 0.5} ,
    {  1.0, 1.0, 0.5} , { 0.5, 1.0, 0.5} ,
    {  0.5, 0.5, 0.5}};

 static Ppoint3    line_points2[]=       /* Define points for back face */
   {{  0.5, 0.5, 0.0} , { 1.0, 0.5, 0.0} ,
    {  1.0, 1.0, 0.0} , { 0.5, 1.0, 0.0} ,
    {  0.5, 0.5, 0.0}};

 static Ppoint3    line_points3[]=       /* Define connecting line      */
   {{  0.5, 0.5, 0.5} , { 0.5, 0.5, 0.0}};

 static Ppoint3    line_points4[]=       /* Define connecting line      */
   {{  1.0, 0.5, 0.5} , { 1.0, 0.5, 0.0}};

 static Ppoint3    line_points5[]=       /* Define connecting line      */
   {{  1.0, 1.0, 0.5} , { 1.0, 1.0, 0.0}};

 static Ppoint3    line_points6[]=       /* Define connecting line      */
   {{  0.5, 1.0, 0.5} , { 0.5, 1.0, 0.0}};


                        /* Define solid cube faces */

 static Ppoint3    fill_points1[]=       /* Define points for front face */
   {{ -0.5, -0.5, 0.5} , {  0.0, -0.5, 0.5} ,
    {   0.0,  0.0, 0.5} , { -0.5,  0.0, 0.5}};

 static Ppoint3    fill_points2[]=       /* Define points for back face */
   {{ -0.5, -0.5, 0.0} , {  0.0, -0.5, 0.0} ,
    {   0.0,  0.0, 0.0} , { -0.5,  0.0, 0.0}};
```

```
static Ppoint3    fill_points3[]=       /* Define points for right face */
  {{  0.0, -0.5, 0.5} , { 0.0, -0.5, 0.0} ,
   {  0.0,  0.0, 0.0} , { 0.0,  0.0, 0.5}};

static Ppoint3    fill_points4[]=       /* Define points for left face */
  {{  -0.5, -0.5, 0.5} , { -0.5, -0.5, 0.0} ,
   {  -0.5,  0.0, 0.0} , { -0.5,  0.0, 0.5}};

static Ppoint3    fill_points5[]=      /* Define points for bottom face */
  {{  -0.5, -0.5, 0.5} , {  0.0, -0.5, 0.5} ,
   {   0.0, -0.5, 0.0} , { -0.5, -0.5, 0.0}};

static Ppoint3    fill_points6[]=       /* Define points for top face */
  {{  -0.5, 0.0, 0.5} , {  0.0, 0.0, 0.5} ,
   {   0.0, 0.0, 0.0} , { -0.5, 0.0, 0.0}};

Ppoint_list3 Line_list[5], Fill_list[5];

pset_edit_mode(PEDIT_INSERT);    /* Set edit mode to insert elements    */
punpost_all_structs(WS);         /* Unpost to remove any old structures */
pdel_all_structs();              /* Delete any old structures           */

  popen_struct(POLYBOX);          /* Open line drawing structure        */
    pset_line_colr_ind(2);       /* Assign default index color 2 to lines*/

    Line_list[0].num_points = 5; /* Fill in number of points in list    */
    Line_list[0].points = line_points1; /* Pointer to point array      */
    ppolyline3(&Line_list[0]);    /* Create a polyline element          */

    pset_line_colr_ind(3);       /* Assign default index color 3 to lines*/

      Line_list[1].num_points = 5;
      Line_list[1].points = line_points2;
      ppolyline3(&Line_list[1]);

    pset_line_colr_ind(4);       /* Assign default index color 4 to lines*/
      Line_list[2].num_points = 2;
      Line_list[2].points = line_points3;
      ppolyline3(&Line_list[2]);

    pset_line_colr_ind(5);       /* Assign default index color 5 to lines*/
      Line_list[3].num_points = 2;
      Line_list[3].points = line_points4;
      ppolyline3(&Line_list[3]);
```

```
pset_line_colr_ind(6);        /* Assign default index color 6 to lines*/
   Line_list[4].num_points = 2;
   Line_list[4].points = line_points5;
   ppolyline3(&Line_list[4]);

pset_line_colr_ind(7);        /* Assign default index color 7 to lines*/
   Line_list[5].num_points = 2;
   Line_list[5].points = line_points6;
   ppolyline3(&Line_list[5]);

pclose_struct();               /* Close line drawing structure        */

popen_struct(FILLBOX);         /* Open filled drawing structure       */
   pset_int_style(PSTYLE_SOLID); /* Set interior style to be solid    */
   pset_int_colr_ind(7);       /* Assign default index color 7 to face */
   Fill_list[0].num_points = 4;   /* Fill in number of points in list */
   Fill_list[0].points = fill_points1; /* Pointer to point array      */
   pfill_area3(&Fill_list[0]);     /* Create a fill area element      */

pset_int_colr_ind(3);         /* Assign default index color 3 to face */
   Fill_list[1].num_points = 4;
   Fill_list[1].points = fill_points2;
   pfill_area3(&Fill_list[1]);

pset_int_colr_ind(2);         /* Assign default index color 2 to face */
   Fill_list[2].num_points = 4;
   Fill_list[2].points = fill_points3;
   pfill_area3(&Fill_list[2]);

pset_int_colr_ind(4);         /* Assign default index color 4 to face */
   Fill_list[3].num_points = 4;
   Fill_list[3].points = fill_points4;
   pfill_area3(&Fill_list[3]);

pset_int_colr_ind(5);         /* Assign default index color 5 to face */
   Fill_list[4].num_points = 4;
   Fill_list[4].points = fill_points5;
   pfill_area3(&Fill_list[4]);

pset_int_colr_ind(6);         /* Assign default index color 6 to face */
   Fill_list[5].num_points = 4;
   Fill_list[5].points = fill_points6;
   pfill_area3(&Fill_list[5]);
```

```
    pclose_struct();                /* Close filled drawing structure      */

    make_identity(identity);

    popen_struct(DISPLAY_STRUCT); /* Open the top level display structure */
    pset_view_ind(VIEW);           /* Set the view index to be used        */
    pset_hlhsr_id(PHIGS_HLHSR_ID_ON);        /* Turn on Z buffering       */
    plabel(TRANS);                /* Insert a label for future updates    */
                          /* set transformation matrix*/
    pset_local_tran3(identity, PTYPE_REPLACE);
       pexec_struct(FILLBOX);        /* Include the filled square          */
       pexec_struct(POLYBOX);       /* Include the line drawn square       */
       pset_hlhsr_id(PHIGS_HLHSR_ID_OFF);     /* Turn off Z buffering      */
    pclose_struct();

    ppost_struct(WS, DISPLAY_STRUCT, 1.0); /* Post DISPLAY_STRUCT, prio 1 */

    pupd_ws(WS,PUPD_PERFORM);                /* Update the workstation     */
    pset_edit_mode(PEDIT_REPLACE);           /* Set edit mode to replace   */

}


/*
make_identity

This routine sets the passed matrix to be an  identity matrix.
*/

make_identity(matrix)
    Pmatrix3 matrix;
{
    matrix[0][0] = 1;
    matrix[0][1] = 0;
    matrix[0][2] = 0;
    matrix[0][3] = 0;
    matrix[1][0] = 0;
    matrix[1][1] = 1;
    matrix[1][2] = 0;
    matrix[1][3] = 0;
    matrix[2][0] = 0;
    matrix[2][1] = 0;
    matrix[2][2] = 1;
    matrix[2][3] = 0;
```

```
    matrix[3][0] = 0;
    matrix[3][1] = 0;
    matrix[3][2] = 0;
    matrix[3][3] = 1;
}


Cleanup()                          /* Cleanup routine when done      */
{
 punpost_all_structs(WS);          /* Unpost all structures on ws    */
 pdel_all_structs();               /* Delete all structures */
 pclose_ws(WS);                    /* Close workstation      */
 pclose_phigs();                   /* Close PHIGS            */
}
```

header3.h

```
/*

   header3.h

Contains global header information for example program 3.

       Author: James Buckmiller  May 1990.

       Copyright (C) 1990, Evans & Sutherland

*/

#define WS              1
#define POLYBOX         1     /* define some constants to be used later */
#define FILLBOX         2
#define DISPLAY_STRUCT  3
#define VIEW            4
#define TRANS           5
#define SPACEKEYSYM    32
#define MAXDIALS        8
#define CHARS_PER_DIAL  8
#define SAFE_PEX(routine) Error_Check(__FILE__, __LINE__, routine)
#define DEG_TO_RAD(D)    ((3.14159265358 / 180.0) * (D))
#define DIALSCALE       .005     /*  Dial Scale value */

extern Window myWin;
extern Display *dpy;
extern char *ProgramName;

extern Pmatrix3 identity;
extern Pint PEX_error;/* For PEX error numbers */
```

example4.c

```
/*

        example4.c

This program incorporates program 3 functionality for model interaction
with dials as well as keyboard control for reset and exit functions. The
new functionality added for program 4 is the E&S picking extension to PEX.
As a model element is picked using mouse button 1 the element is deleted
from the structure.  If an element is picked with either mouse button 2 or
3 a prepick highlight of the object is performed to visually show the user
what elements will be picked if a pick were to be performed. To reset the
picture press the r key on the keyboard. To exit the client press the e key
on the keyboard.

       Author: James Buckmiller May 1990.
       Modified: J. Buckmiller Mar 1991.    Approved C binding

       Copyright (C) 1990, Evans & Sutherland

*/
#include <X11/Xlib.h>
#include <X11/Xatom.h>
#include <X11/extensions/XInput.h>
#include <X11/phigs/phigs.h>
#include <X11/keysymdef.h>
#include <X11/extensions/XPick.h>

#include "header4.h"

Window myWin;
Display *dpy;
char *ProgramName;
Pint PEX_error;                         /* For PEX error numbers */

int Major_op, First_ev, First_err;      /* pick extension parameters */
Pint_list nset_names;                   /* pick list nameset */
Pint namesints[1];                      /* nameset list arrays */
XPickFilter pick_incl, pick_excl;    /* pick inclusion/exclusion filters */
PC pc;                                  /* pick context */

main(argc, argv)
int       argc;
char   *argv[];
{
```

```
   int i;
   char *geom = NULL;
   char *display = NULL;
   int winposx, winposy, winwidth, winheight;

   ProgramName = argv[0];

   winposx = 100;                              /* default window geometry */
   winposy = 100;
   winwidth  = 600;
   winheight = 600;

      for (i=1; i < argc; i++)                  /* Parse the command line */
      {
      char *arg = argv[i];

   if (arg[0] == '-')
         {
       switch (arg[1])
             {
         case 'd':       /* -display host:dpy */
          if (++i >= argc) usage ();
            display = argv[i];
          continue;
         case 'g':       /* -geometry host:dpy */
          if (++i >= argc) usage ();
            geom = argv[i];
          continue;
         default:
          usage ();
        }
    }
 }

if (!(dpy = XOpenDisplay(display)))      /* Attempt to open the display */
   {
   perror("Cannot open display\n");
   exit(-1);
   }

if (geom)
 {               /* Generate position and size from the geometry string */
   (void) XParseGeometry(geom, &winposx, &winposy, &winwidth, &winheight);
 }
```

```
                        /* Create a simple, unmapped input/output window */
myWin = XCreateSimpleWindow(dpy, RootWindow(dpy, DefaultScreen(dpy)),
                        winposx, winposy, winwidth, winheight, 0,NULL,NULL);

                        /* Change the window name property */
XChangeProperty(dpy, myWin, XA_WM_NAME, XA_STRING, 8, PropModeReplace,
                       "Example 4:    r Key = Reset  e Key = Exit", 40);

                        /* Map the window for display */
XMapWindow(dpy, myWin);

                        /* Begin PHIGS calls          */
StartPhigs(dpy, myWin);
}

/*
usage

This routine prints out command line argument information if the user
supplied arguments are incorrect.
*/

usage ()            /* Print usage message */
{
  fprintf (stderr, "usage:  %s [-options ...]\n\n", ProgramName);
  fprintf (stderr, "where options include:\n");
  fprintf (stderr, "    -display host:dpy        X server to use\n");
  fprintf (stderr, "    -geometry geom           geometry of window\n");
  fprintf (stderr, "\n");
  exit (1);
}

/*
Get_events

This routine is the event handeling routine. First the usual X events are
trapped. If an expose event occurs then a PHIGS redrawallstructures is
called. If a Keyboard event occurs the keysym is reviewed to see if it is
an "e" for exit or an "r" for a reset of the display. If the event that is
generated is not a usual X event then it is checked to be an extension event
for dials events or picking events. If dials event then the object is
transformed. If a picking event then the element picked is deleted from the
structure listed in the pick info.
*/
```

```
Get_events(dpy)
Display  *dpy;
{

   XDeviceInfo *devices = NULL;
   XDevice *dials = NULL;
   XEventClass DeviceMotionClass[100];
   XID dials_id;
   XEvent report;

   XKeyPressedEvent *pev;
   KeySym key;
   char buf[20];

   unsigned long event_mask;
   XButtonPressedEvent *bdown;
   XButtonReleasedEvent *bup;
   XPickEvent *pick;

   int knob_totals[MAXDIALS];
   Atom dials_atom = 0;
   XStringFeedbackControl strfc;
   int done = 0,j;
   KeySym ledstring[MAXDIALS][CHARS_PER_DIAL], blankled[MAXDIALS];

   Pmatrix3 composite;
   static Pmatrix3 currmatrix[] =
            {{1,0,0,0},{0,1,0,0},{0,0,1,0},{0,0,0,1}};

   Pvec3 scale, trans;
   int ndevices = 0, i, EventCount = 0, DeviceMotion = -1;
   int rotx=0, roty=0, rotz=0;
   Ppoint3 cntr;
                                                  /* Dial labels */
   static char textstring[MAXDIALS][CHARS_PER_DIAL] =
                                  { " X ROT  ", " Y ROT  ", " Z ROT  ",
                                    " SCALE  ", " X TRAN ", " Y TRAN ",
                                    " Z TRAN ", "        " };

   cntr.x = 0.0;              /* Define center point of transformations */
   cntr.y = 0.0;
   cntr.z = 0.0;
   scale.delta_x = 1.0;       /* Define initial scale to be 1           */
   scale.delta_y = 1.0;
```

```
    scale.delta_z = 1.0;
    trans.delta_x = 0.0;          /*  Define initial translation to be 0     */
    trans.delta_y = 0.0;
    trans.delta_z = 0.0;

                              /* Determine if picking extension is present */
                              /* Get the opcodes for the errors and events */
    XQueryExtension(dpy, PICKNAME, &Major_op, &First_ev, &First_err);

                               /* Get the atom ID for the Knob box */
    dials_atom = XInternAtom(dpy, "KNOB_BOX", True);

                              /* Get list of Input devices        */
    devices = XListInputDevices(dpy, &ndevices);

                /* Find the dials device in the list and open the device */
    for (i = 0; i < ndevices; i++, devices++)
       if ((devices->type == dials_atom) &&
             (devices->use == IsXExtensionDevice))
       {
          dials_id = devices->id;
          dials = XOpenDevice(dpy, dials_id);  /* Get Xdevice structure */
          break;                               /* for the dials         */
       }

       if (!dials)
       {
          fprintf(stderr, "?dials box not found in X extension device
             list.\n");
          exit(1);
       }

                                  /* Get event class values for dials     */
DeviceMotionNotify(dials, DeviceMotion, DeviceMotionClass[EventCount]);
EventCount++;
                                /* Tell server to pass on Extension events */
XSelectExtensionEvent(dpy, myWin, DeviceMotionClass, EventCount);

                                  /* Set the event mask for the window    */
XSelectInput(dpy, myWin, ButtonPressMask | ButtonReleaseMask|
                  EnterWindowMask | LeaveWindowMask | KeyPressMask |
                  ExposureMask);

    /* Set the picking mode to return the item picked that is nearest in Z */
```

```
XSetPickMode(dpy, pc, pick_near);

                        /* Set the pick box size to 5x5 pixels        */
XSetPickBoxSize(dpy, pc, 5, 5);

                      /* Cause all of pickable items to be highlighted */
XSetPickHighlightingMode(dpy, pc, pick_highlighting_command);

                                  /* Select pick events to be returned */
XSelectPickEvents(dpy, myWin, PickMask);

for (i = 0; i < MAXDIALS; i++)           /* Load the keysym arrays */
   for (j = 0; j < CHARS_PER_DIAL; j++)
   {
      ledstring[i][j] = (KeySym) textstring[i][j];     /* Dial labels  */
      blankled[j] = SPACEKEYSYM;                       /* Blank labels */
   }

   strfc.class = StringFeedbackClass;
   strfc.length = sizeof(XStringFeedbackControl);
   strfc.num_keysyms = CHARS_PER_DIAL;

   for (i=0; i<MAXDIALS; i++)             /* Set the dial labels upon */
   {                                      /* startup of client.       */
      strfc.id = i;
      strfc.syms_to_display = ledstring[i];
      XChangeFeedbackControl(dpy, dials, DvString, &strfc);
   }

while (!done)
   {
      XNextEvent(dpy, &report);      /* Get next event from event queue */

      if (report.type < LASTEvent)   /* Check if extension event or not */
      {
         switch (report.type)
         {
         case EnterNotify:                    /* Turn on the dial labels */
            for (i=0; i<MAXDIALS; i++)        /* when the cursor enters  */
             {                                /* the window.             */
             strfc.id = i;
             strfc.syms_to_display = ledstring[i];
             XChangeFeedbackControl(dpy, dials, DvString, &strfc);
             }
```

```
    break;
case KeyPress:                          /* Trap keyboard events   */
                                        /* and perform function.  */
   pev = (XKeyPressedEvent *) &report;
   XLookupString(pev, buf, sizeof(buf), &key, NULL);
   if(buf[0] == 'r') /* Reset picture          */
     {
   make_identity(currmatrix); /* Currmatrix = identity matrix */
      Make_boxes();                    /* Rebuild the structures*/
                                 /* Reset xformation parameters */
        scale.delta_x = scale.delta_y = scale.delta_z = 1.0;
        trans.delta_x = trans.delta_y = trans.delta_z = 0.0;
        rotx = roty = rotz = 0;
        }
        if(buf[0] == 'e')               /* Exit  Program        */
        done = 1;
       break;

case ButtonPress:                         /* Trap button press */
   bdown = (XButtonPressedEvent *) &report;
   if (bdown->button == Button1)
{                                       /* Do pick traversal */
         XPick(dpy, myWin, pc, bdown->x, bdown->y);
}
       else
{                          /* Do pick highlighting traversal */
         XPrePick(dpy, myWin, pc, bdown->x, bdown->y);
}
       break;

case ButtonRelease: /* Redraw to remove prepick highlight */
   bup = (XButtonReleasedEvent *) &report;
   predraw_all_structs(WS, PFLAG_ALWAYS);
 break;

case Expose:                            /* Expose events = redraw */
   predraw_all_structs(WS, PFLAG_ALWAYS);
 break;

case LeaveNotify:      /* Blank LEDs when cursor leaves window */
   for (i=0; i<MAXDIALS; i++)
   {
         strfc.id = i;
         strfc.syms_to_display = blankled;
```

```
                XChangeFeedbackControl(dpy, dials, DvString, &strfc);
              }
              break;

          }
       }
            /* else it's an extension event */

else if (report.type == DeviceMotion)          /* Dials input */
   {
   XDeviceMotionEvent *dm = (XDeviceMotionEvent *) &report;

   for (i=0; i<MAXDIALS; i++)     /* Zero the knob accumulator array */
      knob_totals[i] = 0;
/*-------------------------------------------------------------------------
The following piece of code goes out to the event queue and scoops
off all dial motion events that are found on the queue with the
XCheckTypedEvent call.These events are then accumulated for each
axis and then processed with the acccumulated values.

The reason for doing this is to increase the performance of the system. If
an update of the workstation display is performed for every dial event that
occurs the display will get behind thus causing a lag time between when the
dials stop sending events and the system finishes unpiling the event queue.
For this application grabbing all dial events off the queue works well
however, one must beware that if an application allows the dials to be
redefined with some other event (function key or pick menu) this method may
not be the way to get the dial events since there may be keypress or pick
events intermixed with the dials events. To get around this problem one may
wish to use the XPeekEvent routine to look ahead one event to be sure that
it is the same event class as the ones being accumulated.
-------------------------------------------------------------------------*/

do                        /* Collapse the events before processing */
   for (i=0; i < dm->axes_count; i++)
      knob_totals [dm->first_axis+i] += dm->axis_data[i];
                         /* Gather all dial events from the event queue */
while (XCheckTypedEvent (dpy, DeviceMotion, dm));

          /* Process DeviceMotion events  */

if (knob_totals[0])              /* dial 1 input check rot x */
   rotx += knob_totals[0];

if (knob_totals[1])              /* Dial 2 input check rot y */
   roty += knob_totals[1];
```

```
if (knob_totals[2])          /* Dial 3 input check rot z */
   rotz += knob_totals[2];

if (knob_totals[3])          /* Dial 4 input check uniform scale */
   {
      scale.delta_x += knob_totals[3] * DIALSCALE;
      scale.delta_y += knob_totals[3] * DIALSCALE;
      scale.delta_z += knob_totals[3] * DIALSCALE;
   }

if (knob_totals[4])          /* Dial 5 input check trans x */
   trans.delta_x += knob_totals[4] * DIALSCALE;

if (knob_totals[5])          /* Dial 6 input check trans y */
   trans.delta_y += knob_totals[5] * DIALSCALE;

if (knob_totals[6])          /* Dial 7 input check trans z */
   trans.delta_z += knob_totals[6] * DIALSCALE;

                                  /* Build the transformation matrix */
pbuild_tran_matrix3(&cntr, &trans, DEG_TO_RAD(rotx), DEG_TO_RAD(roty)
               ,DEG_TO_RAD(rotz), &scale, &PEX_error
               ,currmatrix);
   SAFE_PEX("pbuild_tran_matrix3");          /* Check for error status */
                                  /* Combine old and new matrices */
   popen_struct(DISPLAY_STRUCT);          /* Open structure for editing */
      {
         pset_elem_ptr(0);                /* Reset element pointer      */
         pset_elem_ptr_label(TRANS);      /* Find transformation label  */
         poffset_elem_ptr(1);             /* Point at matrix            */
                                  /* replace matrix */
         pset_local_tran3(currmatrix, PTYPE_PRECONCAT);
      }
   pclose_struct();                    /* Close structure            */
   predraw_all_structs(WS, PFLAG_ALWAYS); /* Redraw the structure    */

   }
else if (report.type == First_ev + XPickEventOffset) /* Pick event   */
{       /****** picking event  ******/
/*-------------------------------------------------------------
The picking extension returns a structure that contains
information about what element and structure was picked,
where in screen space or model space the pick occured
and other usefull information. (see pg 3-4 X Picking Extension
document). For this example we are only using the structure
```

```
number and element number to position the element pointer
in preparation for a delete element PHIGS call.
-------------------------------------------------------------*/
   pick = (XPickEvent *) &report;
      popen_struct(pick->structureid);     /* Open structure returned */
      pset_elem_ptr(0);                                 /* by pick */
      poffset_elem_ptr(pick->elementid);   /* Go to picked element   */
      pdel_elem();                         /* Delete the element     */
      pclose_struct();                     /* Close the structure    */
      pupd_ws(WS,PUPD_PERFORM);            /* Cause an update of WS  */
                                        /* to show element removed*/
      }
   }
}
```

motif4.c

```
/*
        m o t i f 4 . c

This program expands on motif3 to include picking of PEX structures.
Picking is accomplished via the Evans & Sutherland picking extension to X.
Structure elements can be picked via the mouse buttons.  Button 1 will
delete the picked element, while buttons 2 or 3 will highlight the picked
element while the button is held down.

Reset and quit push buttons are provided as in motif3.  When the reset
button is pushed, the structures are restored to their original pristine
state.

   Author:  Rich Thomson
   Date: Thursday, June 12th, 1990
   Modified: J. Buckmiller   Mar 1991.   Approved C binding

   Copyright (C) 1990, Evans & Sutherland Computer Corporation
*/

#include <X11/Xlib.h>
#include <X11/extensions/XInput.h>
#include <X11/extensions/XPick.h>
#include <X11/phigs/phigs.h>
#include <X11/Intrinsic.h>
#include <Xm/RowColumn.h>
#include <Xm/PushBG.h>
#include <Xm/DrawingA.h>

#include "header4.h"

Display *dpy;                       /* X11 display connection */
char *ProgramName;
int DeviceMotion;/* device motion event type */
Window drawWindow;/* drawing window */
Pint PEX_error;                     /* For PEX error numbers */

int Major_op, First_ev, First_err;/* pick extension parameters */
Pint_list nset_names;               /* pick list nameset */
Pint namesints[1];                  /* nameset list arrays */
XPickFilter pick_incl, pick_excl;/* pick inclusion/exclusion filters */
PC pc;                              /* pick context */

static XDevice *dials = NULL;       /* dials device */
```

```
static Pmatrix3 currmatrix = {{1,0,0,0},{0,1,0,0},{0,0,1,0},{0,0,0,1}};
static Boolean done = False;
static Pvec3 scale = { 1.0, 1.0, 1.0 };
static Pvec3 trans = { 0.0, 0.0, 0.0 };
static int rotx = 0, roty = 0, rotz = 0;

/*
quit_CB

The callback procedure for the quit pushbutton widget.  It simply sets the
event processing exit flag to True, which will cause Get_events to stop
processing events.
*/
void quit_CB(quitButton, client_data, call_data)
     Widget quitButton;
     caddr_t client_data;
     XmAnyCallbackStruct *call_data;
{
   if (call_data->reason == XmCR_ACTIVATE)
      done = True;
}

/*
reset_CB

The callback procedure for the reset button.  It re-initializes the
parameters that define the transformation matrix corresponding to
translate, rotate and scale operations. The posted structures are also
re-initialized since picking elements could have caused some elements to
be deleted.
*/
void reset_CB(resetButton, client_data, call_data)
     Widget resetButton;
     caddr_t client_data;
     XmAnyCallbackStruct *call_data;
{
   if (call_data->reason == XmCR_ACTIVATE)
      {
         Make_boxes();
                        /* reset xformation parameters */
         scale.delta_x = scale.delta_y = scale.delta_z = 1.0;
         trans.delta_x = trans.delta_y = trans.delta_z = 0.0;
         rotx = roty = rotz = 0;
      }
}
```

```
/*
drawArea_CB

The callback procedure for the drawing area widget.  It redraws all
the structures on the workstation.
*/
void drawArea_CB(drawArea, client_data, call_data)
     Widget drawArea;
     caddr_t client_data;
     XmDrawingAreaCallbackStruct *call_data;
{
   if (call_data->reason == XmCR_EXPOSE)
      predraw_all_structs(WS, PFLAG_ALWAYS);
}

/*
pick_CB

This routine performs the picking operation on the structures drawn in
the drawing area widget.

When a ButtonPress event is received, a pick operation is performed if
the user pressed Button1. The pick operation will cause picking events
to be generated for any objects selected.  If the user pressed some
button other than Button1, a pre-pick operation is performed, which will
highlight the object selected. The pick operations are performed at the
device (pixel) coordinates where the button press took place.

When a ButtonRelease event is received, all structures posted to the
workstation are redrawn.  This will cause deleted structure elements to
be visually reflected on the screen (for Button1) as well as remove
highlighting caused by other mouse buttons.
*/

void pick_CB(drawArea, client_data, call_data)
     Widget drawArea;
     caddr_t client_data;
     XmDrawingAreaCallbackStruct *call_data;
{
   XButtonPressedEvent *bpress = (XButtonPressedEvent *) call_data->event;

   if (call_data->reason == XmCR_INPUT)
      {
         switch (call_data->event->type)
         {
            case ButtonPress:
```

```
                if (bpress->button == Button1)
                 XPick(dpy, call_data->window, pc, bpress->x, bpress->y);
                else
                 XPrePick(dpy, call_data->window, pc, bpress->x, bpress->y);
                break;

              case ButtonRelease:
                predraw_all_structs(WS, PFLAG_ALWAYS);
                break;
              }
        }
}

/*
knob_labels

This array holds the KeySym's that contain the knob labels.  It is used by
the enter and leave window handlers to blank out the labels when the
pointer is not in the drawing area window.

Dial labels are CHARS_PER_DIAL KeySyms per dial.  Ascii characters can be
converted to KeySyms by C type casting.
*/

static KeySym knob_labels[MAXDIALS][CHARS_PER_DIAL];

/*
enter_handler

This event handler restores the knob labels to our labels when the
pointer moves into the drawing window.  Conditionally labelling the dials
in this way gives extra feedback to the user that the dials are active
only when the mouse is inside the appropriate window.
*/
void enter_handler(widget, client_data, event, continue_to_dispatch)
     Widget widget;
     caddr_t client_data;
     XEvent *event;
     Boolean *continue_to_dispatch;
{
   register int i;
   XStringFeedbackControl strfc;

   strfc.class = StringFeedbackClass;
                        /* initialize the feedback structure */
```

```
    strfc.length = sizeof(XStringFeedbackControl);
    strfc.num_keysyms = CHARS_PER_DIAL;

    for (i = 0; i < MAXDIALS; i++)/* change each dial */
        {
            strfc.id = i;  /* id is the knob to change */
            strfc.syms_to_display = knob_labels[i];
            XChangeFeedbackControl(dpy, dials, DvString, &strfc);
        }
}

/*
leave_handler

This event handler blanks the knob labels when the pointer leaves the
drawing window.
*/
void leave_handler(widget, client_data, event, continue_to_dispatch)
     Widget widget;
     caddr_t client_data;
     XEvent *event;
     Boolean *continue_to_dispatch;
{
    static XStringFeedbackControl strfc;
    static KeySym blanks[CHARS_PER_DIAL];
    static Boolean initialized = False;
    register int i;

    if (!initialized)               /* initialize variables the first */
                                    /* time we're called  */
     {
       for (i = 0; i < CHARS_PER_DIAL; i++)
          blanks[i] = (KeySym) ' ';/* prepare a blank keysym array */

       strfc.class = StringFeedbackClass; /*initialize the feedback struct*/
       strfc.length = sizeof(XStringFeedbackControl);
       strfc.num_keysyms = CHARS_PER_DIAL;
       strfc.syms_to_display = blanks;

       initialized = True;/* remember we've been initialized */
     }

    for (i = 0; i < MAXDIALS; i++)/* for each dial */
     {
```

```
        strfc.id = i;/* indicate which dial to change */
                                        /* change it */
XChangeFeedbackControl(dpy, dials, DvString, &strfc);
    }
}

/*
open_knob

This routine opens the knob box on the given display and selects device
motion extension events on the given window.  Extension events are
selected by first invoking the appropriate macro on an XEventClass
structure (in this case DeviceMotionNotify) and then calling
XSelectExtensionEvent.
*/

void open_knob()
{
                                        /* dial labels as ascii strings */
   static char textstring[MAXDIALS][CHARS_PER_DIAL] = {
    " X ROT  ", " Y ROT  ", " Z ROT  "," SCALE  ",
   " X TRAN ", " Y TRAN ", " Z TRAN ", "       " /* eighth label is blank */
    };
   register int knob, i;
   int ndevices;  /* number of extension devices */
   XDeviceInfo *devices = NULL;/* extension device info list */
   Atom dials_atom = XInternAtom(dpy, "KNOB_BOX", True);
                  /* intern device name into an atom */
   XID dials_id;  /* device ID for dials box */
   XEventClass eventClass[1];

   devices = XListInputDevices(dpy, &ndevices); /* get list of devices */

   for (i = 0; i < ndevices; i++, devices++)
      if ((devices->type == dials_atom) && (devices->use ==
            IsXExtensionDevice))
      {            /* did we find the dial box? */
      dials_id = devices->id;/* yes, remember its device ID */
      dials = XOpenDevice(dpy, dials_id); /* and open it */
      break;             /* we only want the first one... */
      }

   if (!dials)     /* couldn't open or find dials */
    {
      fprintf(stderr, "?couldn't open dials box.\n");
```

```
        exit(1);
    }
                    /* select device motion events */
    DeviceMotionNotify(dials, DeviceMotion, eventClass[0]);
    XSelectExtensionEvent(dpy, drawWindow, eventClass, 1);

    for (knob = 0; knob < MAXDIALS; knob++)
                                    /* convert ascii labels to KeySyms */
    for (i = 0; i < CHARS_PER_DIAL; i++)
        knob_labels[knob][i] = (KeySym) textstring[knob][i];
}

/*
init_pick

This routine initializes the picking extension.  It must be called after
the pick context is established (in routine SetupWorkstation, file
phigs4.c)and before event processing is begun.

Major_op is the major opcode of the picking extension.  First_ev is the
first event number dynamically allocated for the extension.  First_err is
the first error number dynamically allocated for the extension.
*/

void init_pick()
{                   /* query picking extension */
                    /* for major opcode, first event */
    XQueryExtension(dpy, PICKNAME, &Major_op, &First_ev, &First_err);

                    /* Set the picking mode to return */
                    /* the item picked that is nearest */
    XSetPickMode(dpy, pc, pick_near);/* in Z */

    XSetPickBoxSize(dpy, pc, 5, 5);/* Set pick box size to 5x5 pixels */
                    /* Cause all of picked item to be */
                    /* highlighted */
    XSetPickHighlightingMode(dpy, pc, pick_highlighting_command);
                    /* Select pick and pick path events */
                    /* to be returned */
    XSelectPickEvents(dpy, drawWindow, PickMask);
}

/*
main
```

The main routine creates the widget hierarchy for the program, opens the knob box and then calls StartPhigs. StartPhigs will then call Get_events to initiate event processing.

The widget hierarchy used here is:

```
motif4 (class topLevelShell)
 |
 +-- rowcol (class RowColumn)
       |
       +--- drawArea (class DrawingArea)
       |
       +--- reset (class PushButtonGadget)
       |
       +--- quit (class PushButtonGadget)
```

The row column widget is used for organizing its child widgets into a columnar layout. The drawing area widget is used for PEX operations and the PEX workstation is opened on its window. The two push buttons are used to supply reset and quit operations.

Any necessary resources for the widgets are specified here in the program, which override any user defaults or command-line options. Note that this is not very friendly to the user who may want to change the font of the push buttons. A friendlier way is to provide an application defaults file which the user may override with user defaults or command-line arguments. For simplicity, I have set the arguments here directly.

The display connection (dpy) and window ID (drawWindow) of the drawing area widget are available after the widget hierarchy has been realized.

BEWARE!! BEWARE!!
The drawing area widget in Motif 1.0 has a bug in that it ignores height and width resources supplied at creation time. A workaround I've found is to set the margins of the drawing area to be half the desired height and width. Since the margins specify the boundary between the drawing area widget's border and any children of the drawing area widget (we have none here), the drawing area widget will be sized to contain its children plus twice the margins in each direction. Hence to get a 600x600 drawing area widget, you can set the margins to 300. Other workarounds suggested involve creating the drawing area widget as a child of other widgets, but where there are no children of the drawing area, I prefer setting the margins.
*/

```
main(argc, argv)
int       argc;
```

```
char   *argv[];
{
   Arg args[10];
   register int n;
   Widget topLevel, rowColumn, quitButton, resetButton, drawArea;
   XFontStruct *buttonFont;
   XmFontList fontList;

   ProgramName = argv[0];
                  /* create topLevelShell */
   topLevel = XtInitialize(ProgramName, "Example", NULL, 0, &argc, argv);

   n = 0;         /* window title is a regular string */
   XtSetArg(args[n], XmNtitle, "Example 4"); n++;
   XtSetValues(topLevel, args, n);

   buttonFont =   /* find the font we want for buttons */
      XLoadQueryFont(XtDisplay(topLevel), "-*-Helvetica-Bold-R-Normal--
            14*");
   if (buttonFont)
      fontList = XmFontListCreate(buttonFont, XmSTRING_DEFAULT_CHARSET);

   rowColumn =    /* create the row column for layout */
      XtCreateManagedWidget("rowcol", xmRowColumnWidgetClass, topLevel,
            NULL, 0);

   n = 0;         /* create the drawing area for PEX */
   XtSetArg(args[n], XmNmarginWidth, 300); n++; /* size appropriately */
   XtSetArg(args[n], XmNmarginHeight, 300); n++;
   drawArea = XtCreateManagedWidget("drawArea", xmDrawingAreaWidgetClass,
               rowColumn, args, n);
                  /* add exposure and input callbacks */
   XtAddCallback(drawArea, XmNexposeCallback, drawArea_CB, NULL);
   XtAddCallback(drawArea, XmNinputCallback, pick_CB, NULL);

                  /* these event handlers take care */
                  /* of blanking and restoring the */
                  /* dial labels */
   XtAddEventHandler(drawArea, EnterWindowMask, False, enter_handler,
            NULL);
   XtAddEventHandler(drawArea, LeaveWindowMask, False, leave_handler,
            NULL);
   n = 0;         /* create the reset button */
   XtSetArg(args[n], XmNlabelString,
         XmStringCreate("Click here to reset the picture.",
```

```
            XmSTRING_DEFAULT_CHARSET)); n++;
    if (buttonFont)
     {
       XtSetArg(args[n], XmNfontList, fontList); n++;
     }
    resetButton = XtCreateManagedWidget("reset", xmPushButtonGadgetClass,
                  rowColumn, args, n);
                  /* add an activation callback */
    XtAddCallback(resetButton, XmNactivateCallback, reset_CB, NULL);

    n = 0;          /* create the quit button */
    XtSetArg(args[n], XmNalignment, XmALIGNMENT_CENTER);
    XtSetArg(args[n], XmNlabelString,
          XmStringCreate("Click here to quit the program.",
             XmSTRING_DEFAULT_CHARSET)); n++;
    if (buttonFont)
     {
       XtSetArg(args[n], XmNfontList, fontList); n++;
     }
    quitButton =
       XtCreateManagedWidget("quit", xmPushButtonGadgetClass, rowColumn,
             args, n);
                  /* add an activation callback */
    XtAddCallback(quitButton, XmNactivateCallback, quit_CB, NULL);

    XtRealizeWidget(topLevel);/* realize widget hierarchy */

    dpy = XtDisplay(drawArea);/* get the display connection */
    drawWindow = XtWindow(drawArea);/* get the d.a. widget's window ID */

    open_knob();    /* open the knob box */

    StartPhigs(dpy, drawWindow);/* Begin PHIGS calls */
}

/*
Get_events

This routine processes events requested by the program.  XtNextEvent
obtains the next event from the input queue and places it in report.  The
type of the event is then examined to determine if it is an extension
event or a regular X event.  The constant LASTEvent (defined in X.h) is
bigger than the event type of any X event and can be used to
differentiate extension events from normal X events.
```

Regular events are handled by the toolkit dispatch mechanism via XtDispatchEvent. Extension events (DeviceMotion and picking events) are hanlded on a case-by-case basis.

When a DeviceMotion event is encountered, all device motion events are removed from the event queue and accumulated into knob_totals, since the dials box reports relative changes. Event explosion is a very real possibility since every device motion event requires 2 XEvent structures (only 6 axes' worth of data fit in a single XEvent) and the sample rate of the dials box is high. Since this program is only concerned with cumulative changes in the dials values, it is safe to condense the device motion events via XCheckTypedEvent. Since XCheckTypedEvent can remove events that are not at the head of the event queue, it may not be appropriate for situations where the semantics of a device motion event can be changed by another event (for instance, a key or button press).

When a pick event is encountered, the picked structure element is deleted from the structure and the workstation is updated.
*/

```
Get_events(dpy)
     Display  *dpy;
{
   XEvent report;
   int i;
   Ppoint3 cntr;

   cntr.x = 0.0;                    /* center point of transformations */
   cntr.y = 0.0;                    /* is the origin */
   cntr.z = 0.0;
   scale.delta_x = 1.0;             /* initial scale is 1 */
   scale.delta_y = 1.0;
   scale.delta_z = 1.0;
   trans.delta_x = 0.0;             /* initial translation is 0 */
   trans.delta_y = 0.0;
   trans.delta_z = 0.0;

   pset_edit_mode(PEDIT_REPLACE);   /* Set edit mode to replace elements */

   init_pick();                     /* initialize picking */

   while (!done)                    /* until user wants to quit... */
    {
      XtNextEvent(&report);/* get the next event */

      if (report.type < LASTEvent)
         XtDispatchEvent(&report);
```

```
else if (report.type == DeviceMotion) /* handle device motion event */
{
   XDeviceMotionEvent *dm = (XDeviceMotionEvent *) &report;
   int knob_totals[8];

   for (i = 0; i < MAXDIALS; i++
                              /* Initialize knob values array */
      knob_totals[i] = 0;

   do                         /* Compress motion events */
    for (i = 0; i < dm->axes_count; i++)
      knob_totals [dm->first_axis+i] += dm->axis_data[i];
   while (XCheckTypedEvent(dpy, DeviceMotion, dm));

                              /* Process device motion events */
   if (knob_totals[0])        /* dial 1 input check rot x */
      rotx += knob_totals[0];

   if (knob_totals[1])        /* Dial 2 input check rot y */
      roty += knob_totals[1];

   if (knob_totals[2])        /* Dial 3 input check rot z */
      rotz += knob_totals[2];

   if (knob_totals[3])        /* Dial 4 input check uniform scale */
    {
      scale.delta_x += knob_totals[3] * DIALSCALE;
      scale.delta_y += knob_totals[3] * DIALSCALE;
      scale.delta_z += knob_totals[3] * DIALSCALE;
    }

   if (knob_totals[4])         /* Dial 5 input check trans x */
      trans.delta_x += knob_totals[4] * DIALSCALE;

   if (knob_totals[5])         /* Dial 6 input check trans y */
      trans.delta_y += knob_totals[5] * DIALSCALE;

   if (knob_totals[6])         /* Dial 7 input check trans z */
      trans.delta_z += knob_totals[6] * DIALSCALE;

                              /* Build the transformation matrix */
      pbuild_tran_matrix3(&cntr, &trans, DEG_TO_RAD(rotx),
         DEG_TO_RAD(roty), DEG_TO_RAD(rotz), &scale, &PEX_error,
         currmatrix);
      SAFE_PEX("pbuild_tran_matrix3");      /* Check for error status */
```

```
                                        /* Combine old and new matrices */
        popen_struct(DISPLAY_STRUCT);   /* Open structure for editing */
    {
        pset_elem_ptr(0);               /* Reset element pointer      */
        pset_elem_ptr_label(TRANS);     /* Find transformation label  */
        poffset_elem_ptr(1);            /* Point at matrix            */
                              /* replace matrix */
            pset_local_tran3(currmatrix, PTYPE_PRECONCAT);
    }
        pclose_struct();                /* Close structure            */
        predraw_all_structs(WS, PFLAG_ALWAYS); /* Redraw the structure */

      }
    else if (report.type == First_ev + XPickEventOffset)
                                  /* picking event */
  {
        XPickEvent *pick = (XPickEvent *) &report;

        popen_struct(pick->structureid); /* open picked structure */
        {
           pset_elem_ptr(0);/* reset the element points */
           poffset_elem_ptr(pick->elementid);
                                  /* set the element pointer to */
                                  /* the picked element  */
           pdel_elem();           /* delete the picked element */
        }
        pclose_struct();

        pupd_ws(WS, PUPD_PERFORM);/* cause an update */
    }
  }
}
```

phigs4.c

```
/*
   phigs4.c
This file contains the PHIGS specific routines for the example program4.

   Author: James Buckmiller  May 1990.
   Modified:  J. Buckmiller  Mar 1991.   Approved C binding

   Copyright (C) 1990, Evans & Sutherland

*/

#include <X11/Xlib.h>
#include <X11/phigs/phigs.h>
#include <X11/extensions/XPick.h>

#include "header4.h"



/*
Error_Check

This routine checks the global variable used to store error codes
returned from PEX.  If the error code is non-zero, it prints out a
diagnostic message and dies.
*/

void Error_Check(File, Line, Routine)
     char *File, *Routine;
     int Line;
{
  if (PEX_error)
    {
      fprintf(stderr, "(file %s; line %d):\n", File, Line);
      fprintf(stderr, "\t?unexpected PEX error %d in routine %s\n",
              PEX_error, Routine);
      exit(1);
    }
}
```

```
/*
StartPhigs

This routine is the top level routine that calls all supporting
routines in the logical order of a usual phigs routine ie
open PEX, setup the workstation parameters, define the phigs
structure and then go into the event loop.
*/

StartPhigs(dpy, win)        /* Routine to start phigs calls */
Display *dpy;
Window win;
{
   OpenPex(dpy);                /* Open PEX                                    */
   SetupWorkstation(dpy, win); /* Setup PHIGS workstation parameters          */
   Make_boxes();                /* Create Phigs structures                     */
   Get_events(dpy);             /* Event loop                                  */
   Cleanup();                 /* Cleanup phigs structures close workstation */
}


/*
   OpenPex

This routine Opens PEX on the display that was passed as an argument.
*/

OpenPex(dpy)
Display *dpy;
{
   Pxphigs_info  xinfo;
   unsigned long  infomask;

   xinfo.display = dpy;
   xinfo.rmdb    = NULL;
   xinfo.appl_id.name = NULL;
   xinfo.appl_id.class = NULL;
   xinfo.args.argc_p    = NULL;
   xinfo.args.argv      = NULL;
   xinfo.flags.no_monitor = 1;
   xinfo.flags.force_client_SS = 0;

   infomask = PXPHIGS_INFO_DISPLAY | PXPHIGS_INFO_FLAGS_NO_MON;

                        /* Open Pex */
```

```
    popen_xphigs((char*)NULL, PDEF_MEM_SIZE, infomask, &xinfo);

}

/*
SetupWorkstation

This routine opens a PHIGS workstation and sets up a Viewport.
The structure edit mode is set to insert elements and the display
update state is set to PWAIT.
Z buffering is enabled by calling psethlhsrmode
The nameset list is then filled with the name PICKABLE. This list is
used to set the picking inclusion filter. This same list will be used
in the addnameset call in Make_boxes to add the PICKABLE name to the
structure. The UNPICKABLE name is listed in the exclusion filter but
is not included in any display structures, thus nothing is excluded from
being picked in this example.
*/

SetupWorkstation(dpy, win)
Display *dpy;
Window   win;
{
    Pconnid_x_drawable  conn;
    Pview_rep3          vrep;              /* Declare viewporting variables */
    Pview_map3          map;
    Ppoint3             vrp, cntr;
    Pvec3               vup;
    Pvec3               vpn;

    conn.display     = dpy;
    conn.drawable_id = win;

                                  /* Open WS */
    popen_ws(WS, (Pconnid *) (&conn), phigs_ws_type_x_drawable);
                                  /* Setup viewport parameters */

    map.proj_type = PTYPE_PARAL;                    /* Set projection type */
    map.vp.x_min = 0.0; map.vp.x_max = 1.0;         /* Set viewport limits */
    map.vp.y_min = 0.0; map.vp.y_max = 1.0;
    map.vp.z_min = 0.0; map.vp.z_max = 1.0;

    map.win.x_min = -1.5; map.win.x_max = 1.5;      /* Set window limits   */
    map.win.y_min = -1.5; map.win.y_max = 1.5;
```

```
map.back_plane  = -2.0;    /* Set the front and back clipping planes */
map.front_plane =  1.0;
map.view_plane  =  0.0;    /* Set the location of the view plane     */

map.proj_ref_point.x = 0.0;          /* Set projection Reference point */
map.proj_ref_point.y = 0.0;           /* in VRC space                  */
map.proj_ref_point.z = 3.0;

vrep.xy_clip    = PIND_NO_CLIP; /* Turn Viewport clipping off          */
vrep.back_clip  = PIND_NO_CLIP; /* not to be confused with the         */
vrep.front_clip = PIND_NO_CLIP;/* clipping at the viewplanes!          */

vrep.clip_limit = map.vp;   /* Set Viewport clipping volume = viewport */

                                /* Setup View Reference Coordinates */

vrp.x = 0.0; vrp.y = 0.0; vrp.z = 1.0;     /* Set View ref point    */
                                /* Set view up vector    */
vup.delta_x = 0.0; vup.delta_y = 1.0; vup.delta_z = 0.0;
                                /* Set view plane normal*/
vpn.delta_x = 0.0; vpn.delta_y = 0.0; vpn.delta_z = 1.0;

peval_view_ori_matrix3(&vrp, &vpn, &vup,    /* Evaluate orient matrix */
              &PEX_error, vrep.ori_matrix);

SAFE_PEX("peval_view_ori_matrix3");          /* Check for error status */

peval_view_map_matrix3( &map, &PEX_error,    /* Evaluate map matrix    */
                        vrep.map_matrix);

SAFE_PEX("peval_view_map_matrix3");          /* Check for error status */

pset_view_rep3( WS, VIEW, &vrep );      /* Set the view representation  */


pset_edit_mode(PEDIT_INSERT);      /* Set edit mode to insert elements */
pset_disp_upd_st(WS, PDEFER_WAIT, PMODE_NIVE);
                                /* Set update state to WAIT */

pset_hlhsr_mode(1, PHIGS_HLHSR_MODE_ZBUFF);/* Enable WS Z buffering   */

nset_names.num_ints = 1;                    /* Name set list count      */
nset_names.ints = namesints;                /* Name set list of integers */
namesints[0] = PICKABLE;                    /* Put the PICKABLE name in  */
```

```
                                                /*  the name set list.       */
    pick_incl.number = 1;        /* Set Picking inclusion filter to 1 name  */
    pick_incl.integers = namesints;
                          /* Set the names list to be same as nameset*/

    pick_excl.number = 0;         /* Empty picking exclusion filter         */
    pick_excl.integers = NULL;

    XCreatePC(dpy, pick_PEX, &pc);                /* Create picking context */
    XSetPickFilters(dpy, pc, &pick_incl, &pick_excl); /* Set pick filters */

}

/*
Make_boxes

make_boxes defines a polyline cube and a fillarea cube in 3
dimensions. Structures POLYBOX and FILLBOX are defined to contain
these data elements along with color and style attributes to be applied
to the data elements. A higher level structure DISPLAY_STRUCT is defined
to include both the POLYBOX and FILLBOX structures and is then
posted to the open workstation to be displayed.
*/

Make_boxes()
{
Pmatrix3 identity;

                  /* Define polyline cube vectors */

    static Ppoint3     line_points1[]=     /* Define points for front face */
    {{  0.5, 0.5, 0.5} , { 1.0, 0.5, 0.5} ,
     {  1.0, 1.0, 0.5} , { 0.5, 1.0, 0.5} ,
     {  0.5, 0.5, 0.5}};

    static Ppoint3     line_points2[]=     /* Define points for back face */
    {{  0.5, 0.5, 0.0} , { 1.0, 0.5, 0.0} ,
     {  1.0, 1.0, 0.0} , { 0.5, 1.0, 0.0} ,
     {  0.5, 0.5, 0.0}};

    static Ppoint3     line_points3[]=      /* Define connecting line     */
    {{  0.5, 0.5, 0.5} , { 0.5, 0.5, 0.0}};

    static Ppoint3     line_points4[]=      /* Define connecting line     */
    {{  1.0, 0.5, 0.5} , { 1.0, 0.5, 0.0}};
```

```
static Ppoint3    line_points5[]=        /* Define connecting line      */
{{  1.0, 1.0, 0.5} , { 1.0, 1.0, 0.0}};

static Ppoint3    line_points6[]=        /* Define connecting line      */
{{  0.5, 1.0, 0.5} , { 0.5, 1.0, 0.0}};

                        /* Define solid cube faces */

static Ppoint3    fill_points1[]=     /* Define points for front face  */
{{  -0.5, -0.5, 0.5} , {  0.0, -0.5, 0.5} ,
 {   0.0,  0.0, 0.5} , { -0.5,  0.0, 0.5}};

static Ppoint3    fill_points2[]=     /* Define points for back face   */
{{  -0.5, -0.5, 0.0} , {  0.0, -0.5, 0.0} ,
 {   0.0,  0.0, 0.0} , { -0.5,  0.0, 0.0}};

static Ppoint3    fill_points3[]=     /* Define points for right face  */
{{  0.0, -0.5, 0.5} , { 0.0, -0.5, 0.0} ,
 {  0.0,  0.0, 0.0} , { 0.0,  0.0, 0.5}};

static Ppoint3    fill_points4[]=     /* Define points for left face   */
{{  -0.5, -0.5, 0.5} , { -0.5, -0.5, 0.0} ,
 {  -0.5,  0.0, 0.0} , { -0.5,  0.0, 0.5}};

static Ppoint3    fill_points5[]=     /* Define points for bottom face */
{{  -0.5, -0.5, 0.5} , {  0.0, -0.5, 0.5} ,
 {   0.0, -0.5, 0.0} , { -0.5, -0.5, 0.0}};

static Ppoint3    fill_points6[]=     /* Define points for top face    */
{{  -0.5, 0.0, 0.5} , {  0.0, 0.0, 0.5} ,
 {   0.0, 0.0, 0.0} , { -0.5, 0.0, 0.0}};

Ppoint_list3 Line_list[5], Fill_list[5];

pset_edit_mode(PEDIT_INSERT);  /* Set edit mode to insert elements    */
punpost_all_structs(WS);       /* Unpost to remove any old structures */
pdel_all_structs();             /* Delete any old structures          */

   popen_struct(POLYBOX);       /* Open line drawing structure        */
   pset_line_colr_ind(2);        /* Assign default index color 2 to lines*/
         Line_list[0].num_points = 5; /* Fill in number of points in
       list   */
   Line_list[0].points = line_points1; /* Pointer to point array */
   ppolyline3(&Line_list[0]);  /* Create a polyline element          */
```

```
    pset_line_colr_ind(3);     /* Assign default index color 3 to lines */

       Line_list[1].num_points = 5;
       Line_list[1].points = line_points2;
       ppolyline3(&Line_list[1]);

    pset_line_colr_ind(4);     /* Assign default index color 4 to lines */
       Line_list[2].num_points = 2;
       Line_list[2].points = line_points3;
       ppolyline3(&Line_list[2]);

    pset_line_colr_ind(5);     /* Assign default index color 5 to lines */
       Line_list[3].num_points = 2;
       Line_list[3].points = line_points4;
       ppolyline3(&Line_list[3]);

    pset_line_colr_ind(6);     /* Assign default index color 6 to lines */
       Line_list[4].num_points = 2;
       Line_list[4].points = line_points5;
       ppolyline3(&Line_list[4]);

    pset_line_colr_ind(7);     /* Assign default index color 7 to lines */
       Line_list[5].num_points = 2;
       Line_list[5].points = line_points6;
       ppolyline3(&Line_list[5]);

pclose_struct();               /* Close line drawing structure        */

popen_struct(FILLBOX);         /* Open filled drawing structure       */
   pset_int_style(PSTYLE_SOLID); /* Set interior style to be solid    */
   pset_int_colr_ind(7);       /* Assign default index color 7 to face */
   Fill_list[0].num_points = 4;  /* Fill in number of points in list  */
   Fill_list[0].points = fill_points1; /* Pointer to point array      */
   pfill_area3(&Fill_list[0]);    /* Create a fill area element       */

   pset_int_colr_ind(3);       /* Assign default index color 3 to face */
      Fill_list[1].num_points = 4;
      Fill_list[1].points = fill_points2;
      pfill_area3(&Fill_list[1]);

   pset_int_colr_ind(2);       /* Assign default index color 2 to face */
      Fill_list[2].num_points = 4;
      Fill_list[2].points = fill_points3;
      pfill_area3(&Fill_list[2]);
```

```
      pset_int_colr_ind(4);       /* Assign default index color 4 to face */
         Fill_list[3].num_points = 4;
         Fill_list[3].points = fill_points4;
         pfill_area3(&Fill_list[3]);

      pset_int_colr_ind(5);       /* Assign default index color 5 to face */
         Fill_list[4].num_points = 4;
         Fill_list[4].points = fill_points5;
         pfill_area3(&Fill_list[4]);

      pset_int_colr_ind(6);       /* Assign default index color 6 to face */
         Fill_list[5].num_points = 4;
         Fill_list[5].points = fill_points6;
         pfill_area3(&Fill_list[5]);

   pclose_struct();               /* Close filled drawing structure       */

   make_identity(identity);

   popen_struct(DISPLAY_STRUCT); /* Open the top level display structure */
      pset_view_ind(VIEW);          /* Set the view index to be used       */
      pset_hlhsr_id(PHIGS_HLHSR_ID_ON);     /* Turn on Z buffering         */
      padd_names_set(&nset_names);/* Add nameset for picking (PICKABLE) */
      plabel(TRANS);               /* Insert a label for future updates   */
      pset_local_tran3(identity, PTYPE_REPLACE);
                                    /* set transformation matrix*/
      pexec_struct(FILLBOX);        /* Include the filled square           */
      pexec_struct(POLYBOX);       /* Include the line drawn square       */
      pset_hlhsr_id(PHIGS_HLHSR_ID_OFF);  /* Turn off Z buffering          */
   pclose_struct();

   ppost_struct(WS, DISPLAY_STRUCT, 1.0); /* Post DISPLAY_STRUCT, prio 1 */

   pupd_ws(WS,PUPD_PERFORM);                /* Update the workstation      */
   pset_edit_mode(PEDIT_REPLACE);           /* Set edit mode to replace    */

}

/*
make_identity

This routine sets the passed matrix to be an  identity matrix.
*/
```

```
make_identity(matrix)
    Pmatrix3 matrix;
{
    matrix[0][0] = 1;
    matrix[0][1] = 0;
    matrix[0][2] = 0;
    matrix[0][3] = 0;
    matrix[1][0] = 0;
    matrix[1][1] = 1;
    matrix[1][2] = 0;
    matrix[1][3] = 0;
    matrix[2][0] = 0;
    matrix[2][1] = 0;
    matrix[2][2] = 1;
    matrix[2][3] = 0;
    matrix[3][0] = 0;
    matrix[3][1] = 0;
    matrix[3][2] = 0;
    matrix[3][3] = 1;
}

Cleanup()                          /* Cleanup routine when done   */
{
 punpost_all_structs(WS);          /* Unpost all structures on ws */
 pdel_all_structs();               /* Delete all structures */
 pclose_ws(WS);                    /* Close workstation     */
 pclose_phigs();                   /* Close PHIGS           */
}
```

header4.h

```
/*

   header4.h
This file contains header information for example 4 programs.

   Author: James Buckmiller  May 1990.
   Modified:  J. Buckmiller  Mar 1991.   Approved C binding

   Copyright (C) 1990, Evans & Sutherland

*/

#define WS              1
#define POLYBOX         1
#define FILLBOX         2
#define DISPLAY_STRUCT  3
#define VIEW            4
#define TRANS           5
#define SPACEKEYSYM    32
#define MAXDIALS        8          /* number of dials on dials box */
#define CHARS_PER_DIAL  8          /* size of dials labels */
#define SAFE_PEX(routine) Error_Check(__FILE__, __LINE__, routine)
#define DEG_TO_RAD(D)    ((3.14159265358 / 180.0) * (D))
#define DIALSCALE      .005        /* Dial scale value */
#define PICKABLE       10          /* name for inclusion filter */

extern Window myWin;
extern Display *dpy;
extern char *ProgramName;

extern int Major_op;               /* input extension values */
extern int First_ev;
extern int First_err;

extern Pint_list nset_names;       /* Pick list nameset declaration */
extern XPickFilter pick_incl, pick_excl; /* Pick filters declaration     */
extern Pint namesints[1];          /* Name arrays for nameset       */
extern PC pc;/* pick context                    */
extern Pint PEX_error;             /* For PEX error numbers */
extern Pmatrix3 identity;
```

Additional Information

General Computer Graphics

Foley, James D., Andries van Dam, Stephen K. Feiner, and John F. Hughes, *Computer Graphics - Principles and Practice, Second Edition*, Reading, MA: Addison-Wesley Publishing Company, 1990.

Newman, William M. and Robert F. Sproull, *Principles of Interactive Computer Graphics, Second Edition,* New York: McGraw-Hill Book Company, 1979.

Computer Graphics Standards

Information processing systems - Computer graphics - Programmer's Hierarchical Interactive Graphics System (PHIGS), Part 1 - functional description, International Standard, ISO/IEC 9592-1:1988(E).

Information processing systems - Computer graphics - Programmer's Hierarchical Interactive Graphics System (PHIGS), Part 2 - archive file format, International Standard, ISO/IEC 9592-2:1988(E).

Information processing systems - Computer graphics - Programmer's Hierarchical Interactive Graphics System (PHIGS), Part 3 - clear-text encoding of archive file, International Standard, ISO/IEC 9592-3:1988(E).

Information processing systems - Computer graphics - Programmer's Hierarchical Interactive Graphics System (PHIGS), Part 4 - Plus Luminère und Surfaces (PHIGS PLUS), DIS PHIGS PLUS, 14 February 1991, ISO/IEC 9592-4:199x.

Standards in the Computer Graphics Industry, Fairfax, VA: National Computer Graphics Association, 1989.

2. X Extensions

Table of Contents

2. X Extensions

Introduction

This chapter describes the following ESV Workstation extensions to the X server:

- X Input Extension
- X Picking Extension
- E&S Extension
  - X Overlay Functionality
  - X Multiscreen Functionality
  - X Video Timing Formats Functionality
  - X Miscellaneous Traversal Functionality

X Input Extension

The X Consortium developed the X Input Extension for what are called "extended devices." This section describes the X Input Extension functions and their implementation in the ESV Workstation. It does not contain a complete description of the X Input Extension. Additional information can be found in the on-line man pages. This section discusses the following topics:

- "Implementation Notes" provides the basic information necessary for using the X Input Extension on an ESV Workstation.
- "Function Descriptions" describes some of the key X Input Extension functions.
- "Macro Descriptions" describes some of the X Input Extension macros.
- "Event Records" describes some of the X Input Extension event records that can be received in an **XNextEvent** call.
- "Additional Structures" describes some of the structures used by the X Input Extension.

Implementation Notes

Extended Device Event Handling

There are six classes of extended devices: **KEY**, **BUTTON**, **VALUATOR**, **PROXIMITY**, **FOCUS**, and **FEEDBACK**. A physical device may provide more than one class of information; *e.g.*, a knob box may provide both **VALUATOR** and **FEEDBACK**.

Event Selection

Input event selection for extended devices requires a little more work than standard X events. There are four steps to selecting events for a set of devices.

1. Get the List of Available Devices

First, the application does an inquiry to determine the list of available devices. This is done with **XListInputDevices**. This returns a list of **XDeviceInfo** structures which contains information about the devices supported by an implementation of the X Input Extension. This information includes the ID, type, name, number of classes supported, and a list of class information. The list of input devices always includes the X pointer and X keyboard.

Following is a list of extension devices supported in the ESV Workstation implementation and their associated classes.

Name	Class
XI_BUTTONBOX	**BUTTON, FEEDBACK**
XI_KNOB_BOX	**VALUATOR, FEEDBACK**
XI_TABLET	**VALUATOR, BUTTON**
XI_SPACEBALL	**VALUATOR, BUTTON**

2. Open the Devices

Each device is opened for use by an application calling **XOpenDevice** with the device ID returned in the **XDeviceInfo** list. This procedure returns a list of **XDevice** structures containing specific class information.

The following example shows how to get a list of extended devices and open them.

```
Display *display;
Window window;
XDeviceInfo *devicelist, *dl;
XDevice *dials = NULL;
XDevice *tablet = NULL;
XDevice *space = NULL:
XDevice *button = NULL;
XID dials_id, tablet_id, space_id, button_id;
int ndev,i;
   .
   .
   .
devicelist = XListInputDevices(display, ndev);
dl = devicelist;
for (i = 0; i < ndev; i++, dl++)
  {
   if (!strcmp(XI_KNOBBOX, dl->name))
   {
    dials_id = dl->id;
    dials = XOpenDevice(display, dials_id);
   }
   if (!strcmp(XI_TABLET, dl->name))
   {
    tablet_id = dl->id;
    tablet = XOpenDevice(display, tablet_id);
   }
   if (!strcmp(XI_SPACEBALL, devices->name))
   {
    space_id = dl->id;
    space = XOpenDevice(display, space_id);
   }
   if (!strcmp(XI_BUTTONBOX, dl->name))
    button_id = dl->id;
    button = XOpenDevice(display, button_id);
  }
```

3. Find Event Types and Classes

X extension events are returned through a call to **XNextEvent** just like normal X events. However the event type is determined by the extension based on an internal event type that is offset by a server determined base. This means that the application must determine the event types in order to compare them to values returned by the call to **XNextEvent**. Also, in order to select events, the event class of the event must be determined.

The information that has been returned at this point can be supplied to macros to find the event types and event classes for each device. Some examples of these are **DeviceButtonPress**, **DeviceButtonRelease**, and **DeviceMotionNotify**. The macros available are listed on the **XOpenDevice** man page.

4. Select Extension Events

The event class information is used by **XSelectExtensionEvent** to select events from extension devices. The following example shows how to get the event classes and use them to select events.

```
    .
    .
    .
XEventClass *eventClass[2];
EventType *event_type[2];
int event_count = 0
int DeviceMotion = -1;
int DevicePress = -1;
    .
    .
    .
if (button)
{
  DeviceButtonPress(button, DevicePress,
    eventClass[event_count]);
  event_count++;
}
if (dials)
{
  DeviceMotionNotify(dials, DeviceMotion,
    eventClass[event_count]);
  event_count++;
}
XSelectExtensionEvent(display, window, eventClass,
    event_count);
```

Event Handling

Below is a table that lists each device, and the associated structure definition of events that are generated. The event type is not predefined as in standard X events. The values for the event types are generated from the above macros. The structures for each event type are described in "Event Records."

Device	Structure
BUTTONBOX	**XDeviceButtonEvent**
KNOB_BOX	**XDeviceMotionEvent**
TABLET	**XDeviceMotionEvent, XDeviceButtonEvent**
SPACEBALL	**XDeviceMotionEvent, XDeviceButtonEvent**

The following example shows how to process input extension events.

```
XEvent pe;
XDeviceMotionEvent *dm = (XDeviceMotionEvent *) & pe;
XDeviceButtonEvent *db = (XDeviceButtonEvent *) & pe;
   .
   .
   .
XNextEvent(dpy, &pe);
switch (pe.type)
{
  /* process normal X events with case xxx: */
  .
  .
  .
  /* process extension events */
  default:
    if (pe.type == DeviceMotion)
    {
      if (dm->deviceid == tablet_id)
      {
        /* Process tablet motion */
      }
      else if (dm->deviceid == space_id)
      {
        /* Process space ball motion */
      }
      else if (dm->deviceid == dials_id)
      {
        /* Process knob box motion */
      }
    }
    else if (pe.type == DevicePress)
    {
      if (db->deviceid == tablet_id)
      {
        /* Process tablet button press */
```

```
        }
        else if (db->deviceid == space_id)
        {
          /* Process space ball button press */
        }
        else if (db->deviceid == button_id)
        {
          /* Process button box button press */
        }
      }
      break;
  }
```

Feedback Control

Feedback is used for sending to an extension device. The two devices on the ESV Workstation that support it are the knob box and button box. The knob box has an 8 character display for each knob and the button box has an LED for each button. These devices are sent to using the **XChangeFeedbackControl** function.

The following example shows how this is done for the first knob label.

```
Display *display;
XDevice *dials;
XStringFeedbackControl feeder;
KeySym message[8];
char *label;
  .
  .
  .
/* the characters to display must be encoded into KeySyms */

feeder.class = StringFeedbackClass;
feeder.length = sizeof(XStringFeedbackControl) +
                (strlen(label) * sizeof(KeySym));
feeder.num_keysyms = strlen(label);
feeder.syms_to_display = message;
feeder.id = 0; /* knob 0*/
XChangeFeedbackControl(display, dials, DvString, &feeder);
.
.
.
```

Function Descriptions

XListInputDevices

Syntax

```
XDeviceInfo *
XListInputDevices (display, ndevices)
    Display         *display;
    int             *ndevices;
```

Arguments

display	The connection to the X server.
ndevices	The address of a variable into which the number of available input devices can be returned.

Description

The **XListInputDevices** procedure is used to determine the number and types of extension devices available for input. An array of **XDeviceInfo** structures is returned, with one element in the array for each device. The number of elements is returned in the ***ndevices*** argument.

XFreeDeviceList

Syntax

```
XFreeDeviceList (list)
    XDeviceInfo     *list;
```

Argument

list	The pointer to the **XDeviceInfo** array returned by a previous call to **XListInputDevices**.

Description

The **XFreeDeviceList** procedure frees the list of input device information.

XOpenDevice

Syntax

```
XDevice *
XOpenDevice (display, device_id)
    Display         *display;
    XID             device_id;
```

Arguments

display	The connection to the X server.
device_id	The ID that identifies the device to be opened. This ID is obtained from the **XListInputDevices** request.

Description

This procedure opens the device for the application and returns an **XDevice** structure if successful. The **XDevice** structure contains a pointer to an array of **XInputClassInfo** structures. Each element in that array contains information about events of a particular input class supported by the input device. A program can determine the event type and event class for a given event by using macros defined by the input extension.

XCloseDevice

Syntax

```
int
XCloseDevice (display, device)
    Display         *display;
    XDevice         *device;
```

Arguments

display	The connection to the X server.
device	The device to be closed.

Description

This function closes the device and frees the **XDevice** structure.

XSelectExtensionEvent

Syntax

XSelectExtensionEvent (*display*, *w*, *event_list*, *event_count*)

```
Display      *display;
Window       w;
XEventClass  *event_list;
int          event_count;
```

Arguments

display	The connection to the X server.
w	The window whose events you are interested in.
event_list	A pointer to a list of **XEventClass**es that specify which events are desired.
event_count	The number of elements in the **event_list**.

Description

This procedure requests that events matching the events and devices described by the event list are reported to the application. The elements of the **XEventClass** array are the **event_class** values returned by the Input Extension macros to retrieve the event classes.

XSelectExtensionEvent

Syntax

XChangeFeedbackControl(*display, device, mask, control*)
Display ***display;***
XDevice ***device;***
Mask ***mask;***
XFeedbackControl ***control;***

Arguments

display	The connection to the X server.
device	The device to be used for feedback.
mask	The mask specific to each feedback that describes how the feedback is to be modified.
control	The address of an **XFeedbackControl** structure that contains the new values for the feedback.

Description

This function is provided to manipulate those input devices that support feedback. A **BadMatch** error will be generated if the requested device does not support feedback. You can determine whether or not a given device supports feedback by examining the information returned by the **XOpenDevice** request. For those devices that support feedback, **XOpenDevice** will return an **XInputClassInfo** structure with the **input_class** field equal to the constant **FeedbackClass** (defined in the file **XI.h**). The feedback classes that are currently defined are: **KbdFeedbackClass**, **PtrFeedbackClass**, **StringFeedbackClass**, **IntegerFeedbackClass**, **LedFeedbackClass**, and **BellFeedbackClass**.

An input device may support zero or more feedback classes, and may support multiple feedbacks of the same class. Each feedback contains a class identifier and an ID that is unique within that class for that input device. The ID is used to identify the feedback when making an **XChangeFeedbackControl** request.

The **XChangeFeedbackControl** function modifies the values of one feedback on the specified device. The feedback is identified by the ID field of the **XFeedbackControl** structure that is passed with the request. The fields of the feedback that are to be modified are identified by the bits of the mask that is passed with the request. **XChangeFeedbackControl** can generate a **BadDevice**, **BadMatch**, or **BadValue** error.

Macro Descriptions

You can determine the event type and event class for a given event by using the macros defined below. The event type is used to check the ***type*** field of an event generated by **XNextEvent**.

The name of the macro corresponds to the desired event, and the macro is passed to the structure that describes the device from which input is desired.

- **DeviceButtonPress(*d, type, class*)**

  Returns the event type and class of button press events for the **BUTTON** device class. The parameter ***d*** is a pointer to an **XDevice** structure for a **BUTTON** class device.

- **DeviceButtonRelease(*d, type, class*)**

  Returns the event type and class of button release events for the **BUTTON** device class. The parameter ***d*** is a pointer to an **XDevice** structure for a **BUTTON** class device.

- **DeviceMotionNotify(*d, type, class*)**

  Returns the event type and class of motion events for the **VALUATOR** device class.The parameter ***d*** is a pointer to an **XDevice** structure for a **VALUATOR** class device.

Event Records

Device Button Events

DeviceButtonPressed/DeviceButtonReleased events are generated when a key is pressed or released on a **BUTTON** extension device and that type of event is selected. The structure associated with the **DeviceButtonPressed/ DeviceButtonReleased** event is defined as follows:

```
typedef struct {
    int             type;
    unsigned long   serial;
    Bool            send_event;
    Display         *display;
    Window          window;
    XID             deviceid;
    Window          root;
    Window          subwindow;
    Time            time;
    int             x, y;
    int             x_root, y_root;
    unsigned int    state;
    unsigned int    button;
    Bool            same_screen;
    unsigned int    device_state;
    unsigned char   axes_count;
    unsigned char   first_axis;
    int             axis_data[6];
} XDeviceButtonEvent;
typedef XDeviceButtonEvent XDeviceButtonPressedEvent;
typedef XDeviceButtonEvent XDeviceButtonReleasedEvent;
```

Device Motion Events

The **DeviceMotion** event is generated when there is motion on a **VALUATOR** extension device. The structure associated with the **DeviceMotion** event is defined as follows:

```
typedef struct {
    int             type;
    unsigned long   serial;
    Bool            send_event;
    Display         *display;
    Window          window;
    XID             deviceid;
    Window          root;
    Window          subwindow;
    Time            time;
    int             x, y;
    int             x_root, y_root;
    unsigned int    state;
    char            is_hint;
    Bool            same_screen;
    unsigned int    device_state;
    unsigned char   axes_count;
    unsigned char   first_axis;
    int             axis_data[6];
} XDeviceMotionEvent;
```

Additional Structures

These are some of the input structures referenced by the X input Extension.

```
typedef struct _XAnyClassInfo *XAnyClassPtr;

typedef struct _XAnyClassInfo {
    XID             class;
    int             length;
} XAnyClassInfo;

typedef struct _XDeviceInfo *XDeviceInfoPtr;

typedef struct _XDeviceInfo {
    XID             id;
    Atom            type;
    char            *name;
    int             num_classes;
    int             use;
    XAnyClassPtr    inputclassinfo;
} XDeviceInfo;

typedef struct _XButtonInfo *XButtonInfoPtr;

typedef struct _XButtonInfo {
    XID             class;
    int             length;
    short           num_buttons;
} XButtonInfo;

typedef struct _XAxisInfo *XAxisInfoPtr;

typedef struct _XAxisInfo {
    int             resolution;
    int             min_value;
    int             max_value;
} XAxisInfo;

typedef struct _XValuatorInfo *XValuatorInfoPtr;
```

```
typedef struct _XValuatorInfo {
    XID             class;
    int             length;
    unsigned char   num_axes;
    unsigned char   mode;
    unsigned long   motion_buffer;
    XAxisInfoPtr    axes;
} XValuatorInfo;

typedef struct {
    unsigned char   input_class;
    unsigned char   event_type_base;
} XInputClassInfo;

typedef struct {
    XID             device_id;
    int             num_classes;
    XInputClassInfo *classes;
} XDevice;
```

X Picking Extension

PHIGS does not interact well with the X Window System. The PHIGS input model is not compatible with the X input model. If you use an output-only PEX workstation and find the workstation's window ID, you can use X input functions with PHIGS output functions. This allows your application to receive input events, but leaves you with no way to do picking on the displayed PEX data structure. The picking extension described in this chapter is a solution to this problem.

This picking extension allows picking operations on PEX graphics displayed in an X window to report their results as events back to your application.

The data returned by the picking extension is more detailed than that defined by PEX. The pick data returned by the picking extension can include the following:

- A window space coordinate on the picked primitive, which is within the pick box
- A model space coordinate on the picked primitive, which is within the pick box
- An index for identifying which element of the picked primitive was picked (*e.g.*, the line item of a polyline primitive or the character within a text string)
- The parameterized values which identify a point on a NURB curve or NURB surface

Overview

Graphics applications often encounter the need for the user to identify some graphics element from its position on the display device. The X Picking Extension addresses the need for identifying PEX graphics elements in PHIGS workstation windows.

A graphics application initiates a pick operation by specifying the window of a PHIGS workstation, a picking context (PC), and the location about which the pick box is to be centered. The graphics subsystem is then responsible for determining the picked graphics primitive. It does this by traversing all posted structures to the PHIGS workstation and determining which of them fall (at least some portion) within the pick box, and whether they are pickable according to the inclusion and exclusion filters of the PC. From this set of graphics primitives, one is returned to the graphics application according to the mode of the PC.

Information about that graphics primitive is returned to the application in an X extension event form. The information that can be returned includes the

screen and model coordinates of the primitive, the element number and structure ID of the primitive, the current pick ID at the moment the primitive was traversed, and the complete path (element, struct, pickid) to the top of the posted structure.

The X event mechanism is used to prevent delaying of the application while the graphics subsystem is traversing the posted structures.

A prepick operation is also available through the X Picking Extension. It can be used to highlight the set of graphics primitives which may be returned as the result of a pick operation. No events are generated as the result of a prepick operation.

A PC is used in the server to maintain state information that affects pick operations. This is analogous to a graphics context (GC) used in X drawing operations. The PC retains the settings of the following:

- *pick mode* - determines selection criteria
- *pick box size* - width and height of pick box
- *pick highlighting color* - pixel value for highlighting
- *pick highlighting mode* - amount of primitive to highlight
- *pick inclusion filter* - set of allowed names when picking
- *pick exclusion filter* - set of disallowed names when picking
- *pick return values* - hints to avoid wasted computation

In a similar manner to the use of a GC, a PC must be created before it can be used. Values of the PC may then be set to any of the legal values specified by this document. An identifier for the PC is always specified for pick or prepick operations. A PC should be destroyed by calling **XFreePC** when it is no longer needed. An application may create as many PCs as it requires. More than one pick or prepick operation may be concurrently pending on the same PC.

X Picking Data Structures

The picking extension routines use the following data structures and types:

- **PC (Pick Context)**
- **XPickEvent**
- **XPickFilter**
- **XPickPath**
- **XPickPathEvent**
- **XPickPathItem**

PC (Pick Context)

Syntax

typedef *XID* PC;

Description

The state of a pick operation is associated with a unique pick context identifier. All state information needed by the server is associated with a pick context. More than one pick request can be pending a pick context at any one time.

XPickEvent

Syntax

```
typedef struct {
        int             type;
        unsigned long   serial;
        Bool            send_event;
        Display         *display;
        PC              pc;
        Window          window;
        Time            time;
        int             prim_type;
        int             valid_flags;
        int             itemid;
        int             elementid;
        int             structureid;
        int             pickid;
        float           modelx;
        float           modely;
        float           modelz;
        float           screenx;
        float           screeny;
        float           screenz;
        float           prim_spcl1;
        float           prim_spcl2;
} XPickEvent;
```

Arguments

type	Equal to the value returned in the second argument of **XGetPickEventType**.
serial	The serial number of the event.
send_event	True if this event was generated by a **XSendEvent** call.
display	The display that generated the event.
pc	The ID of the pick context which generated this event. Multiple picks can be active at one time. Sending the **pc** back lets you know which pick completed.
window	The window in which the event was generated.

time The time stamp of the event (when it happened).

prim_type The type of the picked PEX primitive. It is set to **pick_nopick** if nothing was picked. Otherwise, it is set to one of

pick_marker
pick_marker2d
pick_text
pick_text2d
pick_annotationtext
pick_annotationtext2d
pick_polyline
pick_polyline2d
pick_polylinesetdata
pick_fillarea
pick_fillarea2d
pick_fillareaset
pick_fillareaset2d
pick_fillareadata
pick_fillareasetdata
pick_trianglestrip
pick_quadrilateralmesh
pick_setfillareasetdata
pick_cylinder
pick_sphere

valid_flags Indicates which of the following fields in the **XPickEvent** structure are valid. These flags may consist of any mixture of

pick_screenpt_valid
pick_modelpt_valid
pick_primspcl1_valid
pick_primspcl2_valid

If it is possible to determine the screen coordinates of the point on the picked primitive which is within the pick box, **pick_screenpt_valid** will be set.

If it is possible to determine the model coordinates of the point on the picked primitive which is within the pick box, **pick_modelpt_valid** will be set. If the current transformation matrix is not invertible or if the model point calculation has been turned off in the pick context by the client, **pick_modelpt_valid** will not be set and the values in **modelx**, **modely**, and **modelz** will not be usable.

If valid data exists in **prim_spcl1** for the picked primitive, **pick_primspcl1_valid** will be set. If valid data exists in **prim_spcl2** for the picked primitive, **pick_primspcl2_valid** will be set.

itemId The item number of the picked structure. The interpretation of the item number depends upon the primitive type. The table below describes how to interpret the value for the various types of primitives. The primitive's *index* tells which primitive in the PHIGS element was picked.

Primitive Type	Description
pick_marker	index of the picked marker index starts with 0
pick_marker2d	index of the picked marker index starts with 0
pick_text	index of the picked character index starts with 0
pick_text2d	index of the picked character index starts with 0
pick_annotationtext	index of the picked character index starts with 0
pick_annotationtext2d	index of the picked character index starts with 0
pick_polyline	index of the picked line segment index starts with 1
pick_polyline2d	index of the picked line segment index starts with 1
pick_polylineset	index of the picked line segment index starts with 1
pick_fillarea	always set to zero
pick_fillarea2d	always set to zero
pick_fillareaset	always set to zero
pick_fillareaset2d	always set to zero
pick_fillareadata	always set to zero
pick_fillareasetdata	always set to zero
pick_trianglestrip	index of the picked triangle index starts with 0

pick_quadrilateralmesh	index of the picked mesh index starts with 0
pick_setfillareasetdata	always set to zero
pick_cylinder	index of the picked cylinder index starts with 1
pick_sphere	index of the picked sphere index starts with 0

elementId The element ID of the picked structure.

structureId The structure ID of the picked structure.

pickId The pick ID associated with the structure.

modelx
modely
modelz The 3D coordinates of the pick point in model space. If the picked primitive is 2-dimensional, the z-value will be set to zero.

screenx
screeny
screenz The pixel screen coordinates of the pick point relative to the window in which the pick was performed. The value of ***screenz*** is implementation dependent.

Note: The terms ***screenx*** and ***screeny*** are misnomers because their values are relative to the window and not to absolute screen coordinates.

prim_spcl1 Reserved for future enhancements. Until then, **pick_primspcl1_valid** will never be set in **valid_flags**.

prim_spcl2 Reserved for future enhancements. Until then, **pick_primspcl2_valid** will never be set in **valid_flags**.

Description

An **XPickEvent** is returned for each **XPick()** call made by a client application. If no graphics primitives were in the pick box, or if no graphics primitives within the pick box were pickable, the value of ***prim_type*** will be set to **pick_nopick**. Otherwise, one item will be returned as the result of the pick operation and the elements of the **XPickEvent** structure will be set as indicated.

Note that the client application must have selected to receive **XPickEvents** from the window in which the pick operation is to occur.

XPickFilter

Syntax

```
typedef struct {
        int         number;
        int         *integers;
} XPickFilter;
```

Description

Pick filters are used to specify which elements will be included or excluded during a pick operation. The **XPickFilter** structure is the same as a PHIGS **PintIst**. Only the names have been changed.

XPickPath

Syntax

```
typedef struct {
        Display         *display;
        Window          window;
        PC              pc;
        Window          subwindow;
        int             prim_type;
        int             length;
        int             itemid;
        XPickPathItem   *path;
} XPickPath;
```

Arguments

pc	The ID of the pick context which generated the **XPickPathEvents** used to construct this pick path. Multiple picks can be active at one time. Sending the ***pc*** back lets you know which pick completed.
prim_type	The same as in the **XPickEvent** description.
length	The number of **XPickPathItem** records in the pick path. If nothing was picked, the length will be zero.
itemid	The item number within an element of a structure that was actually picked. This is the same as in the **XPickEvent** description. This is only meaningful for the last element in a path so it is returned separately from the rest of the path.
path	A pointer to a vector of **XPickPathItem** records. The last item in the vector is the picked primitive.

Description

An **XPickPath** structure is returned by the **XPickPathEventToPath** routine when it has assembled a complete pick path from a group of **XPickEvent** records.

XPickPathEvent

Syntax

```
typedef struct {
        int             type;
        unsigned long   serial;
        Bool            send_event;
        Display         *display;
        PC              pc;
        Window          window;
        Time            time;
        int             prim_type;
        int             itemid;
        int             path_length;
        int             first_path;
        XPickPathItem   path[4];
} XPickPathEvent;
```

Arguments

type	The type of the event. It is equal to the value returned in the third argument to **XGetPickEventType**.
serial	The serial number of the event.
send_event	True if the event was generated by an **XSendEvent** call.
display	The display that generated the event.
pc	The ID of the pick context which generated this event. Multiple picks can be active at one time. Sending the ***pc*** back lets you know which pick completed.
window	The window in which the event was generated.
time	The time stamp of the event (when it happened).
prim_type	The same as in the **XPickEvent** description.
itemid	The item number within an element of a structure that was actually picked. This is the same as in the **XPickEvent** description. This is only meaningful for the last element in a path so it is returned separately from the rest of the path.
path_length	The total length of the path.; one greater than the highest path item index. A path length of 0 indicates that nothing was picked.

first_path The position in the pick path of the first item in the path. For example, if ***path_length*** is 6 and ***first_path*** is 4 then this event contains the 5th and 6th items in the path. Paths items are numbered starting at 0.

path A vector of up to four path items. It contains a portion of the actual path data.

Description

The complete pick path is reported to the client as one or more **XPickPathEvents**. If the path length to the picked primitive is more than four, multiple **XPickPathEvents** are sent to the client as needed to return the complete path.

XPickPathItem

Syntax

```
typedef struct {
        int             elementid;
        int             structureid;
        int             pickid;
} XPickPathItem;
```

Arguments

elementId	The element number of the item within the PHIGS structure.
structureId	The ID of the PHIGS structure.
pickId	An ID associated with the picked structure. It is the current pick ID for the element in the **XPickPathItem**.

Description

An **XPickPathItem** structure identifies a structure in the pick path.

Interface Routines

All interface routines return an integer value to indicate success or failure of the call. They return a success if the call worked or an error code indicating the type of failure.

The interface routines include the following:

- **XCreatePC**
- **XFreePC**
- **XFreePickFilter**
- **XFreePickPath**
- **XGetPickBoxSize**
- **XGetPickEventType**
- **XGetPickFilters**
- **XGetPickHighlightingColor**
- **XGetPickHighlightingMode**
- **XGetPickMode**
- **XGetPickReturnVals**
- **XGetSelectedPickEvents**
- **XPick**
- **XPickPathEventToPath**
- **XPrePick**
- **XSelectPickEvents**
- **XSetPickBoxSize**
- **XSetPickFilters**
- **XSetPickHighlightingColor**
- **XSetPickHighlightingMode**
- **XSetPickMode**
- **XSetPickReturnVals**

XCreatePC

Syntax

int

XCreatePC(*dpy*, *type*, *pc*);

Display	***dpy**;*
int	***type**;*
PC	***pc**;*

Arguments

dpy	The display on which you are going to perform a pick.
type	The type of graphics structure on which you are going to perform pick. One of the following defined constants. **pick_PEX** Currently only PHIGS structures can be picked. We expect that other graphics interfaces will also need a way to report picking information in an X compatible way.
pc	Where you want the new pick context identifier stored.

Description

An **XCreatePC** call returns a new PC that you can use in an **XPick** or **XPrePick** call. The new PC has the following default values which can be changed and retrieved:

- pick mode = **pick_first**
- return values = **pick_compute_modelpt | pick_compute_special**
- pick box size = 9 x 9 pixels
- highlighting color = white
- highlighting mode = **pick_highlighting_off**
- inclusion filter = none
- exclusion filter = none

Errors

BadAlloc

BadValue (returned if ***type*** is not **pick_PEX**)

XFreePC

Syntax

```
int
XFreePC(dpy, pc);
        Display         *dpy;
        PC              *pc;
```

Arguments

dpy	The display on which the PC was created.
pc	The PC to be freed.

Description

An **XFreePC** call removes a PC from the server and frees all related resources.

Errors

BadPC

XFreePickFilter

Syntax

int
XFreePickFilter(*filter*)
XPickFilter *filter;

Argument

filter A pointer to an **XPickFilter** structure.

Description

An **XFreePickFilter** call releases the storage used by a pick filter. It is a safe way to free the storage.

XFreePickPath

Syntax

```
int
XFreePickPath(path)
        XPickPath       *path;
```

Argument

path	A pointer to an **XPickPath** structure.

Description

An **XFreePickPath** call releases the storage used by a pick path. It is a safe way to free the storage.

XGetPickBoxSize

Syntax

```
int
XGetPickBoxSize(dpy, pc, width, height)
        Display         *dpy;
        PC              pc;
        int             *width;
        int             *height;
```

Arguments

dpy	The display on which the PC was created.
pc	A pick context whose value you are querying.
width	Where to put the width of the pick box.
height	Where to store the height of the pick box.

Description

An **XGetPickBoxSize** call gets the pick box size from a PC. See **XSetPickBoxSize** for more information.

Errors

BadPC

XGetPickEventType

Syntax

```
int
XGetPickEventType(dpy, PickEventType, PickPathEventType)
        Display         *dpy;
        int             *PickEventType;
        int             *PickPathEventType;
```

Arguments

dpy	The display on which you are going to perform a pick.
PickEventType	Where you want the type of a **PickEvent** to be stored.
PickPathEventType	Where you want the type of a **PickPathEvent** to be stored.

Description

An **XGetPickEventType** call returns the type of an **XPickEvent** and an **XPickPathEvent**. This information is needed so that the programmer can identify these events when they are returned from **XNextEven**t.

XGetPickFilters

Syntax

```
int
XGetPickFilters(dpy, pc, inclusion, exclusion)
        Display         *dpy;
        PC              pc;
        XPickFilter     **inclusion;
        XPickFilter     **exclusion;
```

Arguments

dpy	The display on which the PC was created.
pc	A pick context whose value you are querying.
inclusion	A pointer to a pointer to an **XPickFilter** structure. It will be set to point to the current value of the inclusion filter.
exclusion	A pointer to a pointer to an **XPickFilter** structure. It will be set to point to the current value of the exclusion filter.

Description

An **XGetPickFilters** call returns the picking filters of a picking context. **XFreePickFilter** should be called to release the storage used for each of the inclusion and exclusion filters

Errors

BadPC

XGetPickHighlightingColor

Syntax

Int

XGetPickHighlightingColor(*dpy, pc, color*)

Display	***dpy;**
PC	***pc;**
unsigned long	***color;**

Arguments

dpy	The display on which the PC was created.
pc	A pick context whose value you are querying
color	Where to store the pixel value.

Description

An **XGetPickHighlightingColor** call gets the highlighting color from a PC. See **XSetPickHighlightingColor** for more information.

Errors

BadPC

XGetPickHighlightingMode

Syntax

int
XGetPickHighlightingMode(*dpy*, *pc*, *mode*)

```
Display     *dpy;
PC          pc;
int         *mode;
```

Arguments

dpy	The display on which the PC was created.
pc	A pick context whose value you are querying.
mode	Where to store the returned mode value.

Description

An **XGetPickHighlightingMode** call gets the current highlighting mode from a PC. See **XSetPickHighlightingMode** for more information.

Errors

BadPC

XGetPickMode

Syntax

```
int
XGetPickMode(dpy, pc, mode)
        Display         *dpy;
        PC              pc;
        int             *mode;
```

Arguments

dpy	The display on which the PC was created.
pc	A pick context whose value you are querying.
mode	Where to put the returned value.

Description

An **XGetPickMode** call gets the picking mode from a PC. See **XSetPickMode** for more information.

Errors

BadPC

XGetPickReturnVals

Syntax

XGetPickReturnVals(*dpy*, *pc*, *mask*)

Display	***dpy**;
PC	***pc**;
unsigned long	***mask**;

Arguments

dpy	The display on which you are going to perform a pick.
pc	A pick context whose value you are changing.
mask	A pointer to a long word in which to store the current values.

Description

An **XGetPickReturnVals** call gets the current value of the computation mask for the specified picking context. A computation mask serves as a hint to the server to avoid unnecessary additional computation when returning the details of a completed picking operation. See **XSetPickReturnVals** for more information.

Errors

BadPC

XGetSelectedPickEvents

Syntax

```
Int
XGetSelectedPickEvents(dpy, win, mask)
        Display         *dpy;
        Window          win;
        unsigned long   *mask;
```

Arguments

dpy	The display on which you have selected picking events.
win	A window from which you have selected picking events.
mask	Where to store the pick event mask.

Description

An **XGetSelectedPickEvents** call gets the pick events mask from a window. The pick events mask determines which type of picking events will be sent to the window as the result of a pick operation.

Errors

BadWindow (returned if the window doesn't exist or isn't being used to display 3D graphics)

XPick

Syntax

Int
XPick(*dpy, win, pc, x, y*)

Display	***dpy;**
Window	***win;**
PC	***pc;**
Int	***x;**
Int	***y;**

Arguments

dpy	The display on which you are going to perform a pick.
win	The window in which you are going to pick.
pc	The ID of the pick context for the pick operation. All the parameters that control how a pick or prepick operation is done are taken from the pick context.
x	The *x* coordinate in pixels of the pick location relative to the window where the pick operation is to occur.
y	The *y* coordinate in pixels of the pick location relative to the window where the pick operation is to occur.

Description

An **XPick** call picks at the specified location in the specified window using the parameters associated with the specified pick context.

XPick initiates a pick operation for the specified window according to the parameters set in the specified ***pc***. A pick event and one or more pick path events will always be generated at the completion of the pick operation. They will be reported to the window on which the operation was performed if the setting of the pick events mask for that window has selected that event type.

The pick operation causes a traversal of all posted structures on the PHIGS workstation associated with that window. Each of the drawn primitives is analyzed to determine if any portion of it falls within the pick box centered at the specified *x* and *y*. If two or more primitives fall within the bounds of the pick box, the mode of the ***pc*** determines which of the primitives to return as the result of the pick operation. (Refer to **SetPickMode**.)

Errors

BadAlloc (returned if the server is out of memory)
BadPC
BadWindow (returned if the window doesn't exist or isn't being used to display 3D graphics)

XPickPathEventToPath

Syntax

XPickPath
***XPickPathEventToPath(*event*)**
XPickPathEvent *event;

Argument

event	A pointer to an **XPickPathEvent**.

Description

XPickPathEventToPath is a utility function that can be used to merge a number of separate **XPickPathEvent** records into a single **XPickPath** structure. Because there is no real limit to the length of a pick path, one or more pick path events must be sent to transmit the entire path.

XPrePick

Syntax

```
int
XPrePick(dpy, win, pc, x, y)
        Display     *dpy;
        Window      win;
        PC          pc;
        int         x;
        int         y;
```

Arguments

dpy	The display on which you are going to perform a pick.
win	The window in which you are going to pick.
pc	The ID of the pick context for the pick operation. All the parameters that control how a pick or prepick operation is done are taken from the pick context.
x	The *x* coordinate in pixels of the prepick location relative to the window where the pick operation is to occur.
y	The *y* coordinate in pixels of the prepick location relative to the window where the pick operation is to occur.

Description

An **XPrePick** call prepicks at the specified location in the specified window using the parameters associated with the specified pick context. No pick events are generated. Pick highlighting will be performed according to the pick highlighting mode and pick highlighting color of the specified **pc**.

XPrePick can be used by the graphics application as a visual aid to show the user what primitives fall within the pick box boundaries. All graphical output primitives that are 1) within the pick box, 2) visible, and 3) pickable, according to the value of the current name set and the inclusion/exclusion filters, will be drawn in the highlight color.

Errors

BadAlloc (returned if the server is out of memory)

BadPC

BadWindow (returned if the window doesn't exist or isn't being used to display 3D graphics)

XSelectPickEvents

Syntax

```
int
XSelectPickEvents(dpy, win, mask)
        Display         *dpy;
        Window          win;
        unsigned long   mask;
```

Arguments

dpy	The display on which you are going to perform a pick.
win	The window in which you are going to pick.
mask	The bitwise logical OR of zero or more of the following: **PickMask**, **PickPathMask**.
	A mask value of zero means send no events. Having the **PickMask** bit set will cause picking events to be sent. Having the **PickPathMask** bit set will cause pick path events to be sent.

Description

An **XSelectPickEvents** call selects the events to be sent by the server when a pick occurs. By default, no events will be sent unless they have been selected before a pick is requested.

The pick events mask for a window is analogous to the events mask that controls selection of X core events such as **ButtonPress** or **Exposure**. A limitation of the current implementation is that pick events and pick path events may not be propagated up the window hierarchy. Pick events and pick path events may only be reported to the window associated with the PHIGS workstation for the pick operation.

Errors

BadWindow (returned if the window doesn't exist or isn't being used to display 3D graphics)

BadValue (returned if mask bits other than those specified by **PickMask** and **PickPathMask** are set)

XSetPickBoxSize

Syntax

Int

XSetPickBoxSize(*dpy, pc, width, height*)

Display	***dpy;**
PC	***pc;**
Int	***width;**
Int	***height;**

Arguments

dpy	The display on which you are going to perform a pick.
pc	A pick context whose value you are changing.
width	The width of the pick box in pixels.
height	The height of the pick box in pixels.

Description

An **XSetPickBoxSize** call sets the size of the picking box. The default pick box size is 9 pixels by 9 pixels.

The pick box boundaries need not fall evenly on pixel boundaries. The **screenx** and **screeny** values returned in an **XPickEvent** are calculated at the precision of the floating point hardware in the graphics subsystem.

Errors

BadPC

BadValue (returned if the width or height of the pickbox is less than zero or greater than the size of the screen)

XSetPickFilters

Syntax

int

XSetPickFilters(*dpy, pc, inclusion, exclusion*)

Display	***dpy;**
PC	***pc;**
XPickFilter	***inclusion;**
XPickFilter	***exclusion;**

Arguments

dpy	The display on which the PC was created.
pc	A pick context whose value you are changing.
inclusion	The new inclusion filter.
exclusion	The new exclusion filter.

Description

An **XSetPickFilters** call is used to set a PHIGS-compatible picking filter in a picking context. A drawing primitive will be pickable if the intersection of the current name set with the inclusion filter is not empty and the intersection of the current name set with the exclusion filter is empty. Refer to the ANSI PHIGS specification for more information on name sets.

Errors

BadPC

BadValue (returned if the number of names in a pick filter is negative or greater than an implementation specific limit)

XSetPickHighlightingColor

Syntax

Int
XSetPickHighlightingColor(*dpy, pc, color*)

Display	***dpy**;
PC	***pc**;
unsigned long	***color**;

Arguments

dpy	The display on which you are going to perform a pick.
pc	A pick context whose value you are changing.
color	An X pixel value that will determine the highlighting color used during a structure traversal (in the form 0xffffff).

Description

An **XSetPickHighlightingColor** call sets the pick highlighting color. If highlighting is turned on, pickable items will be highlighted in this color. The default highlighting color is white.

Errors

BadPC

XSetPickHighlightingMode

Syntax

Int

XSetPickHighlightingMode(*dpy, pc, mode*)

Display	***dpy;**
PC	***pc;**
Int	***mode;**

Arguments

dpy	The display on which you are going to perform a pick.
pc	A pick context whose value you are changing.
mode	One of the following defined constants: **pick_highlighting_off**, **pick_highlighting_item**, **pick_highlighting_command**. **pick_highlighting_off** turns off all pick highlighting. **pick_highlighting_item** causes the marker, line segment, or polygon that was picked to be highlighted. **pick_highlighting_command** causes all or part of a structure element to be redrawn highlighted. How much is redrawn is implementation dependent. It is expected that other highlighting modes will be defined.

Description

An **XSetPickHighlightingMode** call sets the pick highlighting mode and turns pick highlighting on or off. This pick highlighting mode controls the highlighting mode for both pick and prepick operations. Note that a prepick operation initiated when the highlighting mode is off is a rather expensive no-op.

Errors

BadPC

BadValue (returned if mode is not one of the defined values)

XSetPickMode

Syntax

```
int
XSetPickMode(dpy, pc, mode)
  Display      *dpy;
  PC           pc;
  int          mode;
```

Arguments

dpy	The display on which you are going to perform a pick.
pc	A pick context whose value you are changing.
mode	One of the following defined constants: **pick_first**, **pick_last**, **pick_near**, or **pick_far**. If the mode is **pick_first**, then the first pickable item encountered in structure posting is picked. If the mode is **pick_last**, then the last pickable item encountered is picked. If the mode is **pick_near**, then the pickable item with the largest *z*-value is picked. If the mode is **pick_far**, then the pickable item with the smallest *z*-value is picked.

Description

An **XSetPickMode** call sets the picking mode. This controls which of the pickable items are reported after a pick operation is initiated.

Errors

BadPC

BadValue (returned if mode is not one of the defined values)

XSetPickReturnVals

Syntax

XSetPickReturnVals(*dpy, pc, mask*)

Display	***dpy**;
PC	***pc**;
unsigned long	***mask**;

Arguments

dpy	The display on which you are going to perform a pick.
pc	A pick context whose value you are changing.
mask	The logical OR of zero or more the following bits: **pick_compute_modelpt, pick_compute_special.**

If **pick_compute_modelpt** is set, the server will attempt to calculate a model space point on the picked primitive which is within the pick box. If the point cannot be calculated, **pick_modelpt_valid** in the **valid_flags** of the **XPickEvent** will not be set. If **pick_compute_modelpt** is not set, the server may not do the extra work necessary to compute the model space point, and **pick_modelpt_valid** in the **valid_flags** of the **XPickEvent** will not be set.

pick_compute_special is a value reserved for future enhancement of the picking extension.

Description

An **XSetPickReturnVals** call sets the desired values to return in an **XPickEvent**. Some of the calculations involved in the **XPickEvent** structure are expensive to compute. Computation of the model space point can involve inversion of the transformation matrix. If the client application does not need this data, it need not pay the additional computational penalty. This call can be used to inform the server that the model space point or the primitive special data will not be needed and needn't be calculated.

Errors

BadPC

BadValue (returned if the mask is not an allowable value)

X Overlay Functionality

The X Overlay Functionality provides easy access to the ESV Workstation overlay hardware. An example program demonstrating the use of overlay planes on the ESV Workstation is found in **/usr/people/fstest/demo/overlay.c**.

XAllocOverlayPlanes

Syntax

Status
XAllocOverlayPlanes(*dpy*, *win*, *planes*)
Display ***dpy**;
Window ***win**;
unsigned int ***planes**;

Description

XAllocOverlayPlanes allocates overlay planes for a specific window. On the ESV Workstation, the total number of overlay planes that can be allocated is four.

These planes are shared with planes that identify PHIGS workstations. Each plane allocated for overlay reduces the total number of PHIGS workstations by one-half. If all four are allocated for overlay, a maximum number of 12 PHIGS workstations will be available. They can be allocated for overlay from one to four.

If too many planes are requested or no planes can be allocated, a **BadValue** or **BadAlloc** error is returned.

Valid overlay pixel values for the window will be 0 to (2^n-1), where **n** is the number of overlay planes.

XFreeOverlayPlanes

Syntax

Status
XFreeOverlayPlanes(*dpy*, *win*)
Display ***dpy**;
Window ***win**;

Description

XFreeOverlayPlanes frees all of the overlay planes allocated for this window. This makes the resource available for other applications.

XStoreOverlayColor

Syntax

Status
XStoreOverlayColor(*dpy*, *win*, *colorcell*)
Display ***dpy**;
Window ***win**;
XColor ***colorcell**;

Description

XStoreOverlayColor sets the color to be displayed by this pixel value in the overlay colormap associated with a window. This routine will return **BadValue** if the color of the pixel cannot be changed or the pixel is not a valid pixel for this window.

Storing the color does not change the color displayed on the screen. To actually change the color, you have to install the overlay colormap.

Pixel value 0 (zero) is the transparent overlay color.

XInstallOverlayColormap

Syntax

Status
XInstallOverlayColormap(*dpy, win*)
Display ***dpy**;
Window ***win**;

Description

XInstallOverlayColormap loads the overlay colormap for a window into the hardware colormap.

XSelectLayer

Syntax

Status
XSelectLayer(*dpy, win, layer*)
Display ***dpy**;
Window ***win**;
int ***layer**;

Description

XSelectLayer selects the layer in which following graphics commands will be done. Layer 0 is overlay, and layer 1 is the normal frame buffer.

Restrictions

- ESV does not support ALU operation in the overlay planes. This means that any X operation with an ALU mode other than **GXcopy** cannot be done.
- Any X operation that needs both a foreground and a background color will not work correctly. The foreground color will be written, but the background will not.
- **XGetImage** will return the pixels in the currently selected layer. The overlay image will be returned in the low order bits of 32 bit pixels.
- **XPutImage** will work when overlay layers are selected. The image must be stored in the low order bits of 32 bit pixels.

X Multiscreen Functionality

Having multiple screens can provide a more efficient and organized working environment than having a single screen. Multiple screens allow you to effectively expand your working surface. The term "screen" here means a logical screen not a physical one. When displayed, a logical screen takes up the entire surface of a monitor. When not displayed, it is not seen. With multiple screens, you can configure the windows you would like on each screen and then not have to rearrange them (as you do now) if you need to look at a window that is buried in a stack or iconified. With the mouse you can switch screens to an entirely new screen of different windows. This general interface is for handling multiple logical screens (including normal screens, stereo screens, *etc.*).

The concept of multiple screens has always existed in the X11 X Server. It provides for having screens that have important different physical qualities, such as pixel resolution, number of available bitplanes, or video modes such as stereo, PAL, or NTSC. Each screen has an associated screen number. To place windows on a given screen call **XOpenDisplay** with the display string parameter containing the screen number. Thus screens are separate and distinct from each other. This is analogous to having several physical monitors on which to do work.

It is interesting to note that the screen handling software in the server also provides for actually having multiple physical screens on which to do graphics. In fact the software in the server at the screen control level does not know the difference between a physical screen and a logical one.

This is an application/user interface which provides a way for you to configure logical screens in a spatial order so that you can easily move from one screen to the next. It gives the application the ability to *warp* (move) between screens. It gives you the ability to define how the screens are conceptually arrayed in a bank of screens.

Defining the Number of Available Screens

To specify that more than one screen is wanted, such as for stereo, the **-nscreens *MxN*** option is used on the command line when the Server is started, where ***M*** and ***N*** are integers. These arguments should not be too large because screens do not come free. The default is `2x2`. The screens are conceptually laid out side by side in ***M*** rows and ***N*** columns. For example, if you use the option `-nscreens 3x4` then the screens will be conceptually thought of as:

Screen 0	**Screen 1**	**Screen 2**	**Screen 3**
Screen 4	**Screen 5**	**Screen 6**	**Screen 7**
Screen 8	**Screen 9**	**Screen 10**	**Screen 11**

Defining Stereo Screens

To identify a screen as a stereo screen with square pixels, the option ***-stereoscr num*** is used. Screens are numbered beginning at 0, and screen 0 is the first one displayed when the server comes up. There can be more than one stereo screen. Screen 0 can be a regular screen or a stereo screen.

To identify a screen as a stereo screen with tall, skinny pixels, the option ***-stereotallscr num*** is used.

Moving from Screen to Screen

There are different ways provided to move between screens.

1) You can use the **mwm** menu or the **csm** X client. See the man pages for these in the "X Clients" chapter of the *ESV Workstation Reference Manual.*

2) You can use the cursor, assuming **XScreenWarpByCursor** has been enabled (see below).

   Logical screens are conceptually laid out as pictured above. You can switch between logical screens by moving the cursor off the side of the physical screen in the direction of the new logical screen. For example, in the configuration above, if you want to go from Screen 0 to Screen 4, you move the cursor off the bottom of Screen 0 and Screen 4 will be displayed.

Also, the cursor will *wrap* in moving between screens so that if you are on Screen 3 you can go to Screen 0 by moving the cursor off the right edge of Screen 3.

3) To change screens through program control, call the extension function

```
XWarpToScreen(display, screen_num);
```

where ***screen_num*** is an integer that defines which screen to display.

Enabling and Disabling Moving Between Screens Via the Cursor

There may be times when you want to disable changing screens using the cursor. This can be done by the extension call

```
XScreenWarpByCursor(display, enable_flag);
```

where ***enable_flag*** can be

True - Enable screen switch by cursor.

False - Disable screen switch by cursor.

The default condition of this parameter is false, which means that the server will not switch screens via cursor movement unless the capability is turned on using this call.

Inquiring Information about Screens

X clients may want to know specific items about available screens such as the ***MxN*** configuration and screen types. This information can be obtained using the following extension

```
XGetScreensInfo(display, screen_info);
Display *display;
ScreenInfo **screen_info;
```

where the routine returns a ScreenInfo structure that looks like this.

```
typedef struct _ScreenInfo_ {
int numofscreens; /* num of screens in the server */
int config_M; /* num of rows of screens */
int config_N; /* num of columns of screens */
struct { /* An array of structures of screen info */
int screennum; /* num of screen */
int screentype; /* screen type */
int rootvisual; /* visual type of root window */
int screen_subtype1; /* reserved, not used */
int monitor; /* which physical monitor screen is on */
} screens[numofscreens];
} ScreenInfo;
```

Once the application is finished looking at this information, it should deallocate the structure by calling

```
XFreeScreenInfo(screen_info);
```

XFreeScreensInfo

Name

XFreeScreensInfo – X extension function to deallocate the **xScreensInfo** structure

Syntax

#include <XMultiScreen.h>

int XFreeScreensInfo(*screen_info*)

xScreensInfo **screen_info*;

where the **xScreensInfo** structure has been created by and returned from a call to **XGetScreensInfo**.

Description

This X extension routine is part of the ESV Multiscreen extension for handling multiple screens in the X Server. **XFreeScreensInfo** is a function designed to free the information buffer that is returned from the function **XGetScreensInfo**. The client is responsible to call this routine to deallocate the memory used.

Related Files

XWarpToScreen

XScreenWarpByCursor

XGetScreensInfo

Xesv (**-nscreens** option)

csm (a screen manager client for warping between screens)

Copyright

XGetScreensInfo

Name

XGetScreensInfo – X extension reply to return information to the client about the number and type of available screens.

Syntax

```
#include <XMultiScreen.h>

int XGetScreensInfo(dpy, screen_info)
 Display          *dpy;
 xScreensInfo     **screen_info;
```

where the **ScreensInfo** structure is defined by the following types:

```
typedef struct _xOneScreen_ {
 int screennum;          /* Screen number */
 int screentype;         /* Screen Type: xDefaultScreen, etc. */
 int rootvisual;         /* visual type of root window */
 int screen_subtype1;    /* reserved, not used */
 int monitor;            /* which monitor screen is assigned to */
} xOneScreen;

typedef struct _xScreensInfo_ {
 int numofscreens;       /* Number of available screens */
 int rowsofscreens;      /* Number of "Rows" of screens */
 int colsofscreens;      /* Number of "Columns" of screens */
 xOneScreen screens[1];  /* Variable length array of screens. */
} xScreensInfo;
```

Description

This X extension routine is part of the ESV Multiscreen extension for handling multiple screens in the X Server. **XGetScreensInfo** is a function designed to give clients information about the number and type of screens available in a given server.

Screen Types

The screen type is returned in the ***screentype*** field of the return argument. The possible screen types are defined by the following constants:

- **xDefaultScreen**

  The standard screen, 1280 x 1024 pixels.

- **xStereoScreen Stereo**

  Video format, 640 x 512 pixels.

- **xStereoTallScreen Stereo**

  Video format, 1280 x 512 pixels.

Visual Types

The visual type of the screen's root window is returned in the ***rootvisual*** field of the return argument. The possible visual types of root windows of screens are defined by standard X constants:

- **TrueColor**

  The root window is a 24 bit True Color window (the default).

- **DirectColor**

  The root window is a 24 bit Direct Color window.

- **PseudoColor**

  The root window is an 8 bit Pseudo Color window.

Note: The default Direct Color and Pseudo Color types are not gamma corrected. This causes their colors to be slightly different from the True Color type.

Screen Rows and Columns

In the ESV X Server, the available screens are conceptually arranged in a rectangular grid. This routine also returns the number of rows and columns in the grid in the ***rowsofscreens*** and the ***colsofscreens*** fields of the return argument. When moving between screens by moving the cursor off-screen, the screen moved to is the next one in the row or column that adjoins the side of the screen that the cursor moved off of.

The Screen Number

Each screen is identified by a non-negative integer. The screen's number is returned in the ***screennum*** field of the return argument. Screens are numbered consecutively from left to right beginning with the first row. The first screen in the first row is Screen 0.

Free the Return Buffer

The returned information is stored in a buffer pointed to by the ***screen_info*** argument. This buffer is allocated by the routine at runtime and should be deallocated by the application calling **XFreeScreensInfo**.

Return Value

If there is an error during the processing of this request, **XGetScreensInfo** will return a 0; if successful it will return 1.

Related Files

XWarpToScreen

XScreenWarpByCursor

XFreeScreensInfo

Xesv (**-nscreens** option)

csm (a screen manager client for warping between screens)

Copyright

XScreenWarpByCursor

Name

XScreenWarpByCursor - X extension request to enable or disable switching screens by moving the cursor off-screen. on the monitor.

Syntax

```
#include <XMultiScreen.h>

int XScreenWarpByCursor(dpy, enable)
        register Display    *dpy;
        unsigned Bool       enable;
```

Description

This X extension routine is part of the ESV Multiscreen extension for handling multiple screens in the X Server. **XScreenWarpByCursor** is a function designed to allow clients to enable or disable the ability to change screens by moving the cursor off-screen. If the argument ***enable*** is passed as true, the ability is enabled, if false, the ability is disabled. When warping between screens by the cursor has been disabled, the displayed screen can only be switched by an active client calling **XWarpToScreen**.

Return Value

If there is an error during the processing of this request, **XScreenWarpByCursor** will return a 0; if successful it will return 1.

Errors

If an invalid value for ***enable*** is passed to this function, the server will return a **BadScreenWarpByCursor** error and will ignore the request.

Related Files

XWarpToScreen

XGetScreensInfo

XFreeScreensInfo

Xesv (**-nscreens** option)

csm (a screen manager client for warping between screens)

Copyright

XWarpToScreen

Name

XWarpToScreen - X extension request to display a different "screen" on the monitor.

Syntax

#include <XMultiScreen.h>

int XWarpToScreen(*dpy, screennum, x, y*)

register Display	***dpy;**
unsigned long	***screennum;***
int	***x;***
int	***y;***

Description

This X extension routine is part of the ESV Multiscreen extension for handling multiple screens in the X Server. **XWarpToScreen** is a function designed to allow clients to switch the display from the currently displayed screen to a new screen identified by ***screennum***. In addition, the cursor position on the new screen is identified by the ***x*** and ***y*** arguments.

Return Value

If there is an error during the processing of this request, **XWarpToScreen** will return a 0; if successful it will return 1.

Errors

If an invalid screen number is passed to this function, the server will return a **BadWarpToScreen** error and will ignore the request.

Related Files

XScreenWarpByCursor

XGetScreensInfo

XFreeScreensInfo

Xesv (**-nscreens** option)

csm (a screen manager client for warping between screens)

Copyright

X Video Timing Formats Functionality

XVideoMode

Name

XVideoMode - X extension to change the ESV video format of the monitor and video hardware.

Syntax

#include <XVideoMode.h>

int XVideoMode(*display*, *videoMode*)
register Display **display*;
unsigned long *videoMode*;

Arguments

display The connection to the server.

videoMode One of the following constants:

Monoscopic (default video mode, pixels 1280x1024)

RS_343_A_1280x1024

RS_343_A_1260x946

PAL_SECAM_768x574

RS_170_A_640x480

Stereo60KHzIntStor1280x1024

Stereo60KHzSplitStor1280x1024

Stereo60KHzIntStor640x1024

Stereo60KHzSplitStor640x1024

Description

This X extension routine is designed to allow clients to switch the ESV workstation to various possible video formats. Many of the formats have pixel resolutions that are different from the default monoscopic resolution of 1280 x 1024. As a result, they do not work well with the ESV X Server. Not all of the formats make sense in terms of using the ESV monitor for display. The stereo formats are generally not accessed via this interface but can be better used by setting up stereo screens when the X Server is started (see the man page on **Xesv**).

Return Value

If there is an error during the processing of this request, **XVideoFormat** will return a 0; if successful it will return 1.

Errors

If an invalid video format is requested, the server will return a **BadVideoMode** error and ignore the request.

Related Files

Xesv

Copyright

X Miscellaneous Traversal Functionality

There are two different types of X extensions that are in the miscellaneous traversal category. One extension is used to inquire traversal data from the PEX server. The other redraws a 3D PHIGS image and keeps it displayed for a specified amount of time.

XGetTraversalInfo allows you to retrieve information about user-defined IDs in GSE Information nodes and the current state of the transformation matrices at points in the PHIGS structure. This information is valuable for molecular modeling applications.

XFreeTraversalInfo frees the memory allocated by **XGetTraversalInfo** and should be called by the application after each **XGetTraversalInfo** call.

XRedrawDelay3D gives you increased control over the timing of the PHIGS/PEX 3D updates. This is used primarily in animation applications.

XFreeTraversalInfo

Name

XFreeTraversalInfo – Free the memory allocated by **XGetTraversalInfo** for its return buffers.

Syntax

```
#include <XTrav3D.h>

XfreeTraversalInfo(trav_data)
        PEXTravData         *trav_data;
```

where the **PEXTravData** structure is defined as:

```
typedef struct _PEXTravData {
        int         num_entries; /* Number of data records in the return. */
        char        *entries;    /* Pointer to the return data    */
} PEXTravData;
```

Description

This routine frees the memory that is allocated by the X extension **XGetTraversalInfo**. See the manual page for **XGetTraversalInfo** for complete information on the extension and its purpose.

Copyright

XGetTraversalInfo

Name

XGetTraversalInfo - X extension to inquire traversal data from the PEX server.

Syntax

```
#include <XTrav3D.h>

int XGetTraversalInfo(dpy, win, trav_info)
        register Display        *dpy;
        Window                  win;
        PEXTravInfo             **trav_info;
```

Arguments

display	The connection to the server.
window	The window ID associated with the 3D PHIGS workstation.
trav_info	The returned traversal data of type **PEXTravData**, where the **PEXTravData** structure is defined as:

```
typedef struct _PEXTravData {
        int     num_entries; /* Number of data records in the return */
        char    *entries;    /* Pointer to the return data */
} PEXTravData;
```

Description

This X extension reply routine is part of a special functionality that allows applications to retrieve certain types of traversal state information from the PEX server's structure walker. Essentially, the returned information consists of (1) user-defined IDs placed in **GSE** Information nodes in the application's PHIGS structure, and (2) the current state of the composite local and global PHIGS transformation matrices at points in the PHIGS structure designated, again, by an Information **GSE**. The IDs allow the application to develop a relationship between a given matrix and primitives of interest. This provides a way for molecular modeling applications to do distance monitoring and energy calculations by supplying them access to transformation matrices which can then be used to calculate point positions. Also, this functionality could conceivably be used to do collision detection.

New PHIGS GSE Node: Information Node

To retrieve traversal-time state information from the Server, the application must add Information **GSE** nodes at points of interest within the PHIGS struc-

ture. Upon traversal of an Information GSE node during a special information traversal (invoked by **XGetTraversalInfo**), the information requested will be buffered and returned to the application. These special nodes are ignored during regular traversals.

The Information **GSE** has a node type of **PES_GSE_INFORMATION**. (The value of this constant depends on the current implementation and may change. The PHIGS application developer should only use the above name and not its value when writing source code.) The data structures supporting this **GSE** are defined by:

```
typedef enum {
        PES_INFORMATION_MATRIX,
        PES_INFORMATION_ID
} Pes_information_type;

typedef struct {
        Pes_information_type    type;
        union {
        int                     unused;
        int                     id;
        } rec;
} Pinformation_data;
```

The **Pinformation_data** structure is pointed to by the ***data*** field of a **Pdata** structure and the ***size*** field is set to the **sizeof(Pinformation_data)**. These definitions are found in **esgdp.h**.

The definition of the **Pdata** structure is in **phigs.h**:

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

An example of coding a PHIGS application to include the Information **GSE** would be:

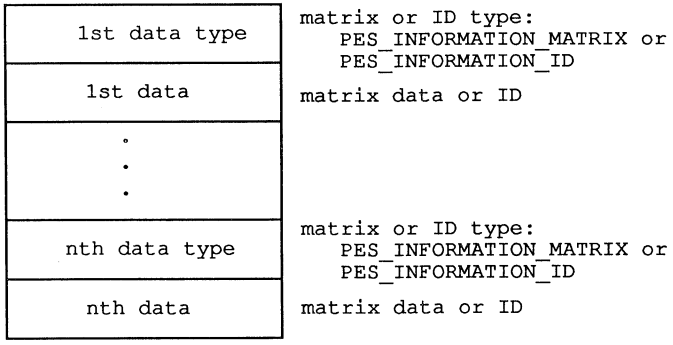
```
#include <phigs.h>
#include <esgdp.h>
#include <XTrav3D.h>

Pdata                gse_data;
Pinformation_data    *info;
   •
   •
   •
gse_data.size = (size_t)sizeof(Pinformation_data);
gse_data.data = malloc((int)gse_data.size);
```

```
info = (Pinformation_data *)gse_data.data;
info->type = PES_INFORMATION_MATRIX;
pgse(PES_GSE_INFORMATION, &gse_data);
   .
   .
   .
```

Requesting the Special Informational Traversal

After the PHIGS structure has been built, the application can request a special information traversal by calling **XGetTraversalInfo**. (See the syntax above for its arguments.) The special information traversal invoked by **XGetTraversalInfo** does not produce any graphical output. The Server will traverse the structure, maintain the matrix stack and generate the information return buffers as dictated by the special GSE nodes in the structure.

The data being returned is pointed to by the **entries** field in the **PEXTravData** structure that is returned by **XGetTraversalInfo**. The data is of the following format:



For convenience we will define the following structure types to aid in processing individual records within the data:

```
typedef struct _PEXTraverseMatrix {
          unsigned          long type;
          Pmatrix3          matrix;
} PEXTraverseMatrix;

typedef struct _PEXTraverseId {
          unsigned          long type;
          unsigned          long id;
} PEXTraverseId;
```

Return Value

If there is an error during the processing of this request, **XGetTraversalInfo** will return a 0; if successful it will return 1.

Freeing the Return Buffer

Once the return data has been processed, the client is responsible to deallocate the return buffer memory by calling:

```
XfreeTraversalInfo(trav_data)
        PEXTravData *trav_data;
```

Related Files

XFreeTraversalInfo(3X)

Copyright

XRedrawDelay3D

Name

XRedrawDelay3D - X extension to redraw a 3D PHIGS image and keep it displayed for at least a given amount of time.

Syntax

```
#include <XTrav3D.h>

int XRedrawDelay3D(dpy, win, delay)
        register Display    *dpy;
        Window              win;
        int                 delay;
```

Arguments

display	The connection to the server.
win	The window ID associated with the 3D PHIGS workstation.
delay	The number of vertical retraces ($1/60^{th}$ of a second) to delay before allowing another buffer swap to occur in a given 3D window after the redraw caused by this request takes place. After the delay period has passed, another redraw in the window will be allowed if one has been requested. If this argument is set to **XDelayCancel**, and if there is a previously processed **RedrawDelay3D** request whose delay time has not yet expired, then the delay time is immediately set to zero, and a new redraw is requested.

Description

This X extension is part of a special functionality that supports animation applications. It gives the application an increased amount of control over the timing of PHIGS/PEX 3D redraws or updates.

This function first causes a PHIGS **UPDATE_WORKSTATION** to occur. Then, if the ***delay*** argument is greater than or equal to zero, this function causes a **REDRAW_ALL_STRUCTURES** to occur, and the newly updated buffer is displayed on the screen and will remain until at least the **delay** number of monitor refreshes (1/60 s) has occurred, after which another image update and buffer swap will be allowed.

If the ***delay*** argument is equal to **DelayCancel**, and if a redraw delay time is still in effect for this window from a previous **RedrawDelay3D** request, the delay time is reset to 0 and a **REDRAW_ALL_STRUCTURES** is requested.

The application programmer should consider the following facts in using this capability:

1) Applications doing animations that do not require precise frame rates can perform adequately by using available UNIX timing capabilities. To simultaneously handle user interactions that effect the animation image, the application can use the **XSync** function to wait until a **REDRAW_ALL_STRUCTURES** or an **UPDATE_WORKSTATION** request is complete. This allows the application to avoid queueing up large numbers of redraw requests or update workstation requests. A large queue of pending updates or redraws would prohibit immediate picture changes to reflect peripheral interaction, such as dial turns, because the previously requested redraws would have to be processed first. After calling **XSync**, the application is blocked until traversal is complete. This assures that the application will never queue more than one update request in addition to the two frames that already are in the frame buffer.

2) Applications that require precise frame rates can use the **RedrawDelay3D** request to precisely control the animation frame rate, as long as the frame complexity does not exceed the ESV hardware capabilities, that is, as long as the next frame to be displayed can be generated in less time than the redraw delay time.

3) Even when using **RedrawDelay3D** requests, the application will probably want to be able to handle user interactions that change the 3D image while the animation is running. Once again, if multiple **RedrawDelay3D** requests have been queued up, they will have to be processed first by the server and graphics hardware before affects from any user interaction can be displayed, thus destroying the feel of interactivity. To avoid queueing up multiple **RedrawDelay3D** requests the application can wait on a call to **XSync**, or it can use **XSync** combined with UNIX timers to determine the approximate traversal time of a workstation and then avoid issuing **RedrawDelay3D** requests more often than the ESV can keep up with. It is a burden on the application to ensure that the **RedrawDelay3D** requests come in at an appropriate rate.

4) To aid in handling user interactions, a single previously requested **RedrawDelay3D** request can be caused to terminate prematurely by passing the constant **XDelayCancel** in the ***delay*** argument. This sets the delay timer to zero and allows other redraw requests to be processed by the graphics hardware. However if several Redraw or RedrawDelay requests have been queued up, then the Redraw that is requested by this invocation of **RedrawRequest3D** will not be processed until the other requests have first been processed.

5) Only one PHIGS workstation may do a **RedrawDelay3D** request at a time. If multiple workstations do it, there is no guarantee as to the response of the system. Neither one will animate at the correct rate.

Return Value

If there is an error during the processing of this request, **XRedrawDelay3D** will return a 0; if successful it will return 1.

Copyright

3. X Clients

Table of Contents

3. X Clients

Introduction

Following is a list of the X Clients supported on the ESV Workstation. The documentation for most of these clients can be found in the *X Window System User's Guide*, published by O'Reilly & Associates. On-line manpage documentation is available for those not found in the O'Reilly & Associates book.

appres	**atobm**	**bdftosnf**
bitmap	**bmtoa**	**csm**
editres	**esvipc**	**ico**
listres	**maze**	**mkfontdir**
muncher	**mwm**	**oclock**
pexscope	**plaid**	**puzzle**
resize	**screen**	**sessreg**
showrgb	**showsnf**	**startesvx**
startx	**uil**	**xauth**
xbiff	**xcalc**	**xclipboard**
xclock	**xcm**	**xcutsel**
xdm	**xdmshell**	**xdpyinfo**
xedit	**xev**	**xeyes**
xfd	**xfontsel**	**xhost**
xinit	**xkill**	**xload**
xlock	**xlogo**	**xlsatoms**
xlsclients	**xlsfonts**	**xlswins**
xmag	**xman**	**xmh**
xmodmap	**xprop**	**xrdb**
xrefresh	**xset**	**xsetpointer**
xsetroot	**xstdcmap**	**xterm**
xwd	**xwdrle**	**xwininfo**
xwud		

This chapter documents the following which are specific to the ESV Workstation:

- **X** - a portable, network-transparent window system
- **Xesv** - X Version 11 server for ESV Series Workstations
- **xcm** - an X client manager provides access via menus
- **csm** - Multiscreen manager for X servers supporting multiple screens
- **mwm** - Motif window manager

X

Name

X - a portable, network-transparent window system

Syntax

The X Window System is a network-transparent window system developed at the Massachusetts Institute of Technology (MIT) which runs on the ESV Workstation. The X Consortium requests that the following names be used when referring to this software:

- X
- X Window System
- X Version 11
- X Window System, Version 11
- X11

X Window System is a trademark of MIT.

Description

X Window System servers run on computers with bitmap displays. The server distributes user input to and accepts output requests from various client programs through a variety of different interprocess communication channels. Although the most common case is for the client programs to be running on the same machine as the server, clients can be run transparently from other machines (including machines with different architectures and operating systems) as well.

X supports overlapping hierarchical subwindows and text and graphics operations, on both monochrome and color displays. For a full explanation of the functions that are available, see the *Xlib - C Language X Interface* manual, the *X Window System Protocol* specification, the *X Toolkit Intrinsics - C Language Interface* manual, and various toolkit documents.

The number of programs that use X is growing rapidly. Of particular interest are: a terminal emulator (**xterm**), a window manager (**mwm**), a display manager (**xdm**), a mail managing utility (**xbiff**), a manual page browser (**xman**), a bitmap editor (**bitmap**), access control programs (**xauth** and **xhost**), user preference setting programs (**xmodmap**, **xrdb**, **xset**, and **xsetroot**), a clock (**xclock**), a font displayer (**xfd**), utilities for listing information about fonts, windows, and displays (**xdpyinfo**, **xfontsel**, **xlsfonts**, **xlswins**, **xwininfo**, **xlsclients**, and **xprop**), and screen image manipulation utilities (**xmag**, **xpr**, **xwd**, and **xwud**).

See your site administrator for other utilities, window managers, games, toolkits, etc., available from user-contributed software.

Starting Up

There are two main ways of getting the X server and an initial set of client applications started. The particular method used depends on what operating system you are running and on whether or not you use other window systems in addition to X.

- **xdm** (the X Display Manager)

  If you want to always have X running on your display, your site administrator can set your machine up to use the X Display Manager **xdm**. This program is typically started by the system at boot time and takes care of keeping the server running and getting users logged in. If you are running **xdm**, you will see a window on the screen welcoming you to the system and asking for your username and password. Simply type them in as you would at a normal terminal, pressing the RETURN key after each. If you make a mistake, **xdm** will display an error message and ask you to try again. After you have successfully logged in, **xdm** will start up your X environment. By default, if you have an executable file named **.xsession** in your home directory, **xdm** will treat it as a program (or shell script) to run to start up your initial clients (such as terminal emulators, clocks, a window manager, user settings for things like the background, the speed of the pointer, etc.). Your site administrator can provide details.

- **xinit** (run manually from the shell)

  Sites that support more than one window system might choose to use the **xinit** program for starting X manually. If this is true for your machine, your site administrator will probably have provided a program named **x11**, **startx**, or **xstart** that will do site-specific initialization (such as loading convenient default resources, running a window manager, displaying a clock, and starting several terminal emulators) in a nice way. If not, you can build such a script using the **xinit** program. This utility simply runs one user-specified program to start the server, runs another to start up any desired clients, and then waits for either to finish. Since either or both of the user-specified programs may be a shell script, this gives substantial flexibility at the expense of a nice interface. For this reason, **xinit** is not intended for end users.

Display Names

From the user's prospective, every X server has a display name of the form:

hostname:displaynumber.screennumber

This information is used by the application to determine how it should connect to the server and which screen it should use by default (on displays with multiple monitors):

- ***hostname***

  The ***hostname*** specifies the name of the machine to which the display is physically connected. If the hostname is not given, the most efficient way of communicating to a server on the same machine will be used.

- ***displaynumber***

  The word "display" is usually used to refer to collection of monitors that share a common keyboard and pointer (mouse, tablet, etc.). Most workstations tend to only have one keyboard, and therefore, only one display. Larger, multi-user systems, however, will frequently have several displays so that more than one person can be doing graphics work at once. To avoid confusion, each display on a machine is assigned a ***displaynumber*** (beginning at 0) when the X server for that display is started. The ***displaynumber*** must always be given in a display name.

- ***screennumber***

  Some displays share a single keyboard and pointer among two or more monitors. Since each monitor has its own set of windows, each screen is assigned a ***screennumber*** (beginning at 0) when the X server for that display is started. If the ***screennumber*** is not given, then screen 0 will be used.

On POSIX systems, the default display name is stored in your ***DISPLAY*** environment variable. This variable is set automatically by the **xterm** terminal emulator. However, when you log into another machine on a network, you'll need to set ***DISPLAY*** by hand to point to your display. For example,

```
% setenv DISPLAY myws:0
$ DISPLAY=myws:0; export DISPLAY
```

Finally, most X programs accept a command line option of **-display *displayname*** to temporarily override the contents of ***DISPLAY***. This is most commonly used to pop windows on another person's screen or as part of a remote shell command to start an **xterm** pointing back to your display. For example,

```
% xeyes -display joesws:0 -geometry 1000x1000+0+0
% rsh big xterm -display myws:0 -ls </dev/null &
```

X servers listen for connections on a variety of different communications channels (network byte streams, shared memory, etc.). Since there can be more than one way of contacting a given server, the ***hostname*** part of the display name is used to determine the type of channel (also called a *transport layer*) to be used. The sample servers from MIT support the following types of connections:

- local

  The ***hostname*** part of the display name should be the empty string. For example, `:0`, `:1`, and `:0.1`. The most efficient local transport will be chosen.

- TCP/IP

  The ***hostname*** part of the display name should be the server machine's IP address name. Full Internet names, abbreviated names, and IP addresses are all allowed. For example:

  `expo.lcs.mit.edu:0`, `expo:0`, `18.30.0.212:0`, `bigmachine:1`, and

  `hydra:0.1`.

- DECnet

  The ***hostname*** part of the display name should be the server machine's nodename followed by two colons instead of one. For example,

  `myws::0`, `big::1`, and `hydra::0.1`.

Access Control

The sample server provides two types of access control: an authorization protocol which provides a list of *magic cookies* clients can send to request access, and a list of hosts from which connections are always accepted. **xdm** initializes magic cookies in the server, and also places them in a file accessible to the user. Normally, the list of hosts from which connections are always accepted should be empty, so that only clients which are explicitly authorized can connect to the display. When you add entries to the host list (with **xhost**), the server no longer performs any authorization on connections from those machines. Be careful with this.

The file for authorization used by both **xdm** and **Xlib** can be specified with the environment variable ***XAUTHORITY***, and defaults to the file **.Xauthority** in the home directory. **xdm** uses **$HOME/.Xauthority** and will create it or merge in authorization records if it already exists when a user logs in.

To manage a collection of authorization files containing a collection of authorization records use **xauth**. This program allows you to extract records and insert them into other files. Using this, you can send authorization to remote machines when you log in. As the files are machine-independent, you can also simply copy the files or use NFS to share them. If you use several machines, and share a common home directory with NFS, then you never really have to worry about authorization files, the system should work correctly by default.

Note: Magic cookies transmitted "in the clear" over NFS or using **ftp** or **rcp** can be "stolen" by a network eavesdropper, and as such may enable unauthorized access. In many environments this level of security is not a concern, but if it is, you need to know the exact semantics of the particular magic cookie to know if this is actually a problem.

One of the advantages of using window systems instead of hardwired terminals is that applications don't have to be restricted to a particular size or location on the screen. Although the layout of windows on a display is controlled by the window manager that the user is running (described below), most X programs accept a command line argument of the form

-geometry *WIDTHxHEIGHT+XOFF+YOFF*

(where ***WIDTH***, ***HEIGHT***, ***XOFF***, and ***YOFF*** are numbers) for specifying a preferred size and location for this application's main window.

The ***WIDTH*** and ***HEIGHT*** parts of the geometry specification are usually measured in either pixels or characters, depending on the application. The ***XOFF*** and ***YOFF*** parts are measured in pixels and are used to specify the distance of the window from the left or right and top and bottom edges of the screen, respectively. Both types of offsets are measured from the indicated edge of the screen to the corresponding edge of the window.

The *x*-offset may be specified in the following ways:

- ***+XOFF***

  The left edge of the window is to be placed ***XOFF*** pixels in from the left edge of the screen (*i.e.*, the *x*-coordinate of the window's origin will be ***XOFF***). ***XOFF*** may be negative, in which case the window's left edge will be off the screen.

- ***-XOFF***

  The right edge of the window is to be placed ***XOFF*** pixels in from the right edge of the screen. ***XOFF*** may be negative, in which case the window's right edge will be off the screen.

The *y*-offset has similar meanings:

- ***+YOFF***

  The top edge of the window is to be ***YOFF*** pixels below the top edge of the screen (*i.e.*, the *y*-coordinate of the window's origin will be ***YOFF***). ***YOFF*** may be negative, in which case the window's top edge will be off the screen.

- ***-YOFF***

  The bottom edge of the window is to be ***YOFF*** pixels above the bottom edge of the screen. ***YOFF*** may be negative, in which case the window's bottom edge will be off the screen.

Offsets must be given as pairs; in other words, in order to specify either ***XOFF*** or ***YOFF*** both must be present. Windows can be placed in the four corners of the screen using the following specifications:

- `+0+0` upper left hand corner.
- `-0+0` upper right hand corner.
- `-0-0` lower right hand corner.
- `+0-0` lower left hand corner.

In the following examples, a terminal emulator will be placed in roughly the center of the screen and a load average monitor, mailbox, and clock will be placed in the upper right-hand corner:

```
xterm -fn 6x10 -geometry 80x24+30+200 &
xclock -geometry 48x48-0+0 &
xload -geometry 48x48-96+0 &
xbiff -geometry 48x48-48+0 &
```

Window Managers

The layout of windows on the screen is controlled by special programs called *window managers*. Although many window managers will honor geometry specifications as given, others may choose to ignore them (requiring the user to explicitly draw the window's region on the screen with the pointer, for example).

Since window managers are regular (albeit complex) client programs, a variety of different user interfaces can be built. The core distribution comes with a window manager named **twm** which supports overlapping windows, popup menus, point-and-click or click-to-type input models, title bars, nice icons, and an icon manager for those who don't like separate icon windows.

Several other window managers are available in the user-contributed software: **gwm**, **m_swm**, **olwm**, and **tekwm**.

Collections of characters for displaying text and symbols in X are known as *fonts*. A font typically contains images that share a common appearance and look nice together (for example, a single size, boldness, slant, and character set). Similarly, collections of fonts that are based on a common type face (the variations are usually called roman, bold, italic, bold italic, oblique, and bold oblique) are called *families*.

Sets of font families of the same resolution (usually measured in dots per inch) are further grouped into *directories* (so named because they were

initially stored in file system directories). Each directory contains a database which lists the name of the font and information on how to find the font. The server uses these databases to translate *font names* (which have nothing to do with file names) into font data.

The list of font directories in which the server looks when trying to find a font is controlled by the *font path.* Although most installations will choose to have the server start up with all of the commonly used font directories, the font path can be changed at any time with the **xset** program. However, it is important to remember that the directory names are on the server's machine, not on the application's.

The default font path for the sample server contains three directories:

- **/usr/lib/X11/fonts/misc**

  This directory contains many miscellaneous fonts that are useful on all systems. It contains a small family of fixed-width fonts in pixel heights 5 through 10, a family of fixed-width fonts from Dale Schumacher in similar pixel heights, several Kana fonts from Sony Corporation, a Kanji font, the standard cursor font, two cursor fonts from Digital Equipment Corporation, and OPEN LOOK™ cursor and glyph fonts from Sun Microsystems. It also has font name aliases for the font names **fixed** and **variable**.

- **/usr/lib/X11/fonts/75dpi**

  This directory contains fonts contributed by Adobe Systems, Inc., Digital Equipment Corporation, Bitstream, Inc., Bigelow and Holmes, and Sun Microsystems, Inc. for 75 dots per inch displays. An integrated selection of sizes, styles, and weights are provided for each family.

- **/usr/lib/X11/fonts/100dpi**

  This directory contains 100 dots per inch versions of some of the fonts in the **75dpi** directory.

Font databases are created by running the **mkfontdir** program in the directory containing the source or compiled versions of the fonts (in both compressed and uncompressed formats). Whenever fonts are added to a directory, **mkfontdir** should be rerun so that the server can find the new fonts. To make the server reread the font database, reset the font path with the **xset** program. For example, to add a font to a private directory, the following commands could be used:

```
% cp newfont.snf ~/myfonts
% mkfontdir ~/myfonts
% xset fp rehash
```

The **xlsfonts** program can be used to list all of the fonts that are found in font databases in the current font path. Font names tend to be fairly long as they contain all of the information needed to uniquely identify individual fonts. However, the sample server supports wildcarding of font names, so the full specification

```
-adobe-courier-medium-r-normal--10-100-75-75-m-60-iso8859-1
```

could be abbreviated as:

```
-*-courier-medium-r-normal--*-100-*-*-*-*-*-*
```

or, more tersely (but less accurately):

```
*-courier-medium-r-normal--*-100-*
```

Because the shell also has special meanings for * and **?**, wildcarded font names should be quoted:

```
% xlsfonts -fn '*-courier-medium-r-normal--*-100-*'
```

If more than one font in a given directory in the font path matches a wildcarded font name, the choice of which particular font to return is left to the server. However, if fonts from more than one directory match a name, the returned font will always be from the first such directory in the font path. The example given above will match fonts in both the **75dpi** and **100dpi** directories; if the **75dpi** directory is ahead of the **100dpi** directory in the font path, the smaller version of the font will be used.

Color Names

Most applications provide ways of tailoring (usually through resources or command line arguments) the colors of various elements in the text and graphics they display. Although black-and-white displays don't provide much of a choice, color displays frequently allow anywhere between 16 and 16 million different colors.

Colors are usually specified by their commonly-used names (for example, red, white, or medium slate blue). The server translates these names into appropriate screen colors using a color database that can usually be found in **/usr/lib/X11/rgb.txt**. Color names are case-insensitive, meaning that red, Red, and RED all refer to the same color.

Many applications also accept color specifications of the following form:

```
#rgb
#rrggbb
#rrrgggbbb
#rrrrggggbbbb
```

where ***r***, ***g***, and ***b*** are hexadecimal numbers indicating how much red, green, and blue should be displayed (zero being none and `ffff` being on full). Each field in the specification must have the same number of digits (*e.g.*, `#rrgb`

or `#gbb` are not allowed). Fields that have fewer than four digits (*e.g.*, `#rgb`) are padded out with zero's following each digit (*e.g.*, `#r000g000b000`). The eight primary colors can be represented as:

- black `#000000000000` (no color at all)
- red `#ffff00000000`
- green `#0000ffff0000`
- blue `#00000000ffff`
- yellow `#ffffffff0000` (full red and green, no blue)
- magenta `#ffff0000ffff`
- cyan `#0000ffffffff`
- white `#ffffffffffff` (full red, green, and blue)

Unfortunately, RGB color specifications are highly unportable since different monitors produce different shades when given the same inputs. Similarly, color names aren't portable because there is no standard naming scheme and because the color database needs to be tuned for each monitor. Application developers should take care to make their colors tailorable.

Keys

The X keyboard model is broken into two layers: server-specific codes (called *keycodes*) which represent the physical keys, and server-independent symbols (called *keysyms*) which represent the letters or words that appear on the keys. Two tables are kept in the server for converting keycodes to keysyms:

- modifier list

  Some keys (such as SHIFT, CONTROL, and CAPSLOCK) are known as modifiers and are used to select different symbols that are attached to a single key (such as SHIFT-a generates a capital A, and CONTROL-l generates a formfeed character ^L). The server keeps a list of keycodes corresponding to the various modifier keys. Whenever a key is pressed or released, the server generates an event that contains the keycode of the indicated key as well as a mask that specifies which of the modifier keys are currently pressed. Most servers set up this list to initially contain the various shift, control, and shift lock keys on the keyboard.

- keysym table

  Applications translate event keycodes and modifier masks into keysyms using a keysym table which contains one row for each keycode and one column for various modifier states. This table is initialized by the server to correspond to normal typewriter conventions, but is only used by client programs.

Although most programs deal with keysyms directly (such as those written with the X Toolkit Intrinsics), most programming libraries provide routines for converting keysyms into the appropriate type of string (such as ISO Latin-1).

Options

Most X programs attempt to use the same names for command line options and arguments. All applications written with the X Toolkit Intrinsics automatically accept the following options:

–display *display*	The name of the X server to use.
–geometry *geometry*	The initial size and location of the window.
–bg *color*, -background *color*	Either option specifies the color to use for the window background.
–bd *color*, -bordercolor *color*	Either option specifies the color to use for the window border.
–bw *number*, -borderwidth *number*	Either option specifies the width in pixels of the window border.
–fg *color*, -foreground *color*	Either option specifies the color to use for text or graphics.
–fn *font*, -font *font*	Either option specifies the font to use for displaying text.
–iconic	Indicates that the user would prefer the application's windows initially not be visible (as if the windows had be immediately iconified). Window managers may choose not to honor the application's request.
–name	The name under which resources for the application should be found. This option is useful in shell aliases to distinguish between invocations of an application, without resorting to creating links to alter the executable file name.
–rv, -reverse	Either option indicates that the program should simulate reverse video if possible, often by swapping the foreground and background colors. Not all programs honor this or implement it correctly. It is usually only used on monochrome displays.

+rv	Indicates that the program should not simulate reverse video. This is used to override any defaults since reverse video doesn't always work properly.
–selectionTimeout	The timeout in milliseconds within which two communicating applications must respond to one another for a selection request.
–synchronous	Indicates that requests to the X server should be sent synchronously, instead of asynchronously. Since Xlib normally buffers requests to the server, errors do not necessarily get reported immediately after they occur. This option turns off the buffering so that the application can be debugged. It should never be used with a working program.
–title *string*	The title to be used for this window. This information is sometimes used by a window manager to provide some sort of header identifying the window.
–xnllanguage *language[_territory][.codeset]*	The language, territory, and codeset for use in resolving resource and other filenames.
–xrm	A resource name and value to override any defaults. It is also very useful for setting resources that don't have explicit command line arguments.

To make the tailoring of applications to personal preferences easier, X supports several mechanisms for storing default values for program resources (*e.g.*, background color, window title, etc.). Resources are specified as strings of the form

appname*subname*subsubname...: value

that are read in from various places when an application is run. By convention, the application name is the same as the program name, but with the first letter capitalized (*e.g.*, **Bitmap** or **Emacs**) although some programs that begin with the letter "x" also capitalize the second letter for historical reasons. The precise syntax for resources is

```
ResourceLine    =  Comment | ResourceSpec
Comment         =  "!" string | <empty line>
ResourceSpec    =  WhiteSpace ResourceName WhiteSpace ":"
WhiteSpace value
ResourceName    =  [Binding] ComponentName {Binding Component-
Name}
Binding         =  "." | "*"
WhiteSpace      =  {" " | "\t"}
ComponentName   =  {"a"-"z" | "A"-"Z" | "0"-"9" | "_" | "-"}
value           =  string
string          =  {<any character not including "\n">}
```

Note that elements enclosed in curly braces ({...}) indicate zero or more occurrences of the enclosed elements

To allow values to contain arbitrary octets, the 4-character sequence `\nnn` (where n is a digit in the range of 0–7) is recognized and replaced with a single byte that contains this sequence interpreted as an octal number. For example, a value containing a NULL byte can be stored by specifying `\000`.

The Xlib routine **XGetDefault(3X)** and the resource utilities within Xlib and the X Toolkit Intrinsics obtain resources from the following sources:

RESOURCE_MANAGER root window property

Any global resources that should be available to clients on all machines should be stored in the **RESOURCE_MANAGER** property on the root window using the **xrdb** program. This is frequently taken care of when the user starts up X through the display manager or **xinit.**

application-specific files

Programs that use the X Toolkit Intrinsics will also look in the directories named by the environment variable ***XUSERFILESEARCHPATH*** or the environment variable ***XAPPLRESDIR***, plus directories in a standard place (usually under **/usr/lib/X11/**, but this can be overridden with the ***XFILESEARCHPATH*** environment variable) for application-specific resources. See the *X Toolkit Intrinsics - C Language Interface* manual for details.

XENVIRONMENT

Any user- and machine-specific resources may be indicated by setting the ***XENVIRONMENT*** environment variable to the name of a resource file to be loaded by all applications. If this variable is not defined, a file named **$HOME/.Xdefaults-*hostname*** is looked for instead, where ***hostname*** is the name of the host where the application is executing.

–xrm *resourcestring*

Applications that use the X Toolkit Intrinsics can have resources specified from the command line. The ***resourcestring*** is a single resource name and value as shown above. Note that if the string contains characters interpreted by the shell (*e.g.*, asterisk), they must be quoted. Any number of **–xrm** arguments may be given on the command line.

Program resources are organized into groups called classes so that collections of individual resources (each of which are called instances) can be set all at once. By convention, the instance name of a resource begins with a lowercase letter and class name with an upper case letter. Multiple word resources are concatenated with the first letter of the succeeding words capitalized. Applications written with the X Toolkit Intrinsics will have at least the following resources:

background (class ***Background***)

This resource specifies the color to use for the window background.

borderWidth (class ***BorderWidth***)

This resource specifies the width in pixels of the window border.

borderColor (class ***BorderColor***)

This resource specifies the color to use for the window border.

Most applications using the X Toolkit Intrinsics also have the resource ***foreground*** (class ***Foreground***), specifying the color to use for text and graphics within the window.

By combining class and instance specifications, application preferences can be set quickly and easily. Users of color displays will frequently want to set ***Background*** and ***Foreground*** classes to particular defaults. Specific color instances such as text cursors can then be overridden without having to define all of the related resources. For example,

```
bitmap*Dashed:  off
XTerm*cursorColor:  gold
XTerm*multiScroll:  on
XTerm*jumpScroll:  on
XTerm*reverseWrap:  on
XTerm*curses:  on
XTerm*Font:  6x10
XTerm*scrollBar: on
XTerm*scrollbar*thickness: 5
XTerm*multiClickTime: 500
XTerm*charClass:  33:48,37:48,45-47:48,64:48
XTerm*cutNewline: off
XTerm*cutToBeginningOfLine: off
XTerm*titeInhibit:  on
```

```
XTerm*ttyModes:  intr ^c erase ^? kill ^u
XLoad*Background: gold
XLoad*Foreground: re
XLoad*highlight: black
XLoad*borderWidth: 0
emacs*Geometry:  80x65-0-0
emacs*Background:  #5b7686
emacs*Foreground:  white
emacs*Cursor:  white
emacs*BorderColor:  white
emacs*Font:  6x10
xmag*geometry: -0-0
xmag*borderColor:  white
```

If these resources were stored in a file called **.Xresources** in your home directory, they could be added to any existing resources in the server with the following command:

```
%  xrdb -merge $HOME/.Xresources
```

This is frequently how user-friendly startup scripts merge user-specific defaults into any site-wide defaults. All sites are encouraged to set up convenient ways of automatically loading resources. See the *Xlib Manual*, section "Using the Resource Manager," for more information.

Examples

The following is a collection of sample command lines for some of the more frequently used commands. For more information on a particular command, please refer to that command's manual page.

```
%  xrdb -load $HOME/.Xresources
%  xmodmap -e "keysym BackSpace = Delete"
%  mkfontdir /usr/local/lib/X11/otherfonts
%  xset fp+ /usr/local/lib/X11/otherfonts
%  xmodmap $HOME/.keymap.km
%  xsetroot -solid '#888'
%  xset b 100 400 c 50 s 1800 r on
%  xset q
%  twm
%  xmag
%  xclock -geometry 48x48-0+0 -bg blue -fg white
%  xeyes -geometry 48x48-48+0
%  xbiff -update 20
%  xlsfonts '*helvetica*'
%  xlswins -l
%  xwininfo -root
%  xdpyinfo -display joesworkstation:0
%  xhost -joesworkstation
%  xrefresh
%  xwd | xwud
%  bitmap companylogo.bm 32x32
%  xcalc -bg blue -fg magenta
%  xterm -geometry 80x66-0-0 -name myxterm $*
```

Diagnostics

A wide variety of error messages are generated from various programs. Various toolkits are encouraged to provide a common mechanism for locating error text so that applications can be tailored easily. Programs written to interface directly to the Xlib C language library are expected to do their own error checking.

The default error handler in Xlib (also used by many toolkits) uses standard resources to construct diagnostic messages when errors occur. The defaults for these messages are usually stored in **/usr/lib/X11/XErrorDB**. If this file is not present, error messages will be rather terse and cryptic.

When the X Toolkit Intrinsics encounter errors converting resource strings to the appropriate internal format, no error messages are usually printed. This is convenient when it is desirable to have one set of resources across a variety of displays (*e.g.*, color *vs.* monochrome, many fonts *vs.* very few, etc.), although it can pose problems for trying to determine why an application might be failing. This behavior can be overridden by the setting the ***StringConversionsWarning*** resource.

To force the X Toolkit Intrinsics to always print string conversion error messages, the following resource should be placed at the top of the file that gets loaded onto the **RESOURCE_MANAGER** property using the **xrdb** program (frequently called **.Xresources** or **.Xres** in your home directory):

```
*StringConversionWarnings: on
```

To have conversion messages printed for just a particular application, the appropriate instance name can be placed before the asterisk:

```
xterm*StringConversionWarnings: on
```

Bugs

If you encounter a repeatable bug, please contact your site administrator for instructions on how to submit an X Bug Report.

See Also

Xlib – C Language X Interface, *X Toolkit Intrinsics - C Language Interface*, and *Using and Specifying X Resources*

Copyright

Authors

The X distribution is brought to you by the MIT X Consortium. The staff members at MIT responsible for this release are: Donna Converse (MIT X Consortium), Jim Fulton (MIT X Consortium), Michelle Leger (MIT X Consortium), Keith Packard (MIT X Consortium), Chris Peterson (MIT X Consortium), Bob Scheifler (MIT X Consortium), and Ralph Swick (Digital/MIT Project Athena).

Xesv

Name

Xesv – X Version 11 server for Evans and Sutherland ESV Workstation

Syntax

Xesv [-keyboard *kbd-dev*][-mouse *mouse-dev*]
[-dials *dials-dev*][-tablet *tablet-dev*]
[-mouseptr *mouse-dev*][-tabletptr *tablet-dev*]
[-nscreens *NxM*] [-stereoscr *screen-number*]
[-stereotallscr *screen-number*]
[-tc32scr *screen-number*]
[-dc32scr *screen-number*]
[-pc8scr *screen-number*]

Description

Xesv is the server for Version 11 of the X Window System on an Evans & Sutherland ESV Workstation. It is normally started using **startesvx**, **xdm(1)**, or **xinit(1)**.

In addition to the standard X Version 11 Revision 4 protocol, **Xesv** also supports the following extensions:

- PEX (PHIGS Extension to X),
- X Input Extension,
- X Picking Extension, and
- Non-Rectangular Window Shape Extension.

The ESV Workstation has a double-buffered 24-plane RGB frame buffer with z-buffered hidden line and hidden surface removal (HLHSR).

Options

-keyboard *kbd-dev*	The device to be used as the X core keyboard. Default is **/dev/kbd**.
-mouse *mouse-dev*	The input extension device to be used as a mouse.
-dials *dials-dev*	Specifies an alternative input extension device to be used as an 8-dial knob box. Default is **/dev/dials**.
-tablet *tablet-dev*	The input extension device to be used as a data tablet. Default is **/dev/tablet**.
-mouseptr *mouse-dev*	The input extension device to be used as mouse and use it as the X core pointer. Default is **/dev/mouse**.

-tabletptr *tablet-dev* The input extension device to be used as a data tablet and use it as the X core pointer.

-nscreens *MxN* Creates multiple screens conceptually arrayed in ***M*** rows and ***N*** columns.

-stereoscr *screen-number*
Makes the screen specified by ***screen-number*** a screen with stereo video format. It will be 640x512 pixels, half the resolution of the standard screen.

-stereotallscr *screen-number*
Makes the screen specified by ***screen-number*** a screen with "tall" stereo video format. It will be 1280x512 pixels, half the resolution of the standard screen in the vertical direction, and full resolution in the horizontal direction.

-tc32scr *screen-number*
Causes the root window on the screen specified by ***screen-number*** to have a visual of class TrueColor and a depth of 32. This is the default. Note that some clients which assume the root window is of class PseudoColor or DirectColor will probably not work on a TrueColor root window.

-dc32scr *screen-number*
Causes the root window on the screen specified by ***screen-number*** to have a visual class of DirectColor and depth of 32.

-pc8scr *screen-number*
Causes the root window on the screen specified by ***screen-number*** to have a visual class of PseudoColor and a depth of 8.

Environment

X_IPC If this environment variable is set, **Xesv** will use its value to specify the program which is called to create the shared memory segment for graphics structure memory and to create the interprocess communication semaphores. Defaults to **/usr/bin/X11/esvipc**.

X_UC — If this environment variable is set, **Xesv** will use its value to specify the program to load the graphics subsystem microcode. Defaults to **/usr/bin/X11/dspstart**.

PEX_CONFIG — If this environment variable is set, **Xesv** will use its value to specify the PEX configuration file. This file determines the maximum number of entries in the various workstation tables among other things. Defaults to **/usr/lib/X11/pex_config.dat**.

GM_CONFIG — If this environment variable is set, **Xesv** will use its value to specify the graphics manager configuration file. This file determines the maximum size of the shared memory segment for graphics structure memory among other things. Defaults to **/usr/lib/X11/gm_config.dat**.

GM_TMPDIR — If this environment variable is set, **Xesv** will use its value to specify the directory in which to create the graphics manager error log file in the unfortunate event that graphics manager errors occur. Defaults to **/tmp**.

ISO_PHIGS — If this environment variable is set, visual priority is guaranteed for posted structures. However, you may experience performance degradation if multiple structures are posted to a single workstation.

Files

pex_config.dat — This file is used to specify the maximum size and the number of predefined entries in PHIGS workstation tables. If the environment variable ***PEX_CONFIG*** is set, its value will be used to specify the full pathname of the file to be used for the PEX configuration file. Normally, **pex_config.dat** is found in the **/usr/lib/X11** directory.

gm_config.dat — This file is used to specify sizes of graphics manager internal objects as well as the default RGB color lookup table. If the environment variable ***GM_CONFIG*** is set, its value will be used to specify the full pathname of the file to be used for the graphics manager configuration file. Normally, **dm_config.dat** is found in the **/usr/lib/X11**directory.

X_gmerrlog	This file is generated by **Xesv** if the graphics manager encounters errors. The error description will be left in this file. If the environment variable ***GM_TMPDIR*** is set, its value will be used to specify the directory in which to place **X_gmerrlog**.
/tmpsminfo	This file is used to communicate the identification information of the shared memory segment for structure memory and the interprocess communication semaphores.

Examples

The **-tc32scr**, **-dc32scr**, and **-pc8scr** options assist with binary compatibility of existing X applications. Since many X servers to date have only supported visuals of depth 8 and class PseudoColor, numerous applications were written assuming this as a default visual type for all servers. Such applications will not work with a server whose depth is 32 and class is TrueColor. If an existing application fails, try restarting the server with the **-pc8scr** option and run the application again.

With the multiple screen option, many screens can be started each with a different depth. You could, for example, start the server with the following options

```
-nscreens 3 -dc32scr 1 -pc8scr 2
```

This results in three screens: screen 0 is TrueColor, screen1 is DirectColor, and screen 2 is PseudoColor.

See Also

Xserver(1), **X(1)**, **xdm(1)**, **xinit(1)**, **esvipc(1)**, **gm_config(5)**, **pex_config(5)**, *ESV Workstation User's Manual*

Diagnostics

Too numerous to list, but all self-explanatory.

xcm

Name

xcm - X Client Manager for easy access to X and PEX Clients on the ESV.

Syntax

xcm

Description

The **xcm** client provides users with the ability to access X clients via a menu system.

The **xcm** menu system is configurable by making changes to the **.xcmrc** file in the users home directory. If this file doesn't exist then the file **/usr/lib/X11/system.xcmrc** is accessed for menu configuration information. The format of the **.xcmrc** file is:

```
Menu
{menu number}
{client program path and name}
{client program path and name}
 .
 .
 .
```

There are 4 menu bar selections that can be specified, XGames (Menu1), XClients (Menu2), PexClients (Menu3) and ESVDemos (Menu4). Most any executable program may be entered as menu items. Command line arguments can be included in the **.xcmrc** file as they would appear if typed at the shell prompt.

These clients are run in the background and should be managed as such.

Options

xcm is written using the Motif Widget Set and it accepts the typical command line options parsed by toolkit clients.

Usage

xcm

Bugs

None known at this time.

Copyright

csm

Name

csm – Multiscreen manager for X servers supporting multiple screens

Syntax

csm

Description

The **csm** client allows users to switch between screens on an ESV workstation. **csm** displays a matrix of buttons and small text areas, each button corresponding to an available screen in the running X Server. The button that corresponds to the screen being displayed is highlighted in a different color. Clicking on a button causes the associated screen to be displayed on the monitor.

In addition, small text "note pads" are found to the right of each numbered button. The note pads can be used to type labels or reminders indicating the use of each screen.

When **csm** first starts, it displays special messages in the note pad areas to identify screens that have special characteristics. Stereo screens are marked "-Stereo." Screens whose root windows have a pseudo-color visual type are marked "-Pseudo." Screens whose root windows have a direct-color visual type are marked "-Direct." Screens whose root windows have a true-color visual type are not marked (true-color is the default).

Options

csm is written using the Motif toolkit and it accepts the typical command line options parsed by toolkit clients.

Keyboard Support

When the cursor is within the borders of **csm**, the tab key may be used to move focus from one button or text note pad to the next. If focus is on one of the numbered buttons, the ENTER key may be used to warp (switch) to the screen associated with the button.

Resources

The **csm** client pays attention to the standard resources used by Motif and the **mwm** window manager. **csm** also pays attention to the following specific resources:

fontList	The font to use in creating **csm**'s buttons and text. Applies only to non-stereo screens. Use ***stereoFontList*** for setting the font for stereo screens.

hlButtonColor	The numbered button that corresponds to the screen being displayed is highlighted by **csm**. This resource allows the user to customize the color of the button.
marginWidth	The size of the horizontal margin between **csm**'s border and **csm**'s button and text widgets. Applies to non-stereo screens. Use ***stereoMarginWidth*** for setting the margin width on stereo screens.
marginHeight	The size of the vertical margin between **csm**'s border and **csm**'s button and text widgets. Applies to non-stereo screens. Use ***stereoMarginHeight*** for setting the margin height on stereo screens.
notePadWidth	Next to each numbered button there is a small text widget that allows users to type anything that they wish as a note to themselves about what is on each screen. This resource sets the size of the text widget in the number of characters. It only applies to non-stereo screens. Use ***stereoNotePadWidth*** for setting the note pad size on stereo screens.
spacing	The horizontal and vertical spacing between buttons and text used by **csm** in pixels. Applies only to non-stereo screens. Use ***stereoSpacing*** for setting **csm** button spacing on stereo screens.
stereoFontList	The same as the ***fontList*** resource except it applies to stereo and tall stereo screens only.
stereoNotePadWidth	The same as the ***notePadWidth*** resource except it applies to stereo and tall stereo screens only.
stereoMarginWidth	The same as the ***marginWidth*** resource except it applies to stereo and tall stereo screens only.
stereoMarginHeight	The same as the ***marginHeight*** resource except it applies to stereo and tall stereo screens only.

stereoSpacing The same as the ***spacing*** resource except it applies to stereo and tall stereo screens only.

Example .Xdefaults

In particular, here is an example of customizing resources for **csm** in your **.Xdefaults** file:

```
csm*fontList: -adobe-new century schoolbook-*-*-*-*-14-*-*-*-*-*-*-*
csm*marginWidth:        3
csm*marginHeight:       3
csm*spacing:            3
csm.notePadWidth:       15
csm*stereoFontList:     -*-*-bold-r-*-*-11-*-*-*-*-*-*-*
csm*stereoMarginWidth:  0
csm*stereoMarginHeight: 0
csm*stereoSpacing:      0
csm.stereoNotePadWidth: 8
csm.geometry:           +5+5
csm.hiButtonColor:      deeppink
```

Bugs

None known at this writing.

Copyright

mwm

Name

mwm - A Window Manager

Syntax

mwm [*options*]

Description

mwm is an X11 client that provides window management functionality and some session management functionality. It provides functions that facilitate control (by the user and the programmer) of elements of window states such as placement, size, icon/normal display, input focus ownership, etc. It also provides session management functions such as stopping a client.

Options

–display *display*	The display to use; see **X(1)**.
xrm *resourcestring*	A resource string to use.

Appearance

The following sections describe the basic default behaviors of windows, icons, the icon box, input focus, and window stacking. The appearance and behavior of the window manager can be altered by changing the configuration of specific resources. Resources are defined under the heading "X Defaults."

Windows

Default **mwm** window frames have distinct components with associated functions:

Title Area	In addition to displaying the client's title, the title area is used to move the window. To move the window, place the pointer over the title area, press button 1 and drag the window to a new location. A wire frame is moved during the drag to indicate the new location. When the button is released, the window is moved to the new location.
Title Bar	This includes the title area, the minimize button, the maximize button and the window menu button.
Minimize Button	To turn the window back into its icon, do a button 1 click on the minimize button (the frame box with a small square in it).
Maximize Button	To make the window fill the screen (or enlarge to the largest size allowed by the configuration files), do a

button 1 click on the maximize button (the frame box with a large square in it).

Window Menu Button — The window menu button is the frame box with a horizontal bar in it. To pop up the window menu, press button 1. While pressing, drag the pointer on the menu to your selection, then release the button when your selection is highlighted. Alternately, you can click button 1 to pop up the menu and keep it posted; then position the pointer and select.

Resize Border Handles — To change the size of a window, move the pointer over a resize border handle (the cursor will change), press button 1, and drag the window to a new size. When the button is released, the window is resized. While dragging is being done, a rubber-band outline is displayed to indicate the new window size.

Matte — An optional matte decoration can be added between the client area and the window frame. A matte is not actually part of the window frame. There is no functionality associated with a matte.

Icons

Icons are small graphic representations of windows. A window can be minimized (iconified) using the minimize button on the window frame. Icons provide a way to reduce clutter on the screen.

Pressing mouse button 1 when the pointer is over an icon will cause the icon's window menu to pop up. Releasing the button (press + release without moving mouse = click) will cause the menu to stay posted. The menu contains the following selections:

Icon Window Menu

Selection	Accelerator	Description
Restore	ALT+F5	Opens the associated window.
Move	ALT+F7	Allows the icon to be moved with keys.
Size	ALT+F8	Inactive (not an option for icons).
Minimize	ALT+F9	Inactive (not an option for icons).
Maximize	ALT+F10	Opens the associated window and makes it fill the screen.
Lower	ALT+F11	Moves icon to bottom of icon stack.
Close	ALT+F4	Removes client from **mwm** management.

Double-clicking button 1 on an icon normalizes the icon into its associated window. Double-clicking button 1 on the icon box's icon opens the icon box and allow access to the contained icons. (In general, double-clicking a mouse button offers a quick way to have a function performed. Another example is double-clicking button 1 with the pointer on the window menu button. This closes the window.)

Icon Box

When icons begin to clutter the screen, they can be packed into an *icon box*. (To use an icon box, **mwm** must be started with the icon box configuration already set.) The icon box is a window manager window that holds client icons. Icons in the icon box can be manipulated with the mouse. The following table summarizes the behavior of this interface. Button actions apply whenever the pointer is on any part of the icon.

Button Action	Description
Button 1 click	Selects the icon.
Button 1 double click	Normalizes (opens) the associated window.
Button 1 double click	Raises an already open window to the top of the stack.
Button 1 drag	Moves the icon.

The window menu of the icon box differs from the window menu of a client window. The "Close" selection is replaced with the "PackIcons ALT+F12" selection. When selected, PackIcons packs the icons in the box to achieve neat rows with no empty slots.

Input Focus

mwm supports (by default) a keyboard input focus policy of explicit selection. This means when a window is selected to get keyboard input, it continues to get keyboard input until the window is withdrawn from window management, another window is explicitly selected to get keyboard input, or the window is iconified. There are numerous resources that control the input focus. The client window with the keyboard input focus has the active window appearance with a visually distinctive window frame.

These tables summarize the keyboard input focus selection behavior:

Button Action	Object	Function Description
Button 1 press	Window/window frame	Keyboard focus selection
Button 1 press	Icon	Keyboard focus selection

Key Action	Function Description
[ALT][TAB]	Move input focus to next window in window stack.
[ALT][SHIFT][TAB]	Move input focus to previous window in stack.

Window Stacking

The stacking order of windows may be changed as a result of setting the keyboard input focus, iconifying a window, or by doing a window manager window stacking function. When a window is iconified, the window's icon is placed on the bottom of the stack.

The following table summarizes the default window stacking behavior of the window manager:

Key Action	Function Description
[ALT][ESC]	Put bottom window on top of stack.
[ALT][SHIFT][ESC]	Put top window on bottom of stack.

A window can also be raised to the top when it gets the keyboard input focus (*e.g.*, by doing a button 1 press on the window or by using [ALT][TAB]) if this auto-raise feature is enabled with the ***focusAutoRaise*** resource.

X Defaults

mwm is configured from its resource database. This database is built from the following sources. They are listed in order of precedence, low to high:

- **app-defaults/Mwm**
- **RESOURCE_MANAGER** root window property or **$HOME/.Xdefaults**
- ***XENVIRONMENT*** variable or **$HOME/.Xdefaults**-host
- **mwm** command line options

Entries in the resource database may refer to other resource files for specific types of resources. These include files that contain bitmaps, fonts, and **mwm** specific resources such as menus and behavior specifications (*i.e.*, button and key bindings).

Mwm is the resource class name of **mwm**, and **mwm** is the resource name used by **mwm** to look up resources. In the following discussion of resource specification **Mwm** and **mwm** can be used interchangeably.

mwm uses the following types of resources:

Component Appearance Resources: These resources specify appearance attributes of window manager user interface components. They can be applied to the appearance of window manager menus, feedback windows (*e.g.*, the window reconfiguration feedback window), client window frames, and icons.

Appearance and Behavior Resources: These resources specify **mwm** appearance and behavior (*e.g.*, window management policies). They are not set separately for different **mwm** user interface components.

Client Specific Resources: These **mwm** resources can be set for a particular client window or class of client windows. They specify client-specific icon and client window frame appearance and behavior.

Resource identifiers can be either a resource name (*e.g.*, ***foreground***) or a resource class (*e.g.*, ***Foreground***). If the value of a resource is a filename and if the filename is prefixed by ~/, then it is relative to the path contained in the ***$HOME*** environment variable (generally the user's home directory). This is the only environment variable **mwm** uses directly (***$XENVIRONMENT*** is used by the resource manager).

- **Specifying Component Appearance Resources**

The syntax for specifying component appearance resources that apply to window manager icons, menus, and client window frames is

Mwmresource_id***

For example, **Mwm**foreground*** is used to specify the foreground color for **mwm** menus, icons, and client window frames.

The syntax for specifying component appearance resources that apply to a particular **mwm** component is

Mwm[menu|Icon|client|feedback]*resource_id***

If **menu** is specified, the resource is applied only to **mwm** menus; if **Icon** is specified, the resource is applied to icons; and if **client** is specified, the resource is applied to client window frames. For example, **Mwm**Icon*foreground*** is used to specify the foreground color for **mwm** icons, **Mwm**menu*foreground*** specifies the foreground color for **mwm** menus, and **Mwm**client*foreground*** is used to specify the foreground color for **mwm** client window frames.

The appearance of the title area of a client window frame (including window management buttons) can be separately configured. The syntax for configuring the title area of a client window frame is:

Mwmclient*title*resource_id***

For example, **Mwm**client*title*foreground*** specifies the foreground color for the title area. Defaults for title area resources are based on the values of the corresponding client window frame resources.

The appearance of menus can be configured based on the name of the menu. The syntax for specifying menu appearance by name is:

Mwmmenu*menu_name*resource_id***

For example, **Mwm**menu*my_menu*foreground*** specifies the foreground color for the menu named **my_menu**.

The following component appearance resources that apply to all window manager parts can be specified:

Component Appearance Resources - All Window Manager Parts

Name	Class	Value Type	Default
background	***Background***	color	varies*
backgroundPixmap	***BackgroundPixmap***	string**	varies*
bottomShadowColor	***Foreground***	color	varies*
bottomShadowPixmap	***BottomShadowPixmap***	string**	varies*
fontList	***FontList***	string***	fixed
foreground	***Foreground***	color	varies*
saveUnder	***SaveUnder***	T/F	F
topShadowColor	***Background***	color	varies*
topShadowPixmap	***TopShadowPixmap***	string**	varies*

*The default is chosen based on the visual type of the screen.
Pixmap image name. See **XmInstallImage(3X).
***X11 R3 Font description.

background (class ***Background***)

This resource specifies the background color. Any legal X color may be specified. The default value is chosen based on the visual type of the screen.

backgroundPixmap (class ***BackgroundPixmap***)

This resource specifies the background Pixmap of the **mwm** decoration when the window is inactive (does not have the keyboard focus). The default value is chosen based on the visual type of the screen.

bottomShadowColor (class ***Foreground***)

This resource specifies the bottom shadow color. This color is used for the lower and right bevels of the window manager decoration. Any legal X color may be specified. The default value is chosen based on the visual type of the screen.

bottomShadowPixmap (class ***BottomShadowPixmap***)

This resource specifies the bottom shadow Pixmap. This Pixmap is used for the lower and right bevels of the window manager decoration. The default is chosen based on the visual type of the screen.

fontList (class ***Font***)

This resource specifies the font used in the window manager decoration. The character encoding of the font should match the character encoding of the strings that are used. The default is "fixed."

foreground (class ***Foreground***)

This resource specifies the foreground color. The default is chosen based on the visual type of the screen.

saveUnder (class ***SaveUnder***)

This is used to indicate whether "save unders" are used for **mwm** components. For this to have any effect, save unders must be implemented by the X server. If save unders are implemented, the X server will save the contents of windows obscured by windows that have the save under attribute set. If the ***saveUnder*** resource is true, **mwm** will set the save under attribute on the window manager frame of any client that has it set. If ***saveUnder*** is false, save unders will not be used on any window manager frames. The default value is false.

topShadowColor (class ***Background***)

This resource specifies the top shadow color. This color is used for the upper and left bevels of the window manager decoration. The default is chosen based on the visual type of the screen.

topShadowPixmap (class ***TopShadowPixmap***)

This resource specifies the top shadow Pixmap. This Pixmap is used for the upper and left bevels of the window manager decoration. The default is chosen based on the visual type of the screen.

The following component appearance resources that apply to frame and icons can be specified:

Frame and Icon Components

Name	Class	Value Type	Default
activeBackground	***Background***	color	varies*
activeBackgroundPixmap	***BackgroundPixmap***	string**	varies*
activeBottomShadowColor	***Foreground***	color	varies*
activeBottomShadowPixmap	***BottomShadowPixmap***	string**	varies*
activeForeground	***Foreground***	color	varies*
activeTopShadowColor	***Background***	color	varies*
activeTopShadowPixmap	***TopShadowPixmap***	string**	varies*

*The default is chosen based on the visual type of the screen.
See **XmInstallImage(3X).

activeBackground (class ***Background***)

This resource specifies the background color of the **mwm** decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

activeBackgroundPixmap (class ***ActiveBackgroundPixmap***)

This resource specifies the background Pixmap of the **mwm** decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

activeBottomShadowColor (class ***Foreground***)

This resource specifies the bottom shadow color of the **mwm** decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

activeBottomShadowPixmap (class ***BottomShadowPixmap***)

This resource specifies the bottom shadow Pixmap of the **mwm** decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

activeForeground (class ***Foreground***)

This resource specifies the foreground color of the **mwm** decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

activeTopShadowColor (class ***Background***)

This resource specifies the top shadow color of the **mwm** decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

activeTopShadowPixmap (class ***TopShadowPixmap***)

This resource specifies the top shadow Pixmap of the **mwm** decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

- **Specifying Appearance and Behavior Resources**

The syntax for specifying appearance and behavior resources is

Mwmresource_id***

For example, **Mwm**keyboardFocusPolicy*** specifies the window manager policy for setting the keyboard focus to a particular client window. The following appearance and behavior resources can be specified:

Specific Appearance and Behavior Resources

Name	Class	Value Type	Default
autoKeyFocus	***AutoKeyFocus***	T/F	T
autoRaiseDelay	***AutoRaiseDelay***	millisec	500
bitmapDirectory	***BitmapDirectory***	directory	**/usr/include/X11/bitmaps**
buttonBindings	***ButtonBindings***	string	NULL
cleanText	***CleanText***	T/F	T
clientAutoPlace	***ClientAutoPlace***	T/F	T
colormapFocusPolicy	***ColormapFocusPolicy***	string	keyboard
configFile	***ConfigFile***	file	.mwmrc
deiconifyKeyFocus	***DeiconifyKeyFocus***	T/F	T
doubleClickTime	***DoubleClickTime***	millisec.	500
enforceKeyFocus	***EnforceKeyFocus***	T/F	T
fadeNormalIcon	***FadeNormalIcon***	T/F	F
frameBorderWidth	***FrameBorderWidth***	pixels	5
iconAutoPlace	***IconAutoPlace***	T/F	T
iconBoxGeometry	***IconBoxGeometry***	string	6x1+0-0
iconBoxName	***IconBoxName***	string	iconbox
iconBoxTitle	***IconBoxTitle***	string	Icons
iconClick	***IconClick***	T/F	T
iconDecoration	***IconDecoration***	string	varies
iconImageMaximum	***IconImageMaximum***	wxh	50x50
iconImageMinimum	***IconImageMinimum***	wxh	32x32
iconPlacement	***IconPlacement***	string	left bottom
iconPlacementMargin	***IconPlacementMargin***	pixels	varies
interactivePlacement	***InteractivePlacement***	T/F	F
keyBindings	***KeyBindings***	string	system
keyboardFocusPolicy	***KeyboardFocusPolicy***	string	explicit
limitResize	***LimitResize***	T/F	T
lowerOnIconify	***LowerOnIconify***	T/F	T
maximumMaximumSize	***MaximumMaximumSize***	wxh(pixels)	2Xscreen w&h
moveThreshold	***MoveThreshold***	pixels	4
passButtons	***PassButtons***	T/F	F
passSelectButton	***PassSelectButton***	T/F	T
positionIsFrame	***PositionIsFrame***	T/F	T
positionOnScreen	***PositionOnScreen***	T/F	T
quitTimeout	***QuitTimeout***	millisec.	1000
resizeBorderWidth	***ResizeBorderWidth***	pixels	10
resizeCursors	***ResizeCursors***	T/F	T
showFeedback	***ShowFeedback***	string	all
startupKeyFocus	***StartupKeyFocus***	T/F	T
transientDecoration	***TransientDecoration***	string	system title
transientFunctions	***TransientFunctions***	string	-minimize -maximize
useIconBox	***UseIconBox***	T/F	F
wMenuButtonClick	***WMenuButtonClick***	T/F	T
wMenuButtonClick2	***WMenuButtonClick2***	T/F	T

autoKeyFocus (class ***AutoKeyFocus***)

This resource is only available when the keyboard input focus policy is **explicit**. If ***autoKeyFocus*** is given a value of true, then when a window with the keyboard input focus is withdrawn from window management or is iconified, the focus is set to the previous window that had the focus. If the value given is false, there is no automatic setting of the keyboard input focus. The default value is true.

autoRaiseDelay (class ***AutoRaiseDelay***)

This resource is only available when the ***focusAutoRaise*** resource is true and the keyboard focus policy is **pointer**. The ***autoRaiseDelay*** resource specifies the amount of time (in milliseconds) that **mwm** will wait before raising a window after it gets the keyboard focus. The default value of this resource is 500 ms.

bitmapDirectory (class ***BitmapDirectory***)

This resource identifies a directory to be searched for bitmaps referenced by **mwm** resources. This directory is searched if a bitmap is specified without an absolute pathname. The default value for this resource is **/usr/include/X11/bitmaps**.

buttonBindings (class ***ButtonBindings***)

This resource identifies the set of button bindings for window management functions. The named set of button bindings is specified in the **mwm** resource description file. These button bindings are merged with the built-in default bindings. The default value for this resource is NULL (*i.e.*, no button bindings are added to the built-in button bindings).

cleanText (class ***CleanText***)

This resource controls the display of window manager text in the client title and feedback windows. If the default value of true is used, the text is drawn with a clear (no stipple) background. This makes text easier to read on monochrome systems where a background Pixmap is specified. Only the stippling in the area immediately around the text is cleared. If false, the text is drawn directly on top of the existing background.

clientAutoPlace (class ***ClientAutoPlace***)

This resource determines the position of a window when the window has not been given a user specified position. With a value of true, windows are positioned with the top left corners of the frames offset horizontally and vertically. A value of false causes the currently configured position of the window to be used. In either case, **mwm** will attempt to place the windows totally on-screen. The default value is true.

colormapFocusPolicy (class ***ColormapFocusPolicy***)

This resource indicates the colormap focus policy that is to be used. If the resource value is **explicit** then a colormap selection action is done on a client window to set the colormap focus to that window. If the value is **pointer** then the client window containing the pointer has the colormap focus. If the value is **keyboard** then the client window that has the keyboard input focus will have the colormap focus. The default value for this resource is **keyboard.**

configFile (class ***ConfigFile***)

The resource value is the pathname for an **mwm** resource description file. The default is **.mwmrc** in the user's home directory (based on the ***$HOME*** environment variable) if this file exists, otherwise it is **/usr/lib/X11/system.mwmrc.**

deiconifyKeyFocus (class ***DeiconifyKeyFocus***)

This resource only applies when the keyboard input focus policy is **explicit.** If a value of true is used, a window will receive the keyboard input focus when it is normalized (not iconified). True is the default value.

doubleClickTime (class ***DoubleClickTime***)

This resource is used to set the maximum time (in ms) between the clicks (button presses) that make up a double-click. The default value of this resource is 500 ms.

enforceKeyFocus (class ***EnforceKeyFocus***)

If this resource is given a value of true, then the keyboard input focus is always explicitly set to selected windows even if there is an indication that they are "globally active" input windows. (An example of a globally active window is a scroll bar that can be operated without setting the focus to that client.) If the resource is false, the keyboard input focus is not explicitly set to globally active windows. The default value is true.

fadeNormalIcon (class ***FadeNormalIcon***)

If this resource is given a value of true, an icon is grayed out whenever it has been normalized (its window has been opened). The default value is false.

frameBorderWidth (class ***FrameBorderWidth***)

This resource specifies the width (in pixels) of a client window frame border without resize handles. The border width includes the 3-D shadows. The default value is 5 pixels.

iconAutoPlace (class ***IconAutoPlace***)

This resource indicates whether icons are automatically placed on the screen by **mwm**, or are placed by the user. Users may specify an initial icon position and may move icons after initial placement; however, **mwm** will

adjust the user-specified position to fit into an invisible grid. When icons are automatically placed, **mwm** places them into the grid using a scheme set with the ***iconPlacement*** resource. If the ***iconAutoPlace*** resource has a value of true, then **mwm** does automatic icon placement. A value of false allows user placement. The default value of this resource is true.

iconBoxGeometry (class ***IconBoxGeometry***)

This resource indicates the initial position and size of the icon box. The value of the resource is a standard window geometry string with the following syntax:

```
[=][widthxheight][{+-}xoffset{+-}yoffset]
```

If the offsets are not provided, the ***iconPlacement*** policy is used to determine the initial placement. The units for width and height are columns and rows.

The actual screen size of the icon box window will depend on the ***iconImageMaximum*** (size) and ***iconDecoration*** resources. The default value for size is (6 * iconWidth + padding) wide by (1 * iconHeight + padding) high. The default value of the location is +0 -0.

iconBoxName (class ***IconBoxName***)

This resource specifies the name that is used to look up icon box resources. The default name is "iconbox."

iconBoxTitle (class ***IconBoxTitle***)

This resource specifies the name that is used in the title area of the icon box frame. The default value is "Icons."

iconClick (class ***IconClick***)

When this resource is given the value of true, the system menu is posted and left posted when an icon is clicked. The default value is true.

iconDecoration (class ***IconDecoration***)

This resource specifies the general icon decoration. The resource value is **label** (only the label part is displayed) or **image** (only the image part is displayed) or **label image** (both the label and image parts are displayed). A value of **activelabel** can also be specified to get a label (not truncated to the width of the icon) when the icon is selected. The default decoration for icon box icons is each one has a label part and an image part (**label image**). The default icon decoration for stand-alone icons is each one has an active label part, a label part and an image part (**activelabel label image**).

iconImageMaximum (class ***IconImageMaximum***)

This resource specifies the maximum size of the icon image. The resource value is ***widthxheight*** (*e.g.*, 64x64). The maximum supported size is 128x128. The default value of this resource is 50x50.

IconImageMinimum (class ***IconImageMinimum***)

This resource specifies the minimum size of the icon image. The resource value is ***widthxheight*** (*e.g.*, 32x50). The minimum supported size is 16x16. The default value of this resource is 32x32.

IconPlacement (class ***IconPlacement***)

This resource specifies the icon placement scheme to be used. The resource value has the following syntax:

primary_layout secondary_layout

The layout values are one of the following:

top	Lay the icons out top to bottom.
bottom	Lay the icons out bottom to top.
left	Lay the icons out left to right.
right	Lay the icons out right to left.

A horizontal (or vertical) layout value should not be used for both the ***primary_layout*** and the ***secondary_layout*** (*e.g.*, don't use **top** for the ***primary_layout*** and **bottom** for the ***secondary_layout***). The ***primary_layout*** indicates whether, when an icon placement is done, the icon is placed in a row or a column and the direction of placement. The ***secondary_layout*** indicates where to place new rows or columns. For example, **top right** indicates that icons should be placed top to bottom on the screen and that columns should be added from right to left on the screen. The default placement is **left bottom** (icons are placed left to right on the screen, with the first row on the bottom of the screen, and new rows added from the bottom of the screen to the top of the screen).

IconPlacementMargin (class ***IconPlacementMargin***)

This resource sets the distance between the edge of the screen and the icons that are placed along the edge of the screen. The value should be greater than or equal to 0. A default value (see below) is used if the value specified is invalid. The default value for this resource is equal to the space between icons as they are placed on the screen. (This space is based on maximizing the number of icons in each row and column.)

InteractivePlacement (class ***InteractivePlacement***)

This resource controls the initial placement of new windows on the screen. If the value is true, then the pointer shape changes before a new window is placed on the screen to indicate to the user that a position should be selected for the upper-left hand corner of the window. If the value is false, then windows are placed according to the initial window configuration attributes. The default value of this resource is false.

keyBindings (class ***KeyBindings***)

This resource identifies the set of key bindings for window management functions. If specified these key bindings replace the built-in default bindings. The named set of key bindings is specified in **mwm** resource description file. The default value is the set of system-compatible key bindings.

keyboardFocusPolicy (class ***KeyboardFocusPolicy***)

If set to **pointer**, the keyboard focus policy is to have the keyboard focus set to the client window that contains the pointer (the pointer could also be in the client window decoration that **mwm** adds). If set to **explicit**, the policy is to have the keyboard focus set to a client window when the user presses button 1 with the pointer on the client window or any part of the associated **mwm** decoration. The default value for this resource is **explicit**.

limitResize (class ***LimitResize***)

If this resource is true, the user is not allowed to resize a window to greater than the maximum size. The default value for this resource is true.

lowerOnIconify (class ***LowerOnIconify***)

If this resource is given the default value of true, a window's icon appears on the bottom of the window stack when the window is minimized (iconified). A value of false places the icon in the stacking order at the same place as its associated window.

maximumMaximumSize (class ***MaximumMaximumSize***)

This resource is used to limit the maximum size of a client window as set by the user or client. The resource value is ***widthxheight*** (*e.g.*, 1024x1024) where the width and height are in pixels. The default value of this resource is twice the screen width and height.

moveThreshold (class ***MoveThreshold***)

This resource is used to control the sensitivity of dragging operations that move windows and icons. The value of this resource is the number of pixels that the locator will be moved with a button down before the move operation is initiated. This is used to prevent window/icon movement when a click or double-click is done and there is unintentional pointer movement with the button down. The default value of this resource is 4 pixels.

passButtons (class ***PassButtons***)

This resource indicates whether or not button press events are passed to clients after they are used to do a window manager function in the client context. If the resource value is false, then the button press will not be passed to the client. If the value is true, the button press is passed to the client window. The window manager function is done in either case. The default value for this resource is false.

passSelectButton (class ***PassSelectButton***)

This resource indicates whether or not the keyboard input focus selection button press (if ***keyboardFocusPolicy*** is **explicit**) is passed on to the client window or used to do a window management action associated with the window decorations. If the resource value is false then the button press will not be used for any operation other than selecting the window to be the keyboard input focus; if the value is true, the button press is passed to the client window or used to do a window management operation, if appropriate. The keyboard input focus selection is done in either case. The default value for this resource is true.

positionIsFrame (class ***PositionIsFrame***)

This resource indicates how client window position information (from the **WM_NORMAL_HINTS** property and from configuration requests) is to be interpreted. If the resource value is true then the information is interpreted as the position of the **mwm** client window frame. If the value is false then it is interpreted as being the position of the client area of the window. The default value of this resource is true.

positionOnScreen (class ***PositionOnScreen***)

This resource is used to indicate that windows should initially be placed (if possible) so that they are not clipped by the edge of the screen (if the resource value is true). If a window is larger then the size of the screen then at least the upper left corner of the window will be on-screen. If the resource value is false, then windows are placed in the requested position even if totally off-screen. The default value of this resource is true.

quitTimeout (class ***QuitTimeout***)

This resource specifies the amount of time (in milliseconds) that **mwm** will wait for a client to update the **WM_COMMAND** property after **mwm** has sent the **WM_SAVE_YOURSELF** message. This protocol will only be used for those clients that have a **WM_SAVE_YOURSELF** atom and no **WM_DELETE_WINDOW** atom in the **WM_PROTOCOLS** client window property. The default value of this resource is 1000 ms. (Refer to the **f.kill** function for additional information.)

resizeBorderWidth (class ***ResizeBorderWidth***)

This resource specifies the width (in pixels) of a client window frame border with resize handles. The specified border width includes the 3-D shadows. The default is 10 pixels.

resizeCursors (class ***ResizeCursors***)

This is used to indicate whether the resize cursors are always displayed when the pointer is in the window size border. If true, the cursors are shown, otherwise the window manager cursor is shown. The default value is true.

showFeedback (class ***ShowFeedback***)

This resource controls when feedback information is displayed. It controls both window position and size feedback during move or resize operations and initial client placement. It also controls window manager message and dialog boxes. The value for this resource is a list of names of the feedback options to be enabled; the names must be separated by a space. The names of the feedback options are shown below:

Name	Description
all	Show all feedback. (Default value.)
behavior	Confirm behavior switch.
move	Show position during move.
none	Show no feedback.
placement	Show position and size during initial placement.
resize	Show size during resize.
restart	Confirm **mwm** restart.

The following command line illustrates the syntax for ***showFeedback***:

MwmshowFeedback*: placement resize behavior restart**

This resource specification provides feedback for initial client placement and resize, and enables the dialog boxes to confirm the restart and set behavior functions. It disables feedback for the **move** function.

startupKeyFocus (class ***StartupKeyFocus***)

This resource is only available when the keyboard input focus policy is **explicit**. When given the default value of true, a window gets the keyboard input focus when the window is mapped (*i.e.*, initially managed by the window manager).

transientDecoration (class ***TransientDecoration***)

This controls the amount of decoration that **Mwm** puts on transient windows. The decoration specification is exactly the same as for the ***clientDecoration*** (client specific) resource. Transient windows are identified by the **WM_TRANSIENT_FOR** property which is added by the client to indicate a relatively temporary window. The default value for this resource is **menu title** (*i.e.*, transient windows will have resize borders and a titlebar with a window menu button).

transientFunctions (class ***TransientFunctions***)

This resource is used to indicate which window management functions are applicable (or not applicable) to transient windows. The function specification is exactly the same as for the ***clientFunctions*** (client specific) resource. The default value for this resource is **-minimize -maximize**.

useIconBox (class ***UseIconBox***)

If this resource is given a value of true, icons are placed in an icon box. When an icon box is not used, the icons are placed on the root window (default value).

wMenuButtonClick (class ***WMenuButtonClick***)

This resource indicates whether a click of the mouse when the pointer is over the window menu button will post and leave posted the system menu. If the value given this resource is true, then the menu will remain posted. True is the default value for this resource.

wMenuButtonClick2 (class ***WMenuButtonClick2***)

When this resource is given the default value of true, a double-click action on the window menu button will do an **f.kill** function.

- **Specifying Client Specific Resources**

The syntax for specifying client specific resources is

Mwmclient_name_or_class*resource_id***

For example, **Mwm**mterm*windowMenu*** is used to specify the window menu to be used with **mterm** clients. The syntax for specifying client specific resources for all classes of clients is

Mwmresource_id***

Specific client specifications take precedence over the specifications for all clients. For example, **Mwm**windowMenu*** is used to specify the window menu to be used for all classes of clients that don't have a window menu specified. The syntax for specifying resource values for windows that have an unknown name and class (*i.e.* the window does not have a **WM_CLASS** property associated with it) is

Mwm*defaults*resource_id

For example, **Mwm*defaults*iconImage** is used to specify the icon image to be used for windows that have an unknown name and class. The following client specific resources can be used.

Client Specific Resources

Name	Class	Value Type	Default
clientDecoration	***ClientDecoration***	string	all
clientFunctions	***ClientFunctions***	string	all
focusAutoRaise	***FocusAutoRaise***	T/F	T
iconImage	***IconImage***	pathname(image)	
iconImageBackground	***Background***	color	icon background
iconImageBottomShadowColor	***Foreground***	color	icon bottom shadow
iconImageBottomShadowPixmap	***BottomShadowPixmap***	color	icon bottom shadow pixmap
iconImageForeground	***Foreground***	color	icon foreground
iconImageTopShadowColor	***Background***	color	icon top shadow color
iconImageTopShadowPixmap	***TopShadowPixmap***	color	icon top shadow pixmap
matteBackground	***Background***	color	background
matteBottomShadowColor	***Foreground***	color	bottom shadow color
matteBottomShadowPixmap	***BottomShadowPixmap***	color	bottom shadow pixmap
matteForeground	***Foreground***	color	foreground
matteTopShadowColor	***Background***	color	top shadow color
matteTopShadowPixmap	***TopShadowPixmap***	color	top shadow pixmap
matteWidth	***MatteWidth***	pixels	0
maximumClientSize	***MaximumClientSize***	wxh	fill the screen
useClientIcon	***UseClientIcon***	T/F	F
windowMenu	***WindowMenu***	string	string

clientDecoration (class ***ClientDecoration***)

This resource controls the amount of window frame decoration. The resource is specified as a list of decorations which can be included in the frame. If a decoration is preceded by a minus sign, then that decoration is excluded from the frame. The sign of the first item in the list determines the initial amount of decoration. If the sign of the first decoration is minus, then **mwm** assumes all decorations are present and starts subtracting from that set. If the sign of the first decoration is plus (or not specified), then **mwm** starts with no decoration and builds up a list from the resource.

Name	Description
all	Include all decorations (default value)
border	Window border
maximize	Maximize button (includes title bar)
minimize	Minimize button (includes title bar)
none	No decorations
resizeh	Border resize handles (includes border)
menu	Window menu button (includes title bar)
title	Title bar (includes border)

Examples:

Mwm*XClock.clientDecoration: -resizeh -maximize

This removes the resize handles and maximize button from **XClock** windows.

Mwm*XClock.clientDecoration: menu minimize border

This does the same thing as above. Note that either **menu** or **minimize** implies **title**.

clientFunctions (class ***ClientFunctions***)

This resource is used to indicate which **mwm** functions are applicable (or not applicable) to the client window. The value for the resource is a list of functions. If the first function in the list has a minus sign in front of it, then **mwm** starts with all functions and subtracts from that set. If the first function in the list has a plus sign in front of it, then **mwm** starts with no functions and builds up a list. Each function in the list must be preceded by the appropriate plus or minus sign and be separated from the next function by a space. The table below lists the functions available for this resource.

Name	Description
all	Include all functions (default value)
none	No functions
resize	**f.resize**
move	**f.move**
minimize	**f.minimize**
maximize	**f.maximize**
close	**f.kill**

focusAutoRaise (class ***FocusAutoRaise***)

When the value of this resource is true, clients are made completely unobscured when they get the keyboard input focus. If the value is false, the stacking of windows on the display is not changed when a window gets the keyboard input focus. The default value is true.

iconImage (class ***IconImage***)

This resource can be used to specify an icon image for a client (*e.g.*, **Mwm*myclock*iconImage**). The resource value is a pathname for a bitmap file. The value of the (client specific) **useClientIcon** resource is used to determine whether or not user supplied icon images are used instead of client supplied icon images. The default value is to display a built-in window manager icon image.

iconImageBackground (class ***Background***)

This resource specifies the background color of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon background color (*i.e.*, specified by **Mwm*background** or **Mwm*icon*background**).

iconImageBottomShadowColor (class ***Foreground***)

This resource specifies the bottom shadow color of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon bottom shadow color (*i.e.*, specified by **Mwm*icon*bottomShadowColor**).

iconImageBottomShadowPixmap (class ***BottomShadowPixmap***)

This resource specifies the bottom shadow Pixmap of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon bottom shadow Pixmap.

iconImageForeground (class ***Foreground***)

This resource specifies the foreground color of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon foreground color (*i.e.*, specified by **Mwm*foreground** or **Mwm*icon*foreground**).

iconImageTopShadowColor (class ***Background***)

This resource specifies the top shadow color of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon top shadow color.

iconImageTopShadowPixmap (class ***TopShadowPixmap***)

This resource specifies the top shadow Pixmap of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon top shadow Pixmap (*i.e.*, specified by **Mwm*icon*topShadowPixmap**).

matteBackground (class ***Background***)

This resource specifies the background color of the matte, when ***matteWidth*** is positive. The default value of this resource is the client background color (*i.e.*, specified by **Mwm*background** or **Mwm*client*background**).

matteBottomShadowColor (class ***Foreground***)

This resource specifies the bottom shadow color of the matte, when ***matteWidth*** is positive. The default value of this resource is the client bottom shadow color (*i.e.*, specified by **Mwm*bottomShadowColor** or **Mwm*client*bottomShadowColor**).

matteBottomShadowPixmap (class ***BottomShadowPixmap***)

This resource specifies the bottom shadow Pixmap of the matte, when ***matteWidth*** is positive. The default value of this resource is the client bottom shadow Pixmap.

matteForeground (class ***Foreground***)

This resource specifies the foreground color of the matte, when ***matteWidth*** is positive. The default value of this resource is the client foreground color (*i.e.*, specified by **Mwm**foreground*** or **Mwm*client**foreground**).

matteTopShadowColor (class ***Background***)

This resource specifies the top shadow color of the matte, when ***matteWidth*** is positive. The default value of this resource is the client top shadow color (*i.e.*, specified by **Mwm**topShadowColor*** or **Mwm*client**topShadowColor**).

matteTopShadowPixmap (class ***TopShadowPixmap***)

This resource specifies the top shadow Pixmap of the matte, when ***matteWidth*** is positive. The default value of this resource is the client top shadow Pixmap.

matteWidth (class ***MatteWidth***)

This resource specifies the width of the optional matte. The default value is 0, which effectively disables the matte.

maximumClientSize (class ***MaximumClientSize***)

This is a size specification indicating the client size to be used when an application is maximized. The resource value is specified as ***widthxheight***. The width and height are interpreted in the units that the client uses (*e.g.*, for terminal emulators this is generally characters). If this resource is not specified then the maximum size from the **WM_NORMAL_HINTS** property is used, if it has been set. Otherwise the default value is the size where the client window with window management borders fills the screen. When the maximum client size is not determined by the ***maximumClientSize*** resource, the ***MaximumSize*** resource value is used as a constraint on the maximum size.

useClientIcon (class ***UseClientIcon***)

If the value given for this resource is true, a client supplied icon image will take precedence over a user supplied icon image. The default value is false, making the user supplied icon image have higher precedence than the client supplied icon image.

windowMenu (class ***WindowMenu***)

This resource indicates the name of the menu pane that is posted when the window menu is popped up (usually by pressing button 1 on the window menu button on the client window frame). Menu panes are specified in the **mwm** resource description file. Window menus can be customized on a client

class basis by specifying resources of the form **Mwm*client_name_or_class*windowMenu** (see "**mwm** Resource Description File Syntax" below). The default value of this resource is the name of the built-in window menu specification.

Resource Description File

The **mwm** resource description file is a supplementary resource file that contains resource descriptions referred to by entries in the defaults files (**.Xdefaults, app-defaults/Mwm**). It contains descriptions of resources that are to be used by **mwm**, and that cannot be easily encoded in the defaults files (a bitmap file is an analogous type of resource description file). A particular **mwm** resource description file can be selected using the **configFile** resource. The following types of resources can be described in the **mwm** resource description file:

- **Buttons**
  Window manager functions can be bound (associated) with button events.
- **Keys**
  Window manager functions can be bound (associated) with key press events.
- **Menus**
  Menu panes can be used for the window menu and other menus posted with key bindings and button bindings.

mwm Resource Description File Syntax

The **mwm** resource description file is a standard text file that contains items of information separated by blanks, tabs, and new lines characters. Blank lines are ignored. Items or characters can be quoted to avoid special interpretation (*e.g.*, the comment character can be quoted to prevent it from being interpreted as the comment character). A quoted item can be contained in double quotes ("). Single characters can be quoted by preceding them by the back-slash character (\). All text from an unquoted pound sign (#) to the end of the line is regarded as a comment and is not interpreted as part of a resource description. If an exclamation point (!) is the first character in a line, the line is regarded as a comment. Window manager functions can be accessed with button and key bindings, and with window manager menus. Functions are indicated as part of the specifications for button and key binding sets, and menu panes. The function specification has the following syntax:

```
function = function_name [function_args]
function_name = window manager function
function_args ={quoted_item | unquoted_item}
```

The following functions are supported. If a function is specified that isn't one of the supported functions then it is interpreted by **mwm** as **f.nop**.

f.beep

This function causes a beep.

f.circle_down ***[icon | window]***

This function causes the window or icon that is on the top of the window stack to be put on the bottom of the window stack (so that it is no longer obscuring any other window or icon). This function affects only those windows and icons that are obscuring other windows and icons, or that are obscured by other windows and icons. Secondary windows (*i.e.* transient windows) are restacked with their associated primary window. Secondary windows always stay on top of the associated primary window and there can be no other primary windows between the secondary windows and their primary window. If an ***icon*** function argument is specified, then the function applies only to icons. If a ***window*** function argument is specified then the function applies only to windows.

f.circle_up ***[icon | window]***

This function raises the window or icon on the bottom of the window stack (so that it is not obscured by any other windows). This function affects only those windows and icons that are obscuring other windows and icons, or that are obscured by other windows and icons. Secondary windows (*i.e.* transient windows) are restacked with their associated primary window. If an ***icon*** function argument is specified then the function applies only to icons. If a ***window*** function argument is specified then the function applies only to windows.

f.exec or **!**

This function causes ***command*** to be executed (using the value of the ***$SHELL*** environment variable if it is set, otherwise **/bin/sh**). The **!** notation can be used in place of the **f.exec** function name.

f.focus_color

This function sets the colormap focus to a client window. If this function is done in a **root** context, then the default colormap (setup by the X Window System for the screen where **mwm** is running) is installed and there is no specific client window colormap focus. This function is treated as **f.nop** if ***colormapFocusPolicy*** is not **explicit**.

f.focus_key

This function sets the keyboard input focus to a client window or icon. This function is treated as **f.nop** if ***keyboardFocusPolicy*** is not **explicit** or the function is executed in a **root** context.

f.kill

If the **WM_DELETE_WINDOW** protocol is set up, the client is sent a client message event indicating that the client window should be deleted. If the **WM_SAVE_YOURSELF** protocol is set up and the **WM_DELETE_WINDOW** protocol is not set up, the client is sent a client message event indicating that the client needs to prepare to be terminated. If the client does not have the **WM_DELETE_WINDOW** or **WM_SAVE_YOURSELF** protocol set up, this function causes a client's X connection to be terminated (usually resulting in termination of the client). Refer to the description of the ***quitTimeout*** resource and the **WM_PROTOCOLS** property.

f.lower *[-client]*

This function lowers a client window to the bottom of the window stack (where it obscures no other window). Secondary windows (*i.e.* transient windows) are restacked with their associated primary window. The ***client*** argument indicates the name or class of a client to lower. If the ***client*** argument is not specified then the context in which the function was invoked indicates the window or icon to lower.

f.maximize

This function causes a client window to be displayed at its maximum size.

f.menu

This function associates a cascading (pull-right) menu with a menu pane entry or a menu with a button or key binding. The ***menu_name*** function argument identifies the menu to be used.

f.minimize

This function causes a client window to be minimized (iconified). When a window is minimized and no icon box is used, its icon is placed on the bottom of the window stack (such that it obscures no other window). If an icon box is used, then the client's icon changes to its iconified form inside the icon box. Secondary windows (*i.e.* transient windows) are minimized with their associated primary window. There is only one icon for a primary window and all its secondary windows.

f.move

This function allows a client window to be interactively moved.

f.next_cmap

This function installs the next colormap in the list of colormaps for the window with the colormap focus.

f.next_key *[icon | window | transient]*

This function sets the keyboard input focus to the next window/icon in the set of windows/icons managed by the window manager (the ordering of this set is based on the stacking of windows on the screen). This function is treated as **f.nop** if ***keyboardFocusPolicy*** is not **explicit**. The keyboard input focus is only moved to windows which do not have an associated secondary window that is application modal. If the ***transient*** argument is specified, then transient (secondary) windows are traversed (otherwise, if only ***window*** is specified, traversal is done only to the last focused window in a transient group). If an ***icon*** function argument is specified, then the function applies only to icons. If a ***window*** function argument is specified, then the function applies only to windows.

f.nop

This function does nothing.

f.normalize

This function causes a client window to be displayed with its normal size. Secondary windows (*i.e.* transient windows) are placed in their normal state along with their associated primary window.

f.pack_icons

This function is used to adjust the icon layout (based on the layout policy being used) on the root window or in the icon box. In general, this causes icons to be "packed" into the icon grid.

f.pass_keys

This function is used to enable/disable (toggle) processing of key bindings for window manager functions. When it disables key binding processing all keys are passed on to the window with the keyboard input focus and no window manager functions are invoked. If the **f.pass_keys** function is invoked with a key binding to disable key binding processing, the same key binding can be used to enable key binding processing.

f.post_wmenu

This function is used to post the window menu. If a key is used to post the window menu and a window menu button is present, the window menu is automatically placed with its top-left corner at the bottom-left corner of the window menu button for the client window. If no window menu button is present, the window menu is placed at the top-left corner of the client window.

f.prev_cmap

This function installs the previous colormap in the list of colormaps for the window with the colormap focus.

f.prev_key ***[icon | window | transient]***

This function sets the keyboard input focus to the previous window/icon in the set of windows/icons managed by the window manager (the ordering of this set is based on the stacking of windows on the screen). This function is treated as **f.nop** if ***keyboardFocusPolicy*** is not **explicit**. The keyboard input focus is only moved to windows which do not have an associated secondary window that is application modal. If the ***transient*** argument is specified, then transient (secondary) windows are traversed (otherwise, if only ***window*** is specified, only the last focused window in a transient group is traversed). If an ***icon*** function argument is specified, the function applies only to icons. If a ***window*** function argument is specified, the function applies only to windows.

f.quit_mwm

This function terminates **mwm** (but not the X window system).

f.raise ***[-client]***

This function raises a client window to the top of the window stack (where it is obscured by no other window). Secondary windows (*i.e.* transient windows) are restacked with their associated primary window. The ***client*** argument indicates the name or class of a client to raise. If the ***client*** argument is not specified, the context in which the function was invoked indicates the window or icon to raise.

f.raise_lower

This function raises a client window to the top of the window stack if it is partially obscured by another window, otherwise it lowers the window to the bottom of the window stack. Secondary windows (*i.e.* transient windows) are restacked with their associated primary window.

f.refresh

This function causes all windows to be redrawn.

f.refresh_win

This function causes a client window to be redrawn.

f.resize

This function allows a client window to be interactively resized.

f.restart

This function causes **mwm** to be restarted (effectively terminated and re-executed).

f.send_msg *message_number*

This function sends a client message of the type **_MOTIF_WM_MESSAGES** with the ***message_type*** indicated by the ***message_number*** function argument. The client message will only be sent if ***message_number*** is included in the client's **_MOTIF_WM_MESSAGES** property. A menu item label is grayed out if it is used to do an **f.send_msg** of a message that is not included in the client's **_MOTIF_WM_MESSAGES** property.

f.separator

This function causes a menu separator to be put in the menu pane at the specified location (the label is ignored).

f.set_behavior

This function causes the window manager to restart with the default OSF behavior (if a custom behavior is configured) or a custom behavior (if an OSF default behavior is configured).

f.title

This function inserts a title in the menu pane at the specified location. Each function may be constrained as to which resource types can specify the function (*e.g.*, menu pane) and also the context in which the function can be used (*e.g.*, the function is done to the selected client window). Function contexts are

- **root**
  No client window or icon has been selected as an object for the function.
- **window**
  A client window has been selected as an object for the function. This includes the window's title bar and frame. Some functions are applied only when the window is in its normalized state (*e.g.*, **f.maximize**) or its maximized state (*e.g.*, **f.normalize**).
- **icon**
  An icon has been selected as an object for the function.

If a function is specified in a type of resource where it is not supported or is invoked in a context that does not apply then the function is treated as **f.nop**. The following table indicates the resource types and function contexts in which window manager functions apply.

Function	Contexts	Resources
f.beep	**root,icon,window**	button,key,menu
f.circle_down	**root,icon,window**	button,key,menu
f.circle_up	**root,icon,window**	button,key,menu
f.exec	**root,icon,window**	button,key,menu
f.focus_color	**root,icon,window**	button,key,menu
f.focus_key	**root,icon,window**	button,key,menu
f.kill	**icon,window**	button,key,menu
f.lower	**root,icon,window**	button,key,menu
f.maximize	**icon,window**(normal)	button,key,menu
f.menu	**root,icon,window**	button,key,menu
f.minimize	**window**	button,key,menu
f.move	**icon,window**	button,key,menu
f.next_cmap	**root,icon,window**	button,key,menu
f.next_key	**root,icon,window**	button,key,menu
f.nop	**root,icon,window**	button,key,menu
f.normalize	**icon,window(max)**	button,key,menu
f.pack_icons	**root,icon,window**	button,key,menu
f.pass_keys	**root,icon,window**	button,key,menu
f.post_wmenu	**root,icon,window**	button,key
f.prev_cmap	**root,icon,window**	button,key,menu
f.prev_key	**root,icon,window**	button,key,menu
f.quit_mwm	**root**	button,key,menu
f.raise	**root,icon,window**	button,key,menu
f.raise_lower	**icon,window**	button,key,menu
f.refresh	**root,icon,window**	button,key,menu
f.refresh_win	**window**	button,key,menu
f.resize	**window**	button,key,menu
f.restart	**root**	button,key,menu
f.send_msg	**icon,window**	button,key,menu
f.separator	**root,icon,window**	menu
f.set_behavior	**root,icon,window**	button,key,menu
f.title	**root,icon,window**	menu

Window Manager Event Specification

Events are indicated as part of the specifications for button and key binding sets, and menu panes.

Button events have the following syntax:

```
button =[modifier_list]<button_event_name>
modifier_list = modifier_name {modifier_name}
```

All modifiers specified are interpreted as being exclusive (this means that only the specified modifiers can be present when the button event occurs). The following table indicates the values that can be used for ***modifier_name***.

The ALT key is frequently labeled EXTEND or META. ALT and META can be used interchangeably in event specification.

Modifier	Description
CTRL	Control Key
SHIFT	Shift Key
ALT	Alt/Meta Key
META	Meta/Alt Key
LOCK	Lock Key
Mod1	Modifier1
Mod2	Modifier2
Mod3	Modifier3
Mod4	Modifier4
Mod5	Modifier5

The following table indicates the values that can be used for ***button_event_name***.

Modifier	Description
Btn1Down	Button 1 Press
Btn1Up	Button 1 Release
Btn1Click	Button 1 Press and Release
Btn1Click2	Button 1 Double Click
Btn2Down	Button 2 Press
Btn2Up	Button 2 Release
Btn2Click	Button 2 Press and Release
Btn2Click2	Button 2 Double Click
Btn3Down	Button 3 Press
Btn3Up	Button 3 Release
Btn3Click	Button 3 Press and Release
Btn3Click2	Button 3 Double Click
Btn4Down	Button 4 Press
Btn4Up	Button 4 Release
Btn4Click	Button 4 Press and Release
Btn4Click2	Button 4 Double Click
Btn5Down	Button 5 Press
Btn5Up	Button 5 Release
Btn5Click	Button 5 Press and Release
Btn5Click2	Button 5 Double Click

Key events that are used by the window manager for menu mnemonics and for binding to window manager functions are single key presses; key releases are ignored. Key events have the following syntax:

```
key = [modifier_list]<Key>key_name
modifier_list = modifier_name {modifier_name}
```

All modifiers specified are interpreted as being exclusive (this means that only the specified modifiers can be present when the key event occurs). Modifiers for keys are the same as those that apply to buttons. The ***key_name*** is an X11 keysym name. Keysym names can be found in the **keysymdef.h** file (remove the **XK_** prefix).

Button Bindings

The ***buttonBindings*** resource value is the name of a set of button bindings that are used to configure window manager behavior. A window manager function can be done when a button press occurs with the pointer over a framed client window, an icon or the root window. The context for indicating where the button press applies is also the context for invoking the window manager function when the button press is done (significant for functions that are context sensitive).

The button binding syntax is

```
Buttons bindings_set_name
   {
       button     context     function
       button     context     function
                              .
                              .
       button     context     function
   }
```

The syntax for the context specification is

```
context =object[|context]
object =root | icon | window | title | frame | border | app
```

The context specification indicates where the pointer must be for the button binding to be effective. For example, a context of **window** indicates that the pointer must be over a client window or window management frame for the button binding to be effective. The **frame** context is for the window management frame around a client window (including the border and titlebar), the **border** context is for the border part of the window management frame (not including the titlebar), the **title** context is for the title area of the window management frame, and the **app** context is for the application window (not including the window management frame).

If an **f.nop** function is specified for a button binding, the button binding will not be done.

Key Bindings

The ***keyBindings*** resource value is the name of a set of key bindings that are used to configure window manager behavior. A window manager function can be done when a particular key is pressed. The context in which the key binding applies is indicated in the key binding specification. The valid con-

texts are the same as those that apply to button bindings. The key binding syntax is

```
Keys bindings_set_name
{
        key     context     function
        key     context     function
                    .
                    .
        key     context     function
    }
```

If an **f.nop** function is specified for a key binding, the key binding will not be done. If an **f.post_wmenu** or **f.menu** function is bound to a key, **mwm** will automatically use the same key for removing the menu from the screen after it has been popped up.

The context specification syntax is the same as for button bindings. For key bindings, the **frame**, **title**, **border**, and **app** contexts are equivalent to the **window** context. The context for a key event is the window or icon that has the keyboard input focus (**root** if no window or icon has the keyboard input focus).

Menu Panes

Menus can be popped up using the **f.post_wmenu** and **f.menu** window manager functions. The context for window manager functions that are done from a menu is **root**, **icon**, or **window** depending on how the menu was popped up. In the case of the window menu or menus popped up with a key binding, the location of the keyboard input focus indicates the context. For menus popped up using a button binding, the context of the button binding is the context of the menu. The menu pane specification syntax is

```
Menu menu_name
    {
        label   [mnemonic]   [accelerator]    function
        label   [mnemonic]   [accelerator]    function
                    .
                    .
        label   [mnemonic]   [accelerator]    function
    }
```

Each line in the **Menu** specification identifies the label for a menu item and the function to be done if the menu item is selected. Optionally a menu button mnemonic and a menu button keyboard accelerator may be specified. Mnemonics are functional only when the menu is posted and keyboard traversal applies.

The **label** may be a string or a bitmap file. The **label** specification has the following syntax:

```
label = text | bitmap_file
bitmap_file = @file_name
text = quoted_item | unquoted_item
```

The string encoding for labels must be compatible with the menu font that is used. Labels are greyed out for menu items that do the **f.nop** function, an invalid function, or a function that doesn't apply in the current context.

A **mnemonic** specification has the following syntax

```
mnemonic  = _character
```

The first matching character in the label is underlined. If there is no matching character in the label, no mnemonic is registered with the window manager for that label. Although the character must exactly match a character in the label, the **mnemonic** will not execute if any modifier (such as SHIFT) is pressed with the character key.

The **accelerator** specification is a key event specification with the same syntax as is used for key bindings to window manager functions.

Environment

mwm uses the environment variable ***$HOME*** specifying the user's home directory.

Files

/usr/lib/X11/system.mwmrc
/usr/lib/X11/app-defaults/Mwm
$HOME/.Xdefaults $HOME/.mwmrc

Copyright

Origin

HP

Related Information

X(1)
VendorShell(3X)
XmInstallImage(3X)