1

Getting Started

This document introduces you to the PEX-SI release. Chapter 1 describes the available documents and directs you to the documents most likely to help you use or port the PEX-SI. A list of related references is given at the end of this chapter. Chapter 2 describes the directory hierarchy of the source code and how it fits into the X11 source tree. The remaining chapters provide information on how to use the PEX-SI Application Programmer's Interface (PEX-API), the PEX-SI Model Extension to the standard X server (PEX-ME), and the clients and test programs provided with the release.

The appendices include:

- The functionality and implementation-specific parameters of the PEX-ME
- A description of the archive format used for PEX-API PHIGS archival.

1.1. Documents

The PEX-SI provides several documents. All of them can be found in subdirectories of the extensions/doc/PEX/SI directory.

PEX-SI User Guide
This is the document you are now reading. It is in the User_Guide subdirectory.

PEX-SI Graphics Library Manual Pages
This document contains descriptions of the functions provided by the PEX-API. The *Graphics Library Manual Pages* document is made up of what are commonly called ''man'' pages (man is an abbreviation of manual). Each function has its own man page entry.

The PHIGS and PHIGS PLUS functions in this library form an Application Programmer's Interface (API) above the PEX protocol. The man pages describe the functionality of all of the API functions and include implementation-specific information. This document can be printed in its entirety or it can be accessed on-line. See the README file in the PHIGS subdirectory for more information on the man pages.
Note: The entire document is over 800 pages long and may take over an hour to format and print.

PEX-SI Architecture Specification
This document describes the general architecture of PEX-SI. It includes descriptions of the components and features of the PEX-SI design. It is located in the Arch_Spec subdirectory.

PEX-SI Porting Guide
The *Porting Guide* is an extension of the *Architecture Specification*. It contains detailed information on how PEX-SI is implemented and how to customize and port it for your needs. The document is located in the Portg_Guide subdirectory.

1.2. Expected Audiences

The PEX-SI documentation has been designed to satisfy four audiences interested in porting and using various levels of the PEX-SI code:

Compiler
: This person is interested in getting PEX-SI compiled *as is*, and in using, running, or writing clients as quickly as possible. The *User Guide* is the document most useful to this person.

Customizer
: This person is interested in easy modifications of PEX-SI. This may include changing implementation-dependent capabilities and parameters, but does not involve changing code. This person should read the *User Guide* and the sections on customization in the *Porting Guide*.

Porter
: This person is interested in porting PEX-SI to specific software and/or hardware. This includes changing and optimizing the code for specific devices. This person should read the *User Guide*, the *Architecture Specification*, and the *Porting Guide*.

Modifier
: This person wants to change or add significant capabilities to PEX. This person should join the PEX architecture team and mold the PEX Protocol to address their needs.

All of the documentation and code assumes that the reader is familiar with the C programming language, the X11 Window System, PHIGS, PHIGS PLUS, and PEX. References for information on these subjects are given in Section 1.3.

1.3. Additional References

PEX Introduction and Overview (v 3.2; 10 October 1988): an introduction to the goals and structure of the PEX project, with a glossary of PEX terms. *It is widely recognized that this document is out of date.*

PEX Protocol Specification (v 5.0P; 14 September 1990): a description of the protocol extensions to the X Window System that comprise PEX.

PEX Protocol Encoding Document: (v 5.0P; 14 September 1990): the bit bindings of the PEX protocol.

X11 Server Extensions (28 August 1987): a description of the interface between the MIT X11 Sample Server and portable extensions in general.

Definition of the Porting Layer for the X v11 Sample Server (1 March 1988)

Xlib — C Language X Interface (X v11 R4)

X Toolkit Intrinsics — C Language Interface (X v11 R4, March 1988)

ISO/IEC 9592-1:1989(E), Information Processing Systems — Computer Graphics — Programmer's Hierarchical Interactive Graphics System (PHIGS), Part 1: Functional Description

Draft proposed PHIGS PLUS, ISO/IEC SC24-N454 (20 March 1990)

2

Source Tree Description

This chapter familiarizes you with the directory hierarchy of PEX-SI. It describes the PEX-SI files and libraries. It also assumes that you are ‘‘X-literate.’’

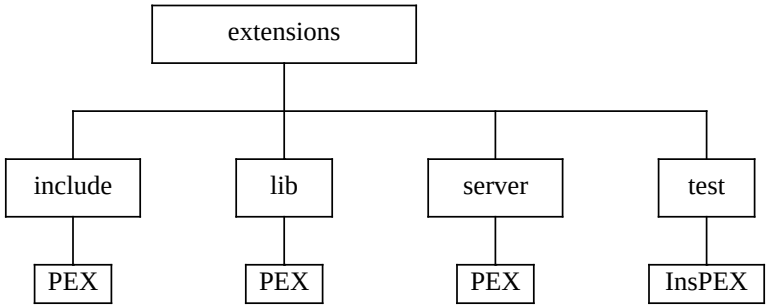

Figure 2-1 *PEX-SI Directory Hierarchy*

2.1. mit/doc/extensions/PEX/SI

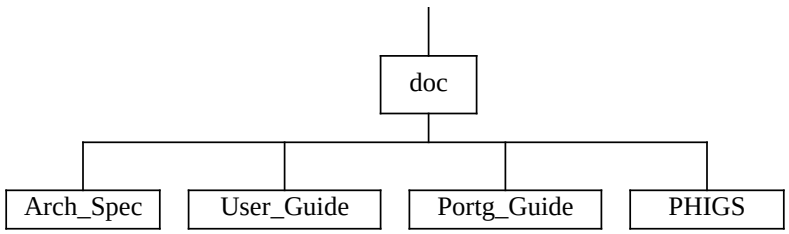

Figure 2-2 *doc* source tree

The doc subdirectory contains the source for, and PostScript versions of, all of the PEX-SI documentation. There is a subdirectory for each document and a README file in each document subdirectory that provides more information on the contents of that subdirectory.

2.2. mit/fonts/PEX

This subdirectory contains the source for the PEX-SI font compiler and the font definitions. These programs are designed to enable a PEX-SI user to define stroke fonts, and to use them within a PEX client. Refer to the *PEX-SI Porting Guide* for more information on using the PEX-SI predefined fonts, and on creating new fonts.

2.3. mit/extensions/include/PEX

This subdirectory contains the client, server, and shared include files. These files define the constants and protocol structures used by the client library and the server for creating and reading PEX packets.

2.4. /mit/extensions/lib/PEX

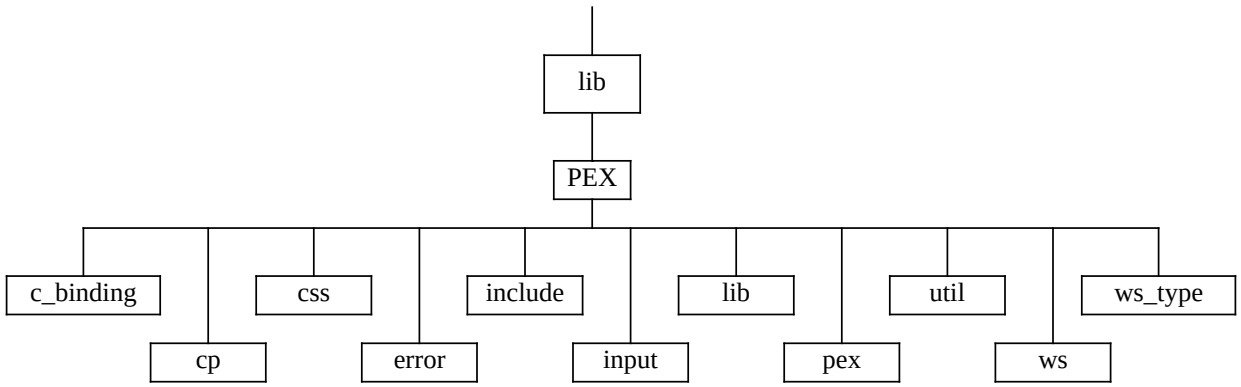

Figure 2-3 *lib* source tree

This subdirectory contains all of the source for the PEX-API library. It contains several subdirectories that separate the files into functional groups. The subdirectories of lib/PEX contain:

c_binding	The C binding. All of the public entry points are contained in this directory.
cp	Internal control and distribution code, and the PHIGS Monitor. The source code in this subdirectory directs PHIGS operations to the appropriate resource, and handles toolkit initialization and top level event processing.
css	The CSS source code for client-side structure storage.
error	The internal error buffering and reporting source code.
include	The public and private header files.
input	The input device control and realization source code.
lib	The run-time files.
pex	The PEX protocol wrappers. This code stands by itself and can be used independently of the rest of the library. Not all wrappers reflect 5.0P protocol, only the ones that the PEX-API uses.
util	Miscellaneous utility code used by any of the other directories.
ws	The workstation control and management code.
ws_type	The Workstation Description Table (WDT) and workstation type code, and the workstation type attribute setting code. The source code in this subdirectory handles WDT initialization and modification.

For more information on the contents of these subdirectories, see the *PEX-SI Porting Guide*.

2.5. mit/extensions/server/PEX

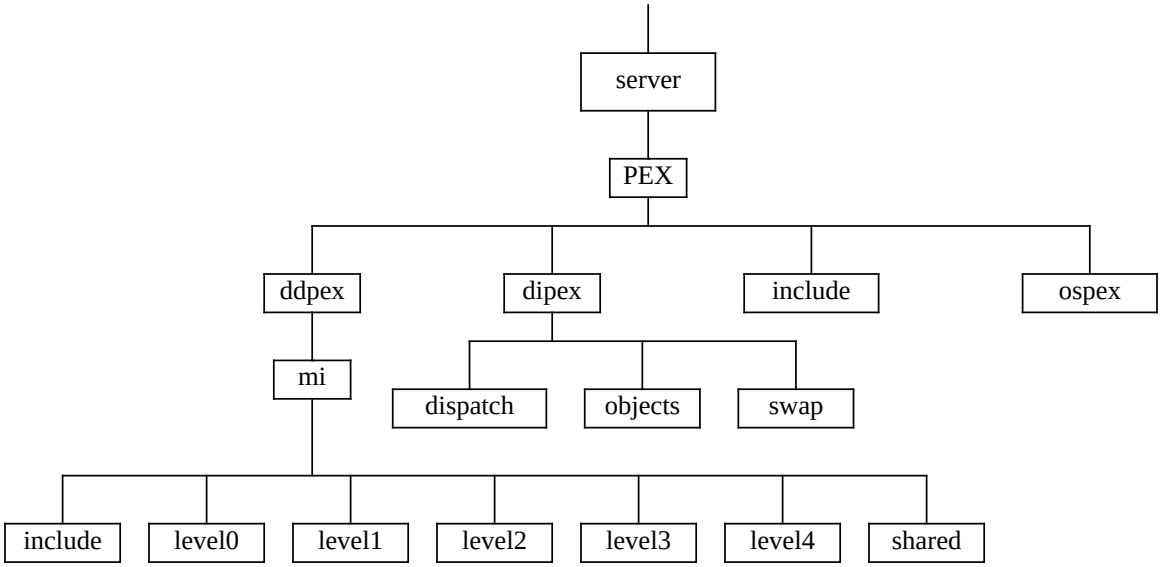

Figure 2-4 *server* source tree

The server subdirectory contains all of the source for the PEX extension to the X server. It has subdirectories for the server include files, the device-independent code (diPEX), the device-dependent code (ddPEX), and the operating system dependent code (osPEX).

The ddpex subdirectory is intended to contain subdirectories for different vendors. The PEX-SI code provides only one subdirectory, which contains the machine-independent code (mi). The mi subdirectory has divided the PEX-SI ddPEX code into several subdirectories that correspond to the design of PEX-SI.

The ddpex/mi/include subdirectory contains all of the files that are included by the ddpex source code. These files are not included by any other PEX-SI code.

The ddpex/mi/level0 subdirectory is a place holder for implementations that require a foundation layer other than ddx. PEX-SI does not use this subdirectory.

The remaining subdirectories correspond to the five modules that make up the PEX-ME code. (See the *PEX-SI Architecture Specification* for a description of each of these modules.)

ddpex/level1	The 3D Rendering Module
ddpex/level2	The Transformation Module
ddpex/level3	The Rendering Control Module
ddpex/level4	The PHIGS Workstation Module
ddpex/shared	The Shared Resources Module

The dipex subdirectory divides the diPEX source into three subdirectories. The dipex/dispatch subdirectory contains source code that ‘‘dispatches’’ requests to the swap and objects subdirectories as required. It also contains the diPEX initialization source code, and the source code for utilities that can be used by ddPEX and osPEX. The dipex/objects subdirectory contains all request handling source code. This code routes requests to the appropriate ddPEX module.

The dipex/swap subdirectory contains byte-swapping and floating-point conversion code.

The ospex subdirectory is designed to contain subdirectories for specific operating system code.

2.6. mit/extensions/test/InsPEX

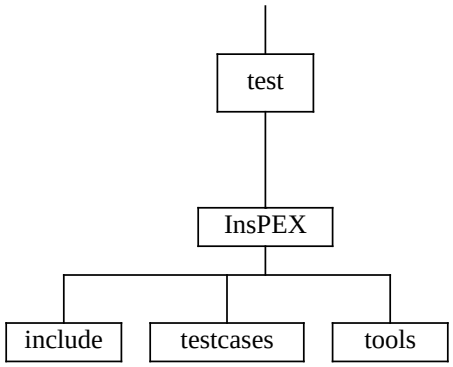

Figure 2-5 *test* source tree

This test directory contains sources for the InsPEX test suite.

InsPEX is an automated test suite that uses the PHIGS/PHIGS PLUS language binding to test the PEX-SI implementation. It performs operations via calls to the PHIGS functions, and tests for the expected results using Xlib and the PHIGS binding itself. Information on using InsPEX is in Chapter 5. There is also a README in the InsPEX directory that covers the information that is in this document.

The main execution script, inspex.sh, resides in the InsPEX directory. The three subdirectories include, tools, and testcases contain supporting programs for the main execution script. The InsPEX/include subdirectory contains implementation-dependent include files. These files should be modified to suit a particular implementation before the suite is run.

The InsPEX/testcases subdirectory contains subdirectories for several tests written either in C, or in the interpreter language.

The InsPEX/tools subdirectory contains subdirectories for three supporting programs: inspector contains the source for *Inspector*, the image comparison tool; newt contains the library-independent source for the interpreter; and pexint contains the InsPEX-specific source for the interpreter.

2.7. mit/demos/gpc

This directory contains the source for the Graphics Performance Characterization (GPC) programs. It has three subdirectories: benchmarks contains the GPC approved benchmarking files that can be used for gauging the performance of your implementation; objects contains other demo programs; and tests contains reference images that can be used to verify an implementation, and documentation describing the images. A README file describing use limitaitons and how to use the benchmarking programs is located in the gpc directory. For additional instructions and information, see Chapter 6.

3

Using PEX-API

The PEX-SI PHIGS graphics library provides the client-side support for PEX-SI. It uses PHIGS and PHIGS PLUS as the Application Programmer's Interface to the PEX protocol (PEX-API). The complete set of functions available in the PEX-API are described in the *PEX-SI Graphics Library Manual Pages*. The binding for PEX-API nearly conforms to the IS PHIGS C Binding (ISO DIS 9593-4:199x(E), 1 August 1990); however, some deviations have been made. This information (as well as other important information) is described in the INTRO PHIGS manual page.

Because a C binding has not yet been defined for PHIGS PLUS, a binding was developed to match the form of the PEX-SI PHIGS binding.

There are several files needed by the client to use the PEX-SI graphics library.

libphigs.a
: The PEX-API module that is linked with your client.

phigsmon
: The PHIGS Monitor (PM) is a run-time module that runs as a child process of the client. The PM is automatically started by PEX-API when a client is run. Note that the PM is optional. (See the OPEN PEX Graphics Library man page for more information.)

PHIGSerr.txt
: A run-time support file needed by PEX-API.

PHIGSfnc.txt
: A run-time support file needed by PEX-API.

phigs.h
: The header file that must be included when you are using the library. phigs.h itself includes some X header files.

The PM and support files can be found in the (default) subdirectory /usr/lib/X11/PEX, and the header file phigs.h can be found in /usr/include/X11/phigs. See the Imakefile in the various mit/demos/auto_box or mit/demos/beach_ball subdirectories for examples of building PEX-SI clients.

Note: If you did not make install when you built the PEX-SI, you will find the API library and the PHIGSfnc.txt and PHIGSerr.txt files in the mit/extensions/lib/PEX subdirectory, and the PM and the phigs.h file in the mit/extensions/lib/PEX/lib subdirectory below it.

Note: You must provide include paths for the X and PEX header files when you compile your clients. You also need to link the PEX library and the standard X library, libX11.a, with your clients when you build them. If the PEXAPIDIR environment variable is set the API looks for the run-time files there, otherwise it looks in the /usr/lib/X11/PEX directory. If this environment variable does not exist, or the files are not in that directory, an the OPEN PHIGS function generates an error message and the client exits.

The PM is a child process of a PEX client. The PM performs the following:

- Processes window system and input device events associated with a client's PHIGS workstations.
- Implements the window system event response policy.
- Issues the required PEX and X calls to enforce that policy.
- Maintains the PHIGS input event queue and services event requests and inquiries from the client side API code.
- Contains the Central Structure Store (CSS) for PHIGS workstations using servers that do not support structure storage.

The API communicates with the PM through UNIX sockets by default; the compiler symbol PEX_API_SOCKET_IPC is defined in the *.cf* configuration file. To force the client to use System V shared memory instead of UNIX sockets, you must remove the PEX_API_SOCKET_IPC definition before the Graphics Library is compiled.

Running PEX-SI with the PM is optional. Instructions for using this option are given in the OPEN PEX man page and in the *PEX-SI Porting Guide* (in the section on customizing PEX-API).

4

Using PEX-ME

The PEX-SI server code provides a Model Extension (PEX-ME) to the standard X server. It is made up of several libraries that are described in the *Architecture Specification* and the *Porting Guide*. These libraries are linked with your X server if you have chosen to do so at build time. See the section on potential problems in the *Porting Guide* if you have had problems building the PEX-SI server.

PEX-ME is started exactly the same way as the standard X server. See the man pages provided for the server with the X release for additional information. PEX-ME looks for and loads a default PEX font when the server is started. This font is placed in the default directory /usr/lib/X11/fonts/PEX when a make install is done. If the font is not found there, PEX-ME looks for the environment variable PEX_FONTPATH for a path to the font. If the font cannot be found there, or that variable is not defined, the server is started but PEX-ME is not initialized and PEX requests return an error.

PEX-ME follows the same search criteria when looking for PEX fonts to load for the PEX OPEN FONT request. An error is returned for this request if the font cannot be found.

5

Using InsPEX

5.1. Introduction

This chapter introduces InsPEX, the PEX-SI test suite, and gives detailed instructions on how to use it and how to interpret the results from using the tests.

5.1.1. Purpose of the InsPEX Test Suite

The goal of InsPEX is to test compliance of the PEX-SI PHIGS/PHIGS PLUS C binding (in connection with a PEX server) with the PHIGS and PHIGS PLUS standard. It is not intended to verify PEX at the protocol level. Nor is it intended to be a complete PHIGS conformance test suite, or to be a useful performance metric.

InsPEX is intended to be portable and useful for developers porting the PEX Sample Implementation (PEX-SI) to different hardware platforms.

5.1.2. Description of Test Methods

InsPEX uses two techniques to test the PEX implementation:

- Completely automatic testing, using PHIGS inquiry functions to read back system states and data structures, and X functions to read back pixel values.
- Testing through image comparison, comparing what is rendered on the screen (retrieved with X functions) with a set of known good images. Slight differences in implementations may require a one-time manual calibration, in which case a user must inspect images flagged as different and mark them as acceptable or not acceptable. An X-based tool is provided for this purpose.

Many tests use an interpreter for faster execution time and more efficient test development. Tests requiring verification of pixels use Xlib to retrieve pixel and color map information. The resulting images are either verified automatically or are compared against saved images. An X-based tool, Inspector, is supplied for comparing images against saved reference images and saving the new images as acceptable images if desired.

5.1.3. The NewT Interpreter

In order to reduce the total execution time of the InsPEX test suite, as well as to shorten the edit-compile-run cycle for test developers, many tests in InsPEX are written for the NewT interpreter. NewT (for New Tool) is a package developed at Sun Microsystems, Inc., for the creation of special-purpose interpreters. The interpreter language's syntax contains elements of csh and C and is tailored for testing. InsPEX contains as an integral part *pexint*, an interpreter for PHIGS and graphics testing functions. The source files for *pexint* are included, so that the interpreter's functions can be modified and extended.

A man page in the NewT source directory includes an explanation of how the interpreter is built, a description of the language syntax, and instructions on how to use the interpreter.

5.1.4. Image Comparison and the Inspector

Unlike X, the PEX standard does not contain rigid requirements for pixel accuracy; it is expected that different hardware and software implementations will render primitives differently. In order to provide the test suite with the flexibility to handle wide-ranging implementations, an image comparison method is used.

The InsPEX test suite is supplied with a set of reference images for the tests that use image comparison. When the test is run, the image retrieved from the screen is compared against the reference image. If it matches, the test succeeds; otherwise, the test fails and the non-matching image is saved for later examination. The test failure message in the log will state that image comparison is required.

After the test has been run, the suite user can run Inspector, an X-based tool for comparing images from test suite runs against the reference images. Inspector displays the images side by side and provides a function for highlighting differences between the images. If the user decides that the differing image is acceptable, he can accept the image and move it into a directory of local reference images. Subsequent runs of the test can then compare against this local reference, and pass if they match the previously approved image. If the user marks the image under comparison as rejected, subsequent test images that match this image are logged as matching a known bad image.

The recommended test sequence, then, is to do a complete run of the test suite, invoke the Inspector and verify or reject any images that do not match the references; then, do another run of InsPEX that takes into account the results of the Inspector session.

5.2. Building the Suite

The InsPEX suite is not built automatically when the rest of the PEX-SI is built. The image comparison tool, inspector, must be compiled separately as described below. The interpreter and the C-based tests are meant to be compiled by the test suite's run script, inspex.sh. The interpreter is preserved after compilation, but the individual C tests are deleted after they are compiled and run to save disk space. (The -keep command line argument will preserve the individual C tests. For more information on -keep see Table 5-2 in Section 5.3.2).

If the user wishes to compile pexint separately, this can be done using the Makefile in the tools/pexint directory. Please examine this Makefile to ensure that all paths are properly specified on your system, and note that it is vital to supply CPPFLAGS=-DNEWT as an argument to make in order for the interpreter to be built correctly.

5.2.1. Compiling Inspector

Versions of Inspector are available using the XView toolkit supplied on the contrib tape for X11R4, or the Athena Widgets. All toolkit-specific code is isolated in two files by a toolkit-independent interface, so porting to other toolkits should be straightforward. Note that the tool does not itself use PEX — it only uses toolkit and Xlib functions for image display and user interaction. Thus, it need not be run on the server or system under test: it can be run any time, on any machine running an X server.

The Makefile in extensions/test/PEX/InsPEX/tools/inspector will build the Inspector. The Makefile requires two environment variables to be set when run. INSPEXDEST is the name of the directory in which to put test suite output. This variable must also be set when running the test suite. XVIEWHOME is where the libraries and include files for Xview can be found. In addition, there are two macros defined in the Makefile that you may want to set. DESTBIN is the directory where the objects will be placed. The default is set to $(INSPEXDEST)/bin. If INSPEXDEST is not defined, the objects will be placed in /bin. XVLIBS defines the name(s) of the XView library to link with. Some versions of Xview split the library into two pieces, requiring {-lxvin -lxvol} or {-lxview -lolgx} to be specified, instead of the single library -lxview.

5.3. Running the Suite

5.3.1. Environment Variables

In addition to any other environment variables that may be required to run a PEX program, such as DISPLAY and PEXAPIDIR, the suite requires the following two variables; if they are not defined the script will complain and abort. These, and the other file path variables, should be absolute paths.

INSPEXHOME
: Path to the InsPEX test suite's directory, where inspex.sh and the associated subdirectories are. In the PEX-SI source tree, INSPEXHOME is the directory extensions/test/InsPEX.

INSPEXDEST
: Destination directory for test suite output. Compilations, new images, and log files are placed under this directory by the test suite. There should be plenty of available space.

In addition, the current PATH should include the location of the X utility xdpyinfo.

The following environment variables are optional:

INSPEXLOG
: The full pathname of log file for the output of inspex.sh. The default is to assemble a log file name from the month, day, and time, taking the form: $INSPEXDEST/ilogMM.DD.HHMM, (for example, $INSPEXDEST/ilog02.21.1232)

INSPEXIMAGEREF
: The master reference image directory. The default is $INSPEXHOME/refimages.

PEXLIBDIR, PEXINCDIR
: Directories containing the PEX include and library files. These default to ../../lib/PEX and ../../../X11/phigs, respectively (absolute paths are derived from these relative ones by inspex.sh). Note that the suite expects the PHIGS include files to be in a subdirectory of $PEXINCDIR named phigs.

XINCDIR, XLIBDIR
: Directories containing the X11 include directory and the library files. These default to $PEXLIBDIR and $PEXINCDIR, respectively. The suite expects to find the X11 include files in the directory $XINCDIR/X11.

I_CFLAGS, I_LDFLAGS
: These variables can be set to any additional required compiler flags for the compile and link phases, respectively; they will be used in the compilation of the interpreter and C-based tests. If there are multiple words in these variables, they should be enclosed in both single and double quotes so that the grouping isn't lost when these are passed to make. (For example, csh% setenv I_LDFLAGS "'-Bstatic -f68881'".)

SERVER_RESTART_CMD
: If the server dies during test execution, and this variable is set, the script will attempt to use the command to restart the server before continuing. The command should exit after starting the server, since the test script does not run it in the background.

The following optional environment variables are used by some or all of the test programs:

MAXFAIL
: The number of failures a test will accept before giving up and aborting; the default is 5.

VERBOSITY
: Can be set to 0, 1, 2, or 3; specifies various levels of tracing and diagnostic output. Any level implies the ones below it as well. Table 5-1 shows the various levels.

Table 5-1 *VERBOSITY Levels*

Level	***Description***
0 (default)	Prints failure messages of the form <testname>: failed: <description of failure> on failure, and prints a PASSED or FAILED message on completion, with a one-line description of the test.
1	Some tests will give a more detailed or complex description to help identify a failure
2	Enables some higher-level test diagnostics
3	Enables diagnostics in some lower-level test functions

PAUSE

If this variable is set, some rendering tests will pause after drawing and prompt for a carriage return, giving the user a chance to view what is on the screen. This variable should *not* be set when executing the test script itself since the prompts will be redirected into the script and will not be seen.

SHOWPIX

If set, most graphical tests will show which pixels are being sampled by drawing a box around the sampled area. Useful for analyzing failures, but may cause subsequent false failures by putting incorrect pixel values on the screen.

MACROPATH

Interpreter scripts search this path for macro files. The default set by inspex.sh is .:$INSPEXHOME/testcases/shared, causing it to check the current directory and then the directory containing shared macro files. You should not need to set this unless you want to substitute different macro definitions, or if you are running the pexint interpreter outside of inspex.sh, in which case you will have to set it to a value such as the one above to run any InsPEX test scripts.

I_CLIENT_SIDE

If set, this variable will direct most tests to call *popen_xphigs()* with appropriate arguments to specify client-side structure storage, rather than calling *popen_phigs()*, which would give the default behavior, server-side storage.

I_NO_MONITOR

Although single-process execution (without phigsmon) is not yet implemented, I_NO_MONITOR will cause most tests to call *popen_xphigs()* with the flags set to force this behavior, in a manner similar to I_CLIENT_SIDE.

5.3.2. Command Line Arguments

The script inspex.sh can take various command line arguments:

Table 5-2 *inspex.sh Command Line Arguments*

Argument	***Description***
-display	Runs the test on the named display's server.
-a <test-area-names>	The user may specify one or more specific test areas; all subsequent arguments up to the next option flag or the end of the line are taken to be test areas. Without this flag, all test areas will be run.
-t <indiv-tests>	Runs the specified tests. The test file names supplied may be in several forms: full path names, test file names alone, or in the form areaname/testname. The file name extension (.c or .pi) is optional. The test name alone will generally be enough if the tests have not been moved from their test area directories. This option can be used alone or in combination with the -a option above to run both individual tests and some entire test areas.
-keep	Keep executables for C-based tests after execution, rather than removing them. If executables are very large, this option will use a lot of disk space, but may be desirable for debugging. The executables are left in $INSPEXDEST/tmp.
-server_side, -client_side	These options will force server-side or client-side structure storage, respectively, in most tests. The same result can be obtained by setting or unsetting the environment variable $I_CLIENT_SIDE described above, though these switches override the value of that variable.
-clean	Before beginning test execution, empty out the bin, lib, and tmp directories under $INSPEXDEST in order to get a clean run. This option is recommended if testing a new version of the PEX library.
-clean_all	This does all of the above, plus deletes everything under $INSPEXDEST/include, where pdt.h and wdt.h live. This should only be used if the user has not customized these files under $INSPEXDEST to suit particular PHIGS and workstation capabilities. (These files are described in the next section.)

The command line options are also documented in a comment at the top of the script; that comment should be considered the most up-to-date description.

5.3.3. Implementation-Dependent Files

Two include files, pdt.h and wdt.h, are used to specify implementation-dependent information about the expected contents of the PHIGS Description Table and the Workstation Description Table, respectively. When the script inspex.sh is executed, it checks for the existence of these files in the directory $INSPEXDEST/include; if they do not exist, they are copied from $INSPEXHOME/include. A set of C-based tests includes these files, and tests that the inquired values match the expected.

If the expected default contents of these description tables are different than those for PEX-SI, the pdt.h and wdt.h files should be copied from $INSPEXHOME/include to $INSPEXDEST/include, and modified appropriately. In this case,

however, the user should be careful not to use the -clean_all option of inspex.sh as this option would cause any modifications to the pdt.h and wdt.h files to be overwritten.

5.3.4. Runtime Environment

As mentioned previously, the X utility *xdpyinfo* must be found in the current command search path.

The server should not be running a window manager, unless the window manager is one that does not wait for the user to position new windows (*olwm* is one such window manager). Even with such a window manager available, one should consider running without it as it may influence window placement, which in turn may influence the rendering of primitives. A window manager may be necessary, however, if the PEX library is ICCCM compliant and depends on the window manager to load its color map.

PEX-SI provides two scripts in the mit/extensions/test/InsPEX directory that will help you to set up the runtime environment for InsPEX:

startpex This script sets the environment variables and executes the command that starts the server. The script contains several environment variables that identify the root of the X source tree, the name of the server executable, paths of other executables needed when running the server, and paths of the font files needed by the server. You will need to set the variables in this script to suit your runtime environment before running InsPEX. Default values are provided in the script.

run_InsPEX This script sets the necessary environment variables and calls inspex.sh to run the InsPEX test suite. If a PEX server is not already running when this script is executed, one will be started. A log file is created in the directory specified by the environment variable INSPEXLOG. You will need to set the variables in this script to suit your runtime environment before running InsPEX. Default values are provided in the script.

5.3.5. Testing From Within the Server

The preferred way to run inspex.sh is to execute it from a screen or terminal other than the one that the PEX server is running on. If the suite is not being run from another screen or terminal (that is, if it is being started from a window on the server being tested), there is the potential problem of a server crash killing the test suite process before the test script has a chance to restart the server with SERVER_RESTART_CMD.

To solve this problem, we have supplied a utility called newpgrp (for NEW Process GRouP), which will execute a command in a process that is disassociated from the current process group. newgrp takes a command name and a list of arguments on the command line to be executed in the new process, for example:

```
host% newpgrp inspex.sh -a archive strcon
```

This utility is a short C program residing in .../test/InsPEX. newgrp can be compiled with the following command:

```
host% cc newpgrp.c -o $INSPEXDEST/newpgrp
```

5.4. Analyzing the Results

5.4.1. Format of Log Messages

The test script inspex.sh places the output in a file called ilogMM.DD.HHMM, (for example, ilog02.21.1232), in the $INSPEXDEST directory. The script records information such as the date, machine name, and value of the DISPLAY variable. If any make operation fails, the output of the make is recorded in the log file. Then the output of each test is logged.

After completion, every test outputs a message containing the test name, PASSED or FAILED, and a brief description of what is tested. If a failure is detected, a message of the form:

<testname>: failed: <description of what happened>

is printed. The test will continue on if possible, until the number of failures reaches $MAXFAIL. If a test cannot be compiled, causes a core dump before completion, or crashes the server, this is detected by inspex.sh and is also logged (inspex.sh uses the X utility *xdpyinfo* to verify the health of the server).

If a test prints a message containing the word ERROR, this means that something unanticipated has happened — either an error in a system or testing function, or a failure in the library under test other than what the test is looking for (for example, if a rendering test cannot open a workstation). Such messages should also contain some descriptive text describing what happened. We consider a *failure* to mean that the test operated properly and found something wrong in the library being tested; we consider an *error* to mean that the test did not work as expected — something was wrong in the test software, the system in general, or in the library functions that the test assumed would work.

5.4.2. Image Comparison Failure Messages

If a test uses the image comparison method, and an image does not match the local or master reference image, several different failure messages can result:

<testname>: failed, pending manual comparison of image <imagename>
: This means that the image did not match the reference image found; we do not know whether it is correct or not. It also does not match any saved image that is marked as incorrect. The comparison tool, *inspector*, must be used to compare the image to the reference.

<testname>: failed - image comparison (<imagename> matches known bad image)
: This means that the image did not match the reference image, but did match an image in the current images directory that has been marked as rejected; we know that this image is incorrect and need not look at it again.

<testname>: failed - no reference image for <imagename> (saving current image)
: This message should not be seen unless the test case is new, or the reference image is for some reason not present. If it is a new test case, this saved image should be viewed with the comparison tool — if it is acceptable, it can be accepted into the local reference, and then manually copied into the master reference if desired.

The image names mentioned above may not be the same as the test file name, since one test file might create several images. The image name corresponds to the name of the file the image is stored in.

The images are stored in a binary run-length-encoded format developed for InsPEX; pixels are stored along with color map information, so that comparison of images can be done by comparing the RGB values that were in the color table for each pixel. The format is described in a comment in $INSPEXHOME/tools/pexint/pixutils.c, just before the definition of the function *i_save_image()*.

5.4.3. Running Inspector

The *inspector* comparison tool requires that the environment variables INSPEXDEST and either INSPEXHOME or INSPEXIMAGEREF be set (see Section 5.3.1). The tool prints a record of its actions on its standard output — the user will probably want to redirect this into a log file.

On startup, the tool displays two empty boxes, labelled *reference image* and *current image*. To the right of these boxes is a current file name, a current file status (either REJECTED or UNSEEN), a menu with a set of buttons, and a scrolling list of file names. The file name list contains the names of the image files in $INSPEXDEST/currimages; the user may scroll this list and select a file name. The meaning and action of the buttons is as follows:

Accept Active only when an image is displayed. Marks the image as accepted by moving it into $INSPEXDEST/localrefimages, deleting from currimages, and removing any REJECTED marking from the file. The boxes, status line, and file name are cleared (since the file is no longer in currimages, it is no longer under consideration).

Reject Active only when an image is displayed. Marks the image's file as being incorrect by appending a hyphen (-) to the filename. The status line becomes REJECTED.

Mark Unseen Active only when an image is displayed. Marks the image's file as unseen by removing a hyphen (-) from the file name, if present. The status line becomes UNSEEN.

Show Diffs This button redisplays both images so that only those pixels that are different from the corresponding pixels in the other are displayed. The rest of the pixels are set to the background color. This button is a toggle; when pressed, the label changes to Show Normal. Pressing it again reverses the action.

Load This button loads the image file currently selected in the scrolling list. This image is displayed in the right box. If a reference image exists in either $INSPEXDEST/localrefimages (checked first) or $INSPEXHOME/refimages, it is displayed in the left box. The pathname of the reference file is printed below the left box, or No reference image available if no reference image is found. Pressing this button again is a good way to force a redraw of the images.

Quit Quits the application.

Below the images is the pathname of the reference image currently being displayed and a line of text describing what the image should look like.

The Inspector follows the X Window System Inter-Client Communications Convention Manual (ICCCM) Version 1.0, Section 6.4 in the way it relies on the window manager to install the colour maps for the two image subwindows. The color maps for these windows should be loaded by the window manager when the mouse is moved into each subwindow. If the window manager in use is not ICCCM-compliant, it may not correctly load the color maps, and the images will not be shown in the correct colors. If this is the case, there is a crude workaround: setting the environment variable INS_MULTIWIN before starting the tool. If this variable is in the environment, the tool creates the image windows as separate X windows (children of the root). The user may have to arrange these windows manually into good positions, but the window manager should be able to handle the color maps correctly. The tool's main window will initially appear over the image windows, but loading an image will raise the image windows to the top.

Note that it is generally necessary to move the mouse into the image subwindows to force the loading of colormaps and see the images in their correct colors.

5.5. Running Tests Outside the Script

Though the InsPEX test suite as a whole is meant to be run using the script inspex.sh, it is sometimes desireable to run individual tests from outside the script, especially when trying to analyze a failure. The environment variables PAUSE and SHOWPIX are also most useful when running a test individually, without the overhead of starting a test run.

5.5.1. Compiling and Running C-Based Tests

The script inspex.sh runs C-based tests by copying them to the directory $INSPEXDEST/tmp and compiling them using the Makefile there (the Makefile is copied there from $INSPEXHOME/testcases/shared if it does not already exist.) The user can compile the C-based tests manually, after examining the Makefile and ensuring that all values are set properly there or on the make command line.

The C-based tests make use of the library libinspex.a, which is built in $INSPEXDEST/lib. If this library does not already exist, it will be built automatically, provided that $INSPEXHOME and the other variables are set properly.

5.5.2. Running Interpreter Scripts

Once the interpreter is compiled, either manually or by inspex.sh, all that is required to run it on a test script is to make sure that all the necessary environment variables normally set by inspex.sh are set in the current environment. The most important of these, beyond those required to run inspex.sh itself, is MACROPATH — this is a colon-separated list of directories in which the interpreter looks for other interpreter files that are read using the *source* interpreter command. For InsPEX tests, all macro files are either in the test area directory itself, or in $INSPEXHOME/testcases/shared, so the correct value for this variable is ‘‘.:$INSPEXHOME/testcases/shared.’’

The script inspex.sh builds the interpreter pexint in $INSPEXDEST/bin. pexint can be run as a command with the test script name as an argument. The scripts are run most easily by first changing to the test area’s directory under $INSPEXHOME/testcases. For example:

```
csh% cd $INSPEXHOME/testcases/utils
csh% alias pi $INSPEXDEST/bin/pexint
csh% pi rotate.pi
```

The manual page for the interpreter’s language, construction, and usage is in tools/newt/newt.man; it describes the various options and capabilities of the interpreter, including tracing, source-level debugging, and automatic C translation. The -t tracing option can be useful for tracking down where a crash occurs: it tells the interpreter to print out each function call before it is made. The manual page also describes the interpreter language, and will be useful if you want to modify tests or add your own.

6

Using GPC

6.1. Overview

Graphics Performance Characterization (GPC) is a set of programs intended to standardize graphics performance measures by establishing a benchmark to be used in comparing the performance of various graphics platforms. GPC is an effort that has been underway for more than four years, with several leading industrial sponsors (Sun, IBM, DEC, HP, and others) led by the NCGA (National Computer Graphics Association).

The performance characterization is achieved by a set of programs called the PLB (Picture-Level Benchmark) programs. These programs are designed to measure the performance of graphics display on a screen. SimGraphics of Pasadena, California, has been the primary developer of PLB programs. The current version is Release 1.1 from SimGraphics and NCGA.

The SimGraphics PLB programs have been ported to the PEX-API, and the ported versions are included in the PEX-SI release. These programs have been provided as a cooperative effort between NCGA and the PEX-SI project. There are use restrictions and warnings associated with the use of these programs for other than implementation testing purposes. Basically, the restrictions are twofold:

- These sources are *not* to be used to measure and subsequently publish performance statistics in any way that might relate to the actual measurement methods being used by the NCGA-based GPC group.
- These tests and sources are subject to change without ntoice by SimGraphics or NCGA

For the most up-to-date programs, source, and information about participation in the GPC benchmarking efforts, contact NCGA or SimGraphics at the following addresses:

National Computer Graphics Association
2722 Merrilee Drive
Suite 200
Fairfax, VA. 22031
(703)698-9600

SimGraphics Engineering Corporation
1137 Huntington Drive
Suite A
South Pasadena, CA. 91030
(213)255-0900

The PLB programs consist of three parts:

1. Benchmark Interface Format (BIF)
2. Benchmark Timing Methodology (BTM)
3. Benchmark Report Format (BRF)

The BIF consists of two parts: the BIF Geometry file and the BIF Verb file. The BIF Geometry file contains structures that define the geometry and attributes of the data to be displayed. The BIF Verb file contains the commands that control the PLB program flow.

The PLB programs included with PEX-SI distribution are complete with the exception of support for the NURB Curve and Surface primitives and the PIXEL_MAP primitive. The BIF geometry files provided in this release test the majority of the PHIGS primitives and attributes. When used, these files generate displays on the screen in a predetermined sequence. The program delivered with PEX-SI is called *plbpex*.

6.2. Running *plbpex*

If the *plbpex* program has not been built automatically, you can create it by typing make all in the mit/demos/gpc directory. The executable created is named plbpex.

To run the GPC program, you must be in the objects subdirectory, and you must have write permission in that directory because the program will be creating and writing to an error file. Type:

```
host% ../plbpex [-display geometry bd bg bw hlhsr buff] filename
```

The filename must be one of the *.vrb* files located in the objects directory. The options are defined as follows:

Table 6-1 *plbpex Options*

Command Line	***Resource***	***Value***
-display	.display	String for X display connection.
-geometry	.geometry	Size and position in X notation. (Width and Height will override WINDOW_SIZE in the CONFIGURATION.)
-bd	.borderColr	Color for the window border in X color notation.
-bg	.background	Color for the window background in X color notation. (Will override any BACKGROUND_COLOR or BACKGROUND_COLOR_INDEX in the verb file.)
-bw	.borderWidth	Width of the window border in pixels.

Table 6-1 *plbpex Options— Continued*

Command Line	*Resource*	*Value*
-hlhsr	.hlhsrMode	One of the following (case is not significant): ZBUFF, PAINTERS, SCANLINE, LINE. These modes may not exist on every platform.
-buff	.bufferMode	One of the following (case is not significant): SINGLE, DOUBLE. Will not override buffer mode in the CONFIGURATION.

plbpex creates its own window and uses the same methods as the API for looking for the correct visual. It uses the **.geo* and **.bif* files as it runs.

The following is an example of the contents of the error file that *plbpex* creates:

```
PHIGS error 111 in SET HLHSR MODE: Ignoring function, the specified HLHSR mode
is not available on the specified workstation
PHIGS error 111 in SET HLHSR MODE: Ignoring function, the specified HLHSR mode
is not available on the specified workstation
PHIGS error 111 in SET HLHSR MODE: Ignoring function, the specified HLHSR mode
is not available on the specified workstation
PHIGS error 111 in function number 73
PHIGS error 111 in function number 73
PHIGS error 111 in function number 73
PHIGS error 111 in function number 73
PHIGS error 111 in SET HLHSR MODE: Ignoring function, the specified HLHSR mode
is not available on the specified workstation
PHIGS error 111 in SET HLHSR MODE: Ignoring function, the specified HLHSR mode
is not available on the specified workstation
PHIGS error -1 in OPEN PHIGS: Ignoring function, cannot open PHIGS, cannot
create communication channel
PHIGS error 111 in SET HLHSR MODE: Ignoring function, the specified HLHSR mode
is not available on the specified workstation
PHIGS error 111 in SET HLHSR MODE: Ignoring function, the specified HLHSR mode
is not available on the specified workstation
PHIGS error 111 in SET HLHSR MODE: Ignoring function, the specified HLHSR mode
is not available on the specified workstation
```

plbpex will exit when it has finished evaluating the specified file. Depending on the *.vrb* file selected, the runtime will vary from a few minutes to more than an hour.

7

Using the Clients

All PEX clients and X clients are run in a similar way. The environment variable DISPLAY should be set to the screen or display that you wish to use (for example, unix:0). If you have done a make install, these clients can be run as described below. If you have not done a make install, you need to set the environment variable PEXAPIDIR to the directory where the support files used by the PEX-API are located. (See Chapter 3 for a description of these files.)

7.1. Demos

This directory contains simple PHIGS test programs developed by the PEX-SI team, including those presented at the 5th Annual X Conference, Boston, January 14-19, 1991. All of these programs can be used as demos of PHIGS applications.

7.1.1. Auto Box

auto_box is a simple program that displays four views of a rotating cube. It initially displays the cube with filled polygons, then with hollow polygons, or polylines. Four views of the cube are presented in four quadrants, each with a different rotation applied. In the first and fourth quadrants the back faces and front faces are respectively culled out, demonstrating one of the PHIGS PLUS features. *auto_box* uses only PHIGS indexed colours. To run the program move to the directory mit/demos/auto_box and type auto_box. It exits after all views and rotations are completed once.

7.1.2. Beach Ball

beach_ball is a simple program that displays a bouncing multicolored pseudo-sphere (a beach ball). It uses filled polygons for the faces of the beach ball. To run the program move to the directory mit/demos/beach_ball and type beach_ball. You must type Control-C to exit this program, as it will run in a continuous loop.

On server implementations that have not been optimized, it may be hard to visualize the bouncing motion of the ball. Because PEX requests may be buffered in queues, there may be a delay between typing Control-C and the client exiting.

A

PEX-SI Server Functional Description

This appendix specifies the implementation-dependent functionality of the PEX-SI server extension. The following sections cover Enumerated Types, Implementation-Dependent Constants, Renderer Dynamics, PHIGS Workstation Dynamics, and Lookup Tables.

A.1. Enumerated Type Information

Tables A-1 through A-23 list the enumerated types supported in the PEX-SI server.

A.1.1. MarkerType

Table A-1 *MarkerType*

Value	*Name*	*Symbol*	*Description*
1	Dot	·	The smallest displayable dot (the *marker_scale* attribute is ignored) at the marker position.
2	Cross	+	A cross or plus sign with intersection at the marker position.
3	Asterisk	*	An asterisk with intersection at the marker position.
4	Circle	◯	A circle with center at the marker position.
5	X	×	A multiplication sign with intersection at the marker position.

A.1.2. ATextStyle

Table A-2 *ATextStyle*

Value	*Name*	*Description*
1	NotConnected	The annotation text primitive will be drawn with no line connecting it to the annotation text reference point.
2	Connected	The annotation text primitive will be connected to the annotation text reference point with a line, which will be drawn using the current set of line attributes.

A.1.3. InteriorStyle

Table A-3 *InteriorStyle*

Value	*Name*	*Description*
1	Hollow	The interiors of surface primitives are not filled, but the boundary is drawn using the surface colour. If the surface primitive is clipped as a result of modeling, view, or workstation clipping, the boundary must be drawn along the clipped boundary as well.
2	Solid	The interiors of surface primitives are filled using the surface colour.
5	Empty	The interior of the surface primitive is not drawn at all.

A.1.4. HatchStyle

No hatch styles are implemented.

A.1.5. LineType

Table A-4 *LineType*

Value	*Name*	*Description*
1	Solid	Draw the polyline or curve with a solid, unbroken line.
2	Dashed	Draw the polyline or curve with a line that is dashed.
3	Dotted	Draw the polyline or curve with a line that is dotted.
4	DashDot	Draw the polyline or curve with a line that contains alternating dots and dashes.

A.1.6. SurfaceEdgeType

Table A-5 *SurfaceEdgeType*

Value	*Name*	*Description*
1	Solid	Draw the surface edge with a solid, unbroken line.
2	Dashed	Draw the surface edge with a line that is dashed.
3	Dotted	Draw the surface edge with a line that is dotted.
4	DashDot	Draw the surface edge with a line that contains alternating dots and dashes.

A.1.7. PickDeviceType

Table A-6 *PickDeviceType*

Value	*Name*	*Description*
1	DC_HitBox	The picking aperture is defined by a pick position and a pick distance, both in device coordinates. The shape of the hit box (square, circle, etc.) is implementation-dependent. The pick distance defines the half-width or radius of the hit box.
2	NPC_HitVolume	The picking aperture is defined by two points that define a parallelepiped in NPC space. Any graphics intersecting with the volume will be selected.

A.1.8. PolylineInterpMethod

Table A-7 *PolylineInterpMethod*

Value	*Name*	*Description*
1	None	No interpolation will be performed between polyline vertices. If colour values are supplied that differ for the endpoints of a polyline segment, it is implementation-dependent whether the colour of the *i*th vertex will be used to draw the line between the *i*th and (*i+1*)th vertices, or whether they will be used to compute an average colour that will be used for the entire segment. If *n* is the number of vertices, the colour at the *n*th will be ignored. PEX-SI computes an average color.

A.1.9. CurveApproxMethod

Table A-8 *CurveApproxMethod*

Value	*Name*	*Description*
1	Implementation Dependent	This type is ConstantBetweenKnots (the same as value 2).
2	ConstantBetweenKnots	This technique tessellates the curve with equal parametric increments between successive pairs of knots. The tolerance value controls tessellation of the curve. If the tolerance value is not an integer value, it is truncated and only the integer portion will be used. If *tolerance* is less than or equal to zero, the curve will be evaluated only at the parameter limits, and at the knots that are within the specified parameter range. If *tolerance* is greater than zero, the curve will be evaluated at the parameter limits, at the knots that are within the specified parameter range, and at the number of positions specified by *tolerance* between each pair of knots.
3	WCS_ChordalSize	This technique tessellates the curve until the length of each line segment (chord) in world coordinates is less than the tolerance.

Table A-8 *CurveApproxMethod— Continued*

Value	*Name*	*Description*
4	NPC_ChordalSize	This technique tessellates the curve until the length of each line segment (chord) in normalized projection coordinates is less than the tolerance.
6	WCS_ChordalDev	This technique tessellates the curve until the maximum deviation (in world coordinates) between the line and the curve is less than the tolerance.
7	NPC_ChordalDev	This technique tessellates the curve until the maximum deviation (in normalized projection coordinates) between the line and the curve is less than the tolerance.

A.1.10. ReflectionModel

Table A-9 *ReflectionModel*

Value	*Name*	*Description*
1	NoShading	No light source shading computation is performed. The surface colour is not affected by light source illumination (effectively, shaded colour = intrinsic colour).
2	Ambient	Only the ambient terms of the lighting equation are used. The shaded colour will be the intrinsic colour as seen under ambient light.
3	Diffuse	Only the ambient and diffuse terms of the lighting equation are used. The shaded colour will be the intrinsic colour as seen under ambient light, plus a diffuse reflection component from each light source.
4	Specular	The ambient, diffuse, and specular terms of the lighting equation are all used during the light source shading computation. The shaded colour will be the same as for *Diffuse*, plus a specular reflection component from each light source.

A.1.11. SurfaceInterpMethod

Table A-10 *SurfaceInterpMethod*

Value	*Name*	*Description*
1	None	The colour resulting from a single light source computation is used for the entire surface. No interpolation will be performed across surface interiors or edges.

A.1.12. SurfaceApproxMethod

Table A-11 *SurfaceApproxMethod*

Value	*Name*	*Description*
1	Implementation Dependent	This type is ConstantBetweenKnots (the same as value 2)
2	ConstantBetweenKnots	This technique tessellates the surface with equal parametric increments between successive pairs of knots. The two tolerance values control tesselation in each of the two parameter dimensions. If the tolerance values are not integer values, they are truncated and only the integer portions of each will be used. If *u_tolerance* is less than or equal to zero, the surface will be evaluated only at the *u* parameter limits in the *u* direction, and at the *u* knots that are within the specified parameter range. If *u_tolerance* is greater than zero, the surface will be evaluated at the *u* parameter limits in the *u* direction, at the *u* knots that are within the specified parameter range, and at the number of positions specified by *u_tolerance* between each pair of knots. The value of *v_tolerance* is used similarly to control the evaluation in the *v* direction.
3	WCS_ChordalSize	This technique tessellates the surface until the length of each line segment (chord) in world coordinates in the *u* parameter direction is less than the specified *u_tolerance* value, and the length of every line segment in world coordinates in the *v* parameter direction is less than the specified *v* tolerance value.
4	NPC_ChordalSize	This technique tessellates the surface until the length of each line segment (chord) in normalized projection coordinates in the *u* parameter direction is less than the specified *u* tolerance value, and the length of every line segment in normalized projection coordinates in the *v* parameter direction is less than the specified *v* tolerance value.
6	WCS_PlanarDev	This technique tessellates the surface into facets. The technique subdivides the surface until the absolute value of the maximum deviation, in world coordinates, between any facet and the surface is less than *u_tolerance*.
7	NPC_PlanarDev	This technique tessellates the surface into facets. The technique subdivides the surface until the absolute value of the maximum deviation, in normalized projection coordinates, between any facet and the surface is less than *u_tolerance*.

A.1.13. TrimCurveApproxMethod

Table A-12 *TrimCurveApproxMethod*

Value	*Name*	*Description*
1	Implementation Dependent	This type is ConstantBetweenKnots (the same as value 2)
2	ConstantBetweenKnots	This technique tessellates the trim curve with equal parametric increments between successive pairs of knots. The tolerance value controls tessellation of the trim curve. If the tolerance value is not an integer value, it is truncated and only the integer portion will be used. If *tolerance* is less than or equal to zero, the trim curve will be evaluated only at the parameter limits, and at the knots that are within the specified parameter range. If *tolerance* is greater than zero, the trim curve will be evaluated at the parameter limits, at the knots that are within the specified parameter range, and at the number of positions specified by *tolerance* between each pair of knots.

A.1.14. ModelClipOperator

Table A-13 *ModelClipOperator*

Value	*Name*	*Description*
1	Replace	The specified halfspaces are used to create a new composite modeling clipping volume that replaces the current composite modeling clipping volume.
2	Intersection	The specified halfspaces are intersected with the current composite modeling clipping volume to compute a new composite modeling clipping volume.

A.1.15. LightType

Table A-14 *LightType*

Value	*Name*	*Description*
1	Ambient	A light source that affects all surface primitives uniformly. Ambient light sources have only a colour attribute.
2	WCS_Vector	A light source that is specified in world coordinates with a colour and a direction vector.
3	WCS_Point	A light source that is specified in world coordinates with a colour, a position, and two attenuation coefficients.
4	WCS_Spot	A light source that is specified in world coordinates with a colour, a position, a direction vector, a concentration exponent, two attenuation coefficients, and a spread angle.

A.1.16. ColourType

Table A-15 *ColourType*

Value	*Name*	*Description*
0	Indexed	A colour that is passed as an unsigned 16-bit integer (i.e., it is of type TABLE_INDEX). The integer value is used as an index into a colour lookup table. Dereferencing of an indexed colour value occurs at the time of rendering, at the time when the actual colour value is needed for rendering an output primitive.
1	RGBFloat	A colour that is passed as three floating-point values, in the order red [0-1], green [0-1], blue [0-1]. A colour in this format has a type defined by: COLOR_RGB_FLOAT: [r, g, b : FLOAT]

A.1.17. FloatFormat

Table A-16 *FloatFormat*

Value	*Name*	*Description*
1	IEEE_754_32	An IEEE 754 standard 32-bit floating-point value.
2	DEC_F_Floating	A DEC F-floating value.

A.1.18. HLHSRMode

Table A-17 *HLHSRMode*

Value	*Name*	*Description*
1	Off	All output primitives are drawn in the order they are processed. No attempt will be made to remove hidden surfaces.

A.1.19. PromptEchoType

Table A-18 *PromptEchoType*

Value	*Name*	*Description*
1	EchoPrimitive	Use an implementation-dependent technique that at least highlights the picked primitive for a short period of time.

A.1.20. DisplayUpdateMode

Table A-19 *DisplayUpdateMode*

Value	*Name*	*Description*
1	VisualizeEach	Visualize each change as it occurs (PHIGS ASAP).
2	VisualizeEasy	Visualize only the changes that are ‘‘easy to do’’ (PHIGS WAIT/UWOR). Things that are ‘‘easy to do’’ are those that have a dynamic modification of *IMM* or can be updated without a regeneration of the displayed image. If regeneration is ‘‘easy to do’’ (as defined by the PEX implementor), then a regeneration may occur.
3	VisualizeNone	Visualize none of the changes (PHIGS WAIT/NIVE). The changes are applied, but the image is not regenerated until there is an explicit request to do so.
4	SimulateSome	Visualize the easy changes and simulate those changes that can be simulated (PHIGS WAIT/UQUM). The PEX-SI server performs no simulations; therefore, this mode behaves the same as VisualizeEasy. The PEX-SI server implements this the same as VisualizeEasy.
5	VisualizeWhenever	All changes will eventually be visualized. If regenerations are necessary, they will be performed at the server’s convenience. One regeneration may cause a number of changes to be visualized. The client can issue an update workstation request to guarantee that all changes have been visualized (PHIGS ASTI/NIVE).

A.1.21. ColourApproxType

Table A-20 *ColourApproxType*

Value	*Name*	*Description*
1	ColourSpace	The rendering pipeline colour is converted into a colour with three individual colour components.
2	ColourRange	The rendering pipeline colour is converted into a single colour index.

A.1.22. ColourApproxModel

Table A-21 *ColourApproxModel*

Value	*Name*	*Description*
1	RGB	red, green, blue

A.1.23. GDP

No Generalized Drawing Primitives are implemented.

A.1.24. GDP3

No Generalized Drawing Primitives are implemented.

A.1.25. GSE

No Generalized Structure Elements are implemented.

A.1.26. RenderingColourModel

Table A-22 *RenderingColourModel*

Value	*Name*	*Description*
0	RGB	red, green, blue colour model
1	RGB	red, green, blue colour model

A.1.27. ParametricSurfaceCharacteristics

Table A-23 *ParametricSurfaceCharacteristics*

Value	*Name*	*Description*
1	None	No additional surface characteristics beyond the current surface attributes.
2	Implementation Dependent	An implementation-dependent method that displays the shape of the surface. This method does not distinguish between front and back facing portions of the surface. The appearance of the representation is controlled by the appropriate set of primitive attributes for the representation. It is implementation-dependent how the representation interacts with any interior rendering indicated by the interior attributes. The data record is ignored for this type.
3	IsoparametricCurves	Isoparametric curves are drawn on the surface. The data record contains the number of curves to draw in each of the parameter dimensions and their placement. If the placement is *Uniform*, the specified number of curves is evenly spaced between the parameter limits of the surface; curves are also drawn at the parameter limits. If the placement is *NonUniform*, the specified number of curves is evenly spaced between each pair of knots; curves are also drawn at the knots. In both cases, only the portions of isoparametric curves are drawn that are within the interior of the surface as defined by any trimming curves. This method does not distinguish between front and back facing portions of the surface. *Continued on next page*.

Table A-23 *ParametricSurfaceCharacteristics— Continued*

Value	*Name*	*Description*
3	IsoparametricCurves — *Continued*	The tessellation and appearance of the isoparametric curves are controlled by the surface approximation criteria and the polyline attributes, respectively. The isoparametric curves are drawn in addition to any interior rendering indicated by the interior style or back interior style attributes. Isoparametric curves have higher visual priority than the primitive's filled or hollow interiors, but lower priority than the primitive's edges.

A.2. Implementation-Dependent Constants

Table A-24 lists the values implemented for the Implementation Dependent Constants.

Table A-24 *Implementation-Dependent Constants*

Name	*Type*	*Value*
NominalLineWidth	CARD32	1
NumSupportedLineWidths	CARD32	~0L
MinLineWidth	CARD32	0
MaxLineWidth	CARD32	~0L
NominalEdgeWidth	CARD32	1
NumSupportedEdgeWidths	CARD32	~0L
MinEdgeWidth	CARD32	0
MaxEdgeWidth	CARD32	~0L
NominalMarkerSize	CARD32	1
NumSupportedMarkerSizes	CARD32	~0L
MinMarkerSize	CARD32	0
MaxMarkerSize	CARD32	~0L
ChromaticityRedU	FLOAT	0.628
ChromaticityRedV	FLOAT	0.346
LuminanceRed	FLOAT	1.0
ChromaticityGreenU	FLOAT	0.268
ChromaticityGreenV	FLOAT	0.588
LuminanceGreen	FLOAT	1.0
ChromaticityBlueU	FLOAT	0.150
ChromaticityBlueV	FLOAT	0.070
LuminanceBlue	FLOAT	1.0
ChromaticityWhiteU	FLOAT	0.313
ChromaticityWhiteV	FLOAT	0.329
LuminanceWhite	FLOAT	1.0
MaxNamesetNames	CARD32	256
MaxModelClipPlanes	CARD32	64
TransparencySupported	CARD32	FALSE
DitheringSupported	CARD32	FALSE
MaxNonAmbientLights	CARD32	64
MaxNURBOrder	CARD32	10
MaxTrimCurveOrder	CARD32	6

Table A-24 *Implementation-Dependent Constants— Continued*

Name	*Type*	*Value*
BestColourApproxValues	CARD32	PEXColourApproxAnyValues
DoubleBufferingSupported	BOOLEAN	FALSE

A.3. Renderer Dynamics

Table A-25 lists the binding of attributes to the renderer during rendering. Changes to statically bound attributes are not visible until the next rendering. Changes to dynamically bound attributes become visible immediately.

Table A-25 *Renderer Dynamics*

Attribute	*Binding Type*
MarkerBundle	dynamic
MarkerBundleContents	dynamic
TextBundle	dynamic
TextBundleContents	dynamic
LineBundle	dynamic
LineBundleContents	dynamic
InteriorBundle	dynamic
InteriorBundleContents	dynamic
EdgeBundle	dynamic
EdgeBundleContents	dynamic
ViewTable	dynamic
ViewTableContents	dynamic
ColourTable	dynamic
ColourTableContents	dynamic
DepthCueTable	dynamic
DepthCueTableContents	dynamic
LightTable	dynamic
LightTableContents	dynamic
ColourApproxTable	dynamic
ColourApproxTableContents	dynamic
PatternTable	dynamic
PatternTableContents	dynamic
TextFontTable	dynamic
TextFontTableContents	dynamic
HighlightNameset	dynamic
HighlightNamesetContents	dynamic
InvisibilityNameset	dynamic
InvisibilityNamesetContents	dynamic
HLHSRMode	dynamic
NPCSubvolume	dynamic
Viewport	dynamic
Cliplist	dynamic

A.4. PHIGS Workstation Dynamics

Table A-26 *PHIGS Workstation Dynamics*

Attribute	*Dynamic*
view_rep	IRG
marker_bundle	IRG
text_bundle	IRG
line_bundle	IRG
interior_bundle	IRG
edge_bundle	IRG
colour_table	IRG
pattern_table	IRG
wks_transform	IRG
highlight_filter	IRG
invisibility_filter	IRG
HLHSR_mode	IRG
structure_modify	IRG
post_structure	IRG
unpost_structure	IRG
delete_structure	IRG
reference_modify	IRG
buffer_modify	IRG

A.5. Lookup Tables

Table A-27 lists the maximum number of entries definable for each lookup table. Table A-28 lists the predefined entries for each lookup table. Each table except the Colour Lookup Table has one predefined entry. The index of that entry is given in the table.

Table A-27 *Lookup Tables*

Name	*Maximum Number of Entries*
LineBundle	20
MarkerBundle	20
TextBundle	20
InteriorBundle	20
EdgeBundle	20
Pattern	0
Colour	256
TextFont	2
View	6
Light	16
DepthCue	6
ColourApprox	6

Table A-28 *Predefined Values*

Attribute	***Index***	***Values***
LineBundle	1	
line_type		1
polyline_interp		1
curve_approx		{0,1.0}
line_width		1.0
line_color		{*Indexed*, 1}
MarkerBundle	1	
marker_type		3
marker_scale		1.0
marker_color		{*Indexed*, 1}
TextBundle	1	
text_font_index		1
text_precision		*String*
char_expansion		1.0
char_spacing		0.0
text_color		{*Indexed*, 1}
InteriorBundle	1	
interior_style		1
interior_style_index		1
surface_color		{*Indexed*, 1}
reflection_attr		{1.0, 1.0, 1.0, (*Indexed*, 1), 0.0, 0.0}
reflection_model		1
surface_interp		1
bf_interior_style		1
bf_interior_style1_index		1
bf_surface_color		{*Indexed*, 1}
bf_reflection_attr		{1.0, 1.0, 1.0, (*Indexed*, 1), 0.0, 0.0}
bf_reflection_model		1
bf_surface_interp		1
surface_approx		{0, 1.0, 1.0}
trim_curve_approx		{0, 1.0}
EdgeBundle	1	
surface_edges		*Off*
surface_edge_type		1
surface_edge_width		1.0
surface_edge_color		{*Indexed*, 1}
Pattern	None	Pattern Lookup Tables are not implemented.
Colour	See Table A-29.	

Table A-28 *Predefined Values— Continued*

Attribute	***Index***	***Values***
TextFont	1	
font†		Roman_M
View	0	
clip_flags		all *On*
clip_limits		(0,0,0),(1,1,1)
orientation		identity matrix
mapping		identity matrix
Light	1	
light_type		Ambient
direction		(0.0,0,0,0.0)
point		(0.0,0.0,0.0)
concentration		0.0
spread_angle		0.0
attenuation		(0.0,0.0)
color		{RGBFloat, (1.0,1.0,1.0)}
DepthCue	1	
mode		off
front_plane		1.0
back_plane		0.0
front_scaling		1.0
back_scaling		0.5
color		{Indexed}
Colour Approx	1	

† TextFont Lookup Tables entries are a list of font identifiers. PEX-SI allows a maximum of 16 font identifiers per entry. The predefined entry sets all 16 font ids to a server generated id that is loaded with the default font *Roman_M*.

Table A-29 *Predefined Colour Entries*

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
≥8	1.0	1.0	1.0	White

A.5.1. Fonts

Two font definitions are provided with PEX-SI. They are called *Roman* and *Roman_M*. *Roman_M* (monospaced *Roman*) is the default font. See the *PEX-SI Porting Guide* for information on how to define new fonts.

B

PEX Archive Files

The *PHIGS Functional Description, Part 2* defines a standard PHIGS archive file. This archive file provides ‘‘a file format suitable for the storage and retrieval of PHIGS structure and structure network definitions.’’ For complete concepts, descriptions, intents and limitations of the standard, see Part 2 of the *PHIGS Functional Description, ISO DP9592-2:1987(E)*.

B.1. File Format

The archive file format used by PEX-SI is used for the storage and subsequent retrieval of PHIGS and PHIGS PLUS structure elements. The file format conforms to the required archive file specification in PHIGS as to form and functionality of a generic set of archive element types. However, since the PEX archive file format is not a registered PHIGS format, it is classified as a private encoding. This encoding may be submitted for ANSI registration in the future.

The following guidelines were used in the design of the format:

- Define structure elements using the PEX protocol encoding.
- Conform to the PHIGS archive file specification.
- Allow all PEX binary representations to be supported.
- Allow efficient support of systems that provide random file access.

The PEX protocol already encodes PHIGS and PHIGS PLUS structure elements for use in a byte stream representation. The PEX archive format also uses this encoding for all structure elements.

B.2. Multiple Binary Representations

The PHIGS archive interface supports any of the binary representations that the PEX protocol can support. By reformatting the binary values, implementations can read and write files that are not written in the host computer’s native format. However, it is not intended that archives of different binary representations be allowed within one archive file.

B.3. Random Access

The archive file format fully allows random access. It also allows sequential access, if necessary.

B.4. File Definition

The PEX archive file format defines the following eight archive file elements:

- Structure Elements
- Begin Archive File Element
- Archive File Descriptor Element
- Begin Structure Element
- End Structure Element
- End Archive File Element
- Archive Free Space Element
- Archive File Index Element

A PEX archive file consists of an ordered set of elements from the above list. The ordering is described later in this chapter in the section *Conceptual States*. The elements are defined below.

B.5. Archive File Elements

B.5.1. Structure Elements

The PEX archive format encodes Structure Elements exactly as defined by the PEX protocol. These elements all begin with a two-byte opcode followed by a two-byte length. Their definition can be found in the *PEX Protocol Encoding Document*.

B.5.2. Begin Archive File Element

The Begin Archive File Element is encoded as shown in Table B-1.

Table B-1 *Begin Archive File Element*

Begin Archive File Element			
Type	***# Bytes***	***Description***	***Value***
CARD16	2	Archive Element Opcode	0x1010
CARD8	1	Length of Name (*n*)	0-255
CARD8	1	Pad	0x0
CARD8	*n*	Name of Archive	ASCII Characters
CARD8	pad(*n*)	Pads to multiple of 4 bytes	0x0

A PEX archive file must start with a Begin Archive File Element. The first two bytes of this element are the element opcode. This opcode value never has to be byte-swapped. PHIGS requires this element to have parameters for a name that can identify the archive. Since the binary format of the archive has not been determined when the Begin Archive File Element is encountered, the length of the name is encoded in one byte to eliminate the need for byte-swapping when the byte ordering has not been defined. It also limits the length of the archive file name to 255 characters. Zero length indicates that no archive file name is specified.

B.5.3. Archive File Descriptor Element

The Archive File Descriptor Element describes the functional capabilities required to process the archive file. A PEX archive file contains one Archive File Descriptor Element immediately following the Begin Archive File Element. The format of this element is shown in Table B-2.

Table B-2 *Archive File Descriptor Element*

Archive File Descriptor Element			
Type	***# Bytes***	***Description***	***Value***
CARD16	2	Archive File Descriptor Opcode	0x1111
CARD8	1	PEX Binary Format Byte	0x0,0x1,0x2,0x3
CARD8	1	Pad	0x0
INT32	4	Archive Version	Integer
CARD16	2	Text Length (*n*)	Integer
CARD8	2	Pad	0x0/0x0
CARD8	*n*	Character Description	ASCII Characters
CARD8	pad(*n*)	Pads to multiple of 4 bytes	0x0

The first two bytes are the opcode of the element type. This opcode does not have to be byte-swapped. The next byte describes the binary format of the archive file.

Bit 0 (the least significant bit) of the binary format byte indicates the byte ordering for *all subsequent numerical* values in the archive file. If this bit is zero, values are represented with the least significant byte first. If the bit is one, values in the archive file are represented with the most significant byte first.

Bit 1 of the binary format byte indicates the floating-point format for all floating-point values in the archive file. A value of zero for this bit indicates floating-point values are stored in IEEE-754-32 format. A value of one indicates floating-point values are stored in DEC F-Floating format. Bits 2-7 of this byte are zero.

Note

Until the binary format byte is read, no format conversion is performed. If the host computer's binary format differs from this specified format, all numerical values in the remainder of the archive must be converted.

The next four bytes of the Archive File Descriptor Element give an integer version number of the PHIGS archive file version (see PHIGS for a description of the appropriate values for the PHIGS version). The next four bytes of the Archive File Descriptor Element that give Archives that adhere to this document will have a version number of 1. The version number is encoded as an integer to match the PHIGS clear-text archive encoding.

The final part of the Archive File Descriptor Element is a counted list of character data that describes the archive file. This list contains a two-byte length field followed by the character data. The length may be zero, indicating that no character description exists.

B.5.4. Begin Structure Element

The format of the Begin Structure Element is shown in the Table B-3.

Table B-3 *Begin Structure Element*

Begin Structure Element			
Type	***# Bytes***	***Description***	***Value***
CARD16	2	Begin Structure Opcode	0x1212
CARD8	2	Pad	0x0/0x0
INT32	4	Structure Identifier	Integer
CARD32	4	Number of Elements	Integer
CARD32	4	Length of Structure Element Data	Integer

The opcode is followed by the structure identifier and number of structure elements. The last field specifies the number of bytes of Structure Element data that follows the Begin Structure Element. This length does not include the End Structure Element at the end of the structure definition. The specified number of Structure Elements (encoded using the PEX protocol) follow this element.

B.5.5. End Structure Element

The End Structure Element consists of only its opcode (0x1313) and a two-byte pad.

B.5.6. End Archive File Element

The End Archive File Element consists of only its opcode (0x1414) and a two-byte pad.

B.5.7. Archive Free Space Element

The format of an Archive Free Space Element is shown in Table B-4.

Table B-4 *Archive Free Space Element*

Archive Free Space Element			
Type	***# Bytes***	***Description***	***Value***
CARD16	2	Free Space Opcode	0x1515
CARD8	2	Pad	0x0/0x0
CARD32	4	Length (*n*)	Integer multiple of 4
CARD8	*n*	Unused	Undefined

The element consists of its opcode (0x1515) followed by a length field. The length specifies the number of bytes (following the length field itself) that remain in the Free Space Element. The rest of the block is undefined and contains information that is not examined.

B.5.8. Archive File Index Element

The Archive File Index Element maintains a list of contiguous blocks in the file. Each defined list entry of an Archive File Index Element describes the file position (the number of bytes from the beginning of the file) and size (number of bytes) of one block. A block consists of an Archive Free Space Element or a structure definition. For an Archive Free Space Element, the list entry specifies the position of the Archive Free Space Element. The size is the total number of bytes the element uses. In the case of a structure definition, the list entry gives the position of a Begin Structure Element. The specified size is the total number of bytes used by the Begin Structure Element and all subsequent Structure Elements up to and including the End Structure Element.

A PEX archive contains a singly-linked list of Archive File Index Elements. The first element of the list immediately follows the Archive File Descriptor Element. The list completely specifies the structures and Archive Free Space Elements that exist in the archive.

The contents of an Archive File Index Element are shown in Table B-5.

Table B-5 *Archive File Index Element*

Archive File Index Element				
Type	***# Bytes***	***Description***	***Value***	
CARD16	2	Archive File Index Opcode	0x1616	
CARD8	2	Pad	0x0/0x0	
CARD16	2	Number of entries used	Integer	
CARD16	2	Number of entries in list	Integer	
CARD32	4	Position of next Archive File Index Element	Integer	
CARD32	4	Length (Number of bytes remaining)	Integer	
CARD8	1	Block Type (Beginning of entry 1)	0x1=Struct, 0x2=Free	**Entry 1**
CARD8	3	Pad	0x0/0x0/0x0	
CARD32	4	Block Size	Integer	
CARD32	4	Block Position (Number of bytes from beginning of file)	Integer	
INT32	4	Structure Identifier	Undefined if Free Space	
CARD32	4	Number of Elements (End of entry 1)	Undefined if Free Space	
CARD8	1	Block Type (Beginning of entry 2)	0x1=Struct, 0x2=Free	**Entry 2**
CARD8	3	Pad	Undefined	
CARD32	4	Block Size	Integer	
CARD32	4	Block Position (Number of bytes from beginning of file)	Integer	
INT32	4	Structure Identifier	Undefined if Free Space	
CARD32	4	Number of Elements (End of entry 2)	Undefined if Free Space	
• • •				

The first two bytes of an Archive File Index Element specify the element opcode. The next field gives the number of blocks defined by this element. The following field gives the number of blocks that this element has space to define. (Some entries in the list may be unused.) The fourth field specifies the file offset in bytes from the beginning of the file to the next Archive File Index Element. This field is zero if there are no more Archive File Index Elements in the archive. Archive File Index Elements are maintained in a singly-linked list using this field. The fifth field is a length that specifies the number of bytes remaining in the element definition.

Each list entry in an Archive File Index Element consists of the following elements:

- A block type indicating whether this entry defines a Structure Element or an Archive Free Space Element.
- The size of the block.
- The position of the block.
- The structure identifier.
- The number of elements in the structure.

The last two values are undefined if the entry describes an Archive Free Space Element.

B.5.9. Conceptual States

A set of required relationships between Archive File Elements determines if an archive file is syntactically correct. The sequence of elements can be thought of as representing changes of state in a virtual system. The valid sequences of Archive File Elements can then be documented by means of a state transition table.

Table B-6 is a state transition table for the PEX archive file format. It lists Archive File Element/state combinations. Processing an element causes a transition from the starting state to the resulting state. This table is a more restrictive form of a similar table provided in Chapter 2 of the *Programmer's Hierarchical Interactive Graphics System* (PHIGS).

Table B-6 *Archive State Transition Table*

Archive File Elements and States		
Element	***Starting State***	***Resulting State***
Begin Archive File	Initial0	Initial1
Archive File Descriptor	Initial1	Initial2
Archive File Index	Initial2	Structure Closed
Archive File Index	Structure Closed	Structure Closed
Begin Structure	Structure Closed	Structure Open
Structure Element	Structure Open	Structure Open
End Structure	Structure Open	Structure Closed
Archive Free Space	Structure Closed	Structure Closed
End Archive File	Structure Closed	Final

Any other element/state combinations are illegal and can be considered to cause transition to an error state.