Lisa

Development System

INTERNALS

DOCUMENTATION

apple computer inc

Development Tools Group

FIRST DRAFT--February 1984

Contents

Lisa Development Software Documentation: A Road Map

Introduction

This road map was designed to help you to find your way around the various documents describing program development for the Lisa. It will help you decide which software you need to learn more about, which software you can ignore for the moment, and how you should proceed in studying the rest of the technical documentation.

General Overview of the Environments Available

There are as many ways of writing programs as there are creative programmers. However, Apple supports only three general styles of programs that you can write for the Lisa: those written for 1) the Workshop environment, 2) the QuickPort environment, and 3) the ToolKit environment. Programs written for any of these environments can use most of the same units and libraries, but there are some important differences of which you should be aware.

The *Workshop* (Figure 1) provides a simple non-window, character and graphic environment within which a program may run. Programs written to run in this environment may use Pascal's built-in I/O for both files and textual display to the console's terminal emulator, or they may directly utilize the Lisa OS's file system primitives. They may also use the QuickDraw unit for drawing bitmap graphics and displaying text in a variety of fonts with various attributes, and may utilize a variety of other useful library routines. These programs are not able to use the Lisa Desktop libraries dealing with windows, menus, and dialog boxes, nor do they have easy access to Lisa Office System documents.

In addition to providing these run-time facilities, the Workshop also includes a command shell which makes available to users an extensive set of facilities for: 1) interactive program development in Pascal, Assembly, BASIC, and COBOL; 2) file and device manipulation; and 3) interactive and batch program execution and control.

QuickPort (Figure 2) provides the simplest Desktop environment, at least from the programmer's viewpoint. In most respects, writing a program for the QuickPort environment is identical to writing one for the Workshop environment. Using Pascal's built-in I/O facilities, programs written for QuickPort may do textual display to a variety of window-based terminal emulators, and may also display graphics using QuickDraw. These programs do not directly use the Lisa Desktop libraries, and are, in fact, unaware of such things as the window environment, the mouse, and menus. The

may, however, exchange information with Lisa Office System documents via the Cut/Paste mechanism.

The *ToolKit* (Figure 3) provides the most complete access to the Desktop facilities. From the programmer's viewpoint, it also requires the most knowledge of these facilities. Programs written using the ToolKit use the Generic Application and may use any of the ToolKit building blocks, which provide easy, controlled access to the Lisa Desktop libraries, the mouse, and menus. They may also exchange information with Lisa Office System documents via the Cut/Paste mechanism.

Overview of the Pieces

QuickPort is a set of units that are USEd and linked with a program which is to be run in the Desktop environment. QuickPort then provides the program with a "terminal window", to which the program's console I/O may be directed through the use of Pascal's built-in Text I/O facilities. The program simply makes ReadLn and WriteLn calls to display text or receive keyboard input. QuickPort code hides from the program such issues as cutting and pasting information from other Desktop applications, communicating with the Desktop shell, growing and shrinking the window, covering and uncovering the window, and activating or deactivating the program. For a program using QuickPort, such issues are of no concern.

The ***ToolKit*** is a set of libraries that provides standard Lisa application behavior, including windows that can be moved, resized, and scrolled, pull-down menus with standard functions such as saving and printing, and the Cut/Paste mechanism. The ToolKit defines the parts of an application common to all Lisa applications. The object-oriented structure of the ToolKit allows you to implement your application as extensions to the "Generic Application".

The ***Lisa Operating System*** provides the program with an environment in which multiple processes can coexist, with the ability to communicate and share data. It provides a device-independent file system for I/O and information storage, and handles exceptions (software interrupts) and memory management for both code and data segments.

PASLIB is the Pascal run-time support library. Most of the routines in PASLIB support the Pascal built-in facilities, including routines for initialization, integer arithmetic, data and string manipulation, sets, range checking, the heap, and I/O.

Floating Point Libraries provide numeric routines which implement the proposed IEEE Floating Point Standard (Standard 754 for Binary Floating-Point Arithmetic), and higher-level mathematical algorithms. **FPLib** provides Single (32-bit), Double (64-bit), and Extended (80-bit) floating-point data types, a 64-bit integer data type, conversion from one arithmetic type to another (or to ASCII), arithmetic operations, transcendental functions, and tools for handling exceptions. **MathLib** provides, among others, algorithms such as extra elementary functions, sorting, extended conversion routines, financial analysis, zeros of functions, and linear algebra.

QuickDraw is a unit for doing bit-mapped graphics. It consists of procedures, functions, and data types you need to perform highly complex graphic operations very easily and very quickly. You can draw text characters in a number of fonts, with

variations that include boldface, italic, underlined, and outlined; you can draw arbitrary or predefined shapes, either hollow or filled; you can draw straight lines of any length and width; or you can draw any combination of these items, with a single procedure call.

The ***Desktop Libraries*** provide window, graphics, mouse, and menu routines used by all Office System applications. They are not directly called by any programs written for the three run-time environments discussed here, but provide the hidden foundation for both the QuickPort and the ToolKit environments.

The ***Hardware Interface*** unit lets you access Lisa hardware elements such as the mouse, the cursor, the display, the contrast control, the speaker, the keyboard, the micro- and millisecond timers, and the hardware clock/calendar.

The ***Standard Unit*** lets you do string, character, and file-name manipulation, prompting, retrieval of messages from disk files, abort exec file processing, and conversions between numbers and strings.

The ***IOPrimitives*** unit provides you with fast, efficient text-file input and output.

The ***Program Communication*** unit allows programs to communicate with each other and with the Workshop shell.

LisaBug allows you to examine and modify memory, set breakpoints, assemble and disassemble instructions, and perform other functions for run-time debugging.

More Detail

QuickPort: A program which is to make full use of the capabilities of the Lisa Office System will be structured as an endless loop, within which the program continually polls the Window Manager for any events it should respond to. We will refer to such a program as an *Integrated Program*. An integrated program must handle such asynchronous events as the program's window being activated or deactivated, the window being opened, closed, moved, resized, or needing update, the mouse button going down or up, and a key going down or up. The program must also be a good citizen in Lisa's multi-tasking but non-preemptive scheduling environment by volunteering periodically to yield the CPU to any other process needing service. These are just a few of the important characteristics of an integrated program. The result of a program following these and other guidelines will be that it exhibits the same consistent, responsive behavior as other Apple-written programs like LisaDraw.

QuickPort is a collection of pieces which make writing programs for the Office System's window environment as easy as writing them for the Workshop's non-window environment. NOTE: In order to differentiate the QuickPort modules from the program which uses them, we will refer to the program itself as a *Vanilla Program*. QuickPort allows the vanilla program to be more traditionally structured, as if its user interfacing were being done through a smart text/graphics terminal; the vanilla program presents its display to the user by a combination of text I/O calls (e.g., WriteLn/ReadLn) and QuickDraw calls (e.g., DrawString/PaintRect). The QuickPort modules handle all events from the Window Manager, provide for yielding the CPU to competing processes at specific points, and in general shelter the program from the

sometimes tricky requirements of writing an integrated program for the Lisa Office System.

QuickPort provides the vanilla program with a window, which may be divided into a *Text Panel* and a *QuickDraw Panel* for displaying both textual and graphic information. Each of these optional panels is configurable in size and location, and may be independently scrolled horizontally or vertically. Text and Graphics windows may be overlaid, so the resulting window presents a composite of both types of output. The window may be resized, moved, covered, or uncovered without the vanilla program even being aware of such events. Textual and graphic information may be exchanged between a vanilla program's document and other documents, whether vanilla or integrated, by using the familiar Cut/Paste mechanism. Without any effort on the part of the vanilla program, the end user is given a large measure of control over the window's configuration and behavior, using mouse and menu actions supported by QuickPort.

The user may request printing of either the text panel or the graphics panel. In addition, vanilla programs may produce printed output under program control by writing to the -PRINTER logical device. Whereas, in the Workshop environment, printing is immediate (each line printing as soon as the program "writes" it), in the QuickPort/Desktop environment printing is all spooled. This means that the printed output of a vanilla program will be submitted to the Office system's PrintShop, which determines from the print queue when the document will be printed.

The *Text Panel* emulates a terminal display which corresponds to the Pascal built-in OUTPUT file, the built-in INPUT file, and the -CONSOLE and -KEYBOARD logical devices. Apple provides emulators for the *VT100* and *SOROC* terminals, and makes it possible for you to either customize them or create entirely new terminal emulators. These terminal emulators are actually *filters* which pre-process the character output stream destined for the *Standard Terminal Unit*, which provides the Text Panel display. Each emulator's job is to recognize the terminal-specific character sequences imbedded in the output stream which are commands to the terminal, and to call upon the Standard Terminal Unit to take the appropriate actions. A program may eliminate the filtering step, if desired, by calling directly upon the Standard Terminal Unit for display actions.

The *Graphics Panel* allows your program to display graphics on a bitmap which is a maximum of 720 pixels wide by 364 pixels high--the same size as Lisa's physical screen bitmap. This panel can be resized by the user or under program control, and can be scrolled horizontally and vertically to display different parts of the entire bitmap. The Graphics Panel supports every QuickDraw call, including those related to setting foreground and background colors for printed output. An application may write anywhere in the coordinate plane of its graphics panel ('grafPort', to use QuickDraw's terminology), without having to worry about where its window is placed on the screen or what other windows are in front of it. QuickDraw, with a little help from the Window Manager, keeps the application's output from getting out of the graphics panel or from clobbering other windows.

The ToolKit: The ToolKit is a set of libraries that provides standard behavior that follows the design principles characterizing Lisa applications:

- Extensive use of graphics, including windows and the mouse pointer.
- Use of pull-down menus for commands.
- Few or no operating modes.
- Data transfer between documents by simple cut and paste operations.

For example, all Lisa applications have windows that can be moved around the screen, and that can usually be resized and scrolled. The ToolKit takes care of all these functions. The ToolKit also displays a menu bar for the active application, and provides a number of standard menu functions, such as saving, printing, and setting aside.

However, the ToolKit is more than a set of libraries. Because the ToolKit is written using Clascal, the ToolKit is almost a complete program by itself. You can, in fact, write a five-line main program, compile it, link it with the ToolKit, and run it. What results is the Generic Application.

The Generic Application has many of the standard Lisa application characteristics. A piece of Generic Application stationary can be torn off, and, when the new document is opened, it presents the user with a window with scroll bars, split controls, size control, and a title bar. The mouse pointer is handled correctly when it is over the window. The window can be moved, resized, and split into multiple panes. There is a menu bar with a few standard functions, so that the generic document can be saved, printed, and set aside. The single Generic Application process can manage any number of documents. You cannot, however, do anything within the window, aside from creating panes. The space within the window, along with the additional menu fuctions, is the responsibility of the real application.

Therefore, when you write a Lisa application using the ToolKit, you essentially write extensions to the Generic Application. It is very easy to write extensions to any Clascal program. To insert your application's functions, you create a set of subclasses, including methods to perform the work of you application, and then you write a simple main program, and compile and link it with the ToolKit.

Whenever necessary, the ToolKit calls your application's routines. For example, if the user scrolls the document, the ToolKit tells your program to redraw the changed portions of the window. Your program does not need to be concerned with when redrawing is required.

One effect of Clascal is that you can write applications in steps. You can begin by doing the least amount possible, and get an application that does very little, but will run. You can then extend your application bit by bit, checking as you go. This characteristic of Clascal makes it easy to extend the capabilities of ToolKit programs, even years after the original program.

The ToolKit's debugger, KitBug, provides run-time debugging of ToolKit Clascal programs. It allows you to do performance measurements, set breakpoints and traces, single-step through your program one statement at a time, and do high-level examinations of data objects

multiple processes can coexist, with the ability to communicate and share data. It provides a file system for I/O and information storage, and handles exceptions (software interrupts) and memory management.

The *File System* provides input and output. It accesses devices, volumes, and files. Each object, whether a printer, disk file, or any other type of object, is referenced by a pathname. Every I/O operation is performed as an uninterpreted byte stream. Using the File System, all I/O is device-independent. The File System also provides device-specific control operations.

A *process* consists of an executing program and the data associated with it. Several processes can exist at once, and will appear to run simultaneously because the processor is multiplexed among them. These processes can be broken into multiple segments which are automatically swapped into memory as needed. Communication between processes is accomplished through events and exceptions. An *event* is a message sent from one process to another, or from a process to itself, that is delivered to the receiving process only when the process asks for it. An *exception* is a special type of event that forces itself on the receiving process. In addition to a set of system-defined exceptions (errors), such as division by zero, you can use the system calls provided to define any other exceptions you want.

Memory management routines handle data segments and code segments. A *data segment* is a file that can be placed in memory and accessed directly. A *code segment* is a swapping unit that you can define. If a process uses more memory than the available RAM, the OS will swap code segments in and out of memory as they are needed.

PASLIB: PASLIB is the Pascal run-time support library. It provides the procedures and functions that are built into the Pascal language, acts as the run-time interface to the Operating System, and "completes" the 68000 instruction set by providing routines for the compiler-generated code to call upon in lieu of actual hardware instructions.

PASLIB routines are called with all parameters passed on the stack. There is an initialization routine to initialize necessary variables, libraries, and exception-handlers and set up global file buffer addresses, and a termination routine to kill processes. You can do four-byte integer arithmetic. Data can be moved, or scanned for a particular character. String manipulation routines include concatenating, copying, inserting or deleting a substring, determining the position of a substring, and comparing strings for equality. Set manipulation routines let you find set intersections or differences, adjust the size of a set, and compare sets for equality. There are range-checking and string range-checking routines. Heap routines let you allocate memory in the heap, mark or release the heap, check available memory in the heap, and check the heap result. I/O routines let you read and write lines, characters, strings, packed arrays of characters, booleans, and integers, as well as check for a keypress or an end-of-line, and send page marks. File I/O routines

Floating-Point Libraries: The Lisa provides arithmetic, elementary functions, and higher level mathematical algorithms in its intrinsic units FPLib and MathLib, which are contained in the file OSFPLIB.

FPLib provides the same functionality as the SANE and Elems units on the Apple][and ///, including:

- Arithmetic for all floating-point and Comp types.
- Conversions between numerical types.
- Conversions between numerical types, ASCII strings, and intermediate forms.
- Control of rounding modes and numerical exception handling.
- Common elementary functions.

MathLib provides the extra procedures available only on the Lisa:

- Extra environments procedures.
- Extra elementary functions.
- Miscellaneous utility procedures.
- Sorting.
- Free-format conversion to ASCII.
- Correctly rounded conversion between binary and decimal.
- Financial analysis.
- Zeros of functions.
- Linear algebra.

QuickDraw: Virtually all of Lisa's graphics are performed by the QuickDraw unit. You can draw text, lines, and shapes, and you can draw pictures combining these elements. Drawing can be done to many distinct "ports" on the screen, each of which is a complete drawing environment. You can "clip" drawing to arbitrary areas, so that you only draw where you want. You can draw to an off-screen buffer without disturbing the screen, then quickly move your drawing to the screen.

Text characters are available in a number of proportionally-spaced fonts. Any font can be drawn in any size--if a font isn't available in a particular size, QuickDraw will scale it to the specified size. You can draw characters in any combination of boldface, italic, underlined, outlined, or shadowed styles. Text can be condensed or extended, and it can be justified (aligned with both a left and a right margin).

Straight *lines* can be drawn in any length and width, and can be solid-colored (black, white, or shades of grey) or patterned.

Shapes defined by QuickDraw are rectangles, rectangles with rounded corners, full circles or ovals, wedge-shaped sections of circles or ovals, and polygons. In addition, you can describe any arbitrary shape you want. All shapes can be drawn either hollow (just an outline, which has all the width and pattern characteristics of other lines) or solid (filled in with a color or pattern that you define).

QuickDraw lets you combine any of these elements into a *picture*, which can then be drawn--to any scale--with a single procedure call.

Three-dimensional graphics capabilities are also available, in a unit called Graf3D, which is layered on top of the QuickDraw routines. Graf3D lets you draw three-dimensional objects in true perspective, using real variables and world coordinates.

The Hardware Interface: The Hardware Interface unit lets you access Lisa hardware elements such as the mouse, the cursor, the display, the speaker, the keyboard, and the timers and clocks.

Mouse routines determine the location of the mouse, set the frequency with which software knowledge of the mouse location is updated, change the relationship between physical mouse movement and the movement of the cursor on the screen, and keep track of how far the mouse has moved since boot time.

Cursor routines let you define different cursors, track mouse movements, and display a busy cursor when an operation takes a long time.

Screen-control routines can set the size of the screen, and set contrast and automatic fading levels.

Speaker routines allow you to find out and set the speaker volume, and create sounds.

Routines are provided to handle the different *keyboards* available for the Lisa, as well as the mouse button and plug, the diskette buttons and insertion switches, and the power switch. You can find out which keyboard is attached, and set the system to believe that a different physical keyboard is connected. You can check to see what keys (including the mouse button) are currently being held down, look at or return the events in the keyboard queue, and read and set the repeat rates for repeatable keys.

Date and time routines let you access the microsecond and millisecond timers and check or set the date and time.

The Standard Unit: The Standard Unit (StdUnit) is an intrinsic unit providing a number of standard, generally-useful functions. The functions are divided into areas of functionality: character and string manipulation, file name manipulation, prompting, retrieval of error messages from disk files, Workshop support, and conversions.

The unit provides types for standard strings and for sets of characters, definitions for a number of standard characters (such as <CR> and <BS>), and procedures for case conversion on characters and strings, trimming blanks, and appending strings and characters.

File name manipulation functions let you determine if a pathname is a volume or device name only, add file name extensions (such as ".TEXT"), split a pathname into its three basic components (the device or volume, the file name, and the extension) put the components back together into a file name, and modify a file name given optional defaults for missing volume, file, or extension components.

Prompting procedures let you get characters, strings, file names, integers, yes or no responses, and so forth from the console, providing for default values where appropriate.

Special Workshop functions let you stop the execution of an EXEC file in progress, find out the name of the boot and current process volumes, and open system files, looking at the prefix, boot, and current process volumes when trying to access a file.

Conversion routines let you convert between INTEGERs (or LONGINTs) and strings.

The IOPrimitives Unit: The IOPrimitives unit provides you with fast, efficient text-file input and output routines with the functionality of the Pascal I/O routines. It includes routines for reading characters or lines, and for writing characters, lines, strings, and integers, plus the low-level routines on which the others are based.

The Program Communications Unit: The Program Communications unit (ProgComm) provides three mechanisms for communication between one program and another or between a program and the shell. The first two involve strings sent from a program to the shell; one tells the shell which program to run next, the other is a "return string" that can be read by the exec file processor to tell an exec file, for example, whether the program completed successfully. The third mechanism involves reading from and writing to a 1K byte communications buffer, global to the Workshop. Using the unit, a program can invoke another program and provide its input through the buffer, without user intervention.

LisaBug: LisaBug provides commands for displaying and setting memory locations and registers, for assembling and disassembling instructions, for setting breakpoints and traces to trace program execution, for manipulating the memory management hardware, and for measuring execution times using timing functions. Utility commands are also available to clear the screen, print either the main screen or the LisaBug screen, change between decimal and hexadecimal, change the setting of the NMI key, and display the values of symbols.

Where to Go from Here

The Lisa development software is not fully documented yet. The following is a list of what is available, some of it only internally, as of this publication. Note that the spring-release manuals will be organized differently from the current versions, and will incorporate much of the information that is now in the internals documentation or in seperate documents.

Pascal Reference Manual for the Lisa
includes: QuickDraw
Hardware Interface
Floating-Point Library

Operating System Reference Manual for the Lisa

Workshop User's Guide for the Lisa

Lisa Development System Internals Documentation
includes: Pascal Run-Time Library
Standard Unit
LisaBug
Floating-Point Libraries

*QuickPort Applications User Guide**

*QuickPort Programmer's Guide**

An Introduction to Clascal

Clascal Self-Study

ToolKit Reference Manual

ToolKit Training Segments

Numerics Manual: A Guide to Using the Apple /// Pascal SANE and Elems Units
FPLib provides the same functionality as these units.

*MathLib Guide**

*These manuals currently in rough draft form.

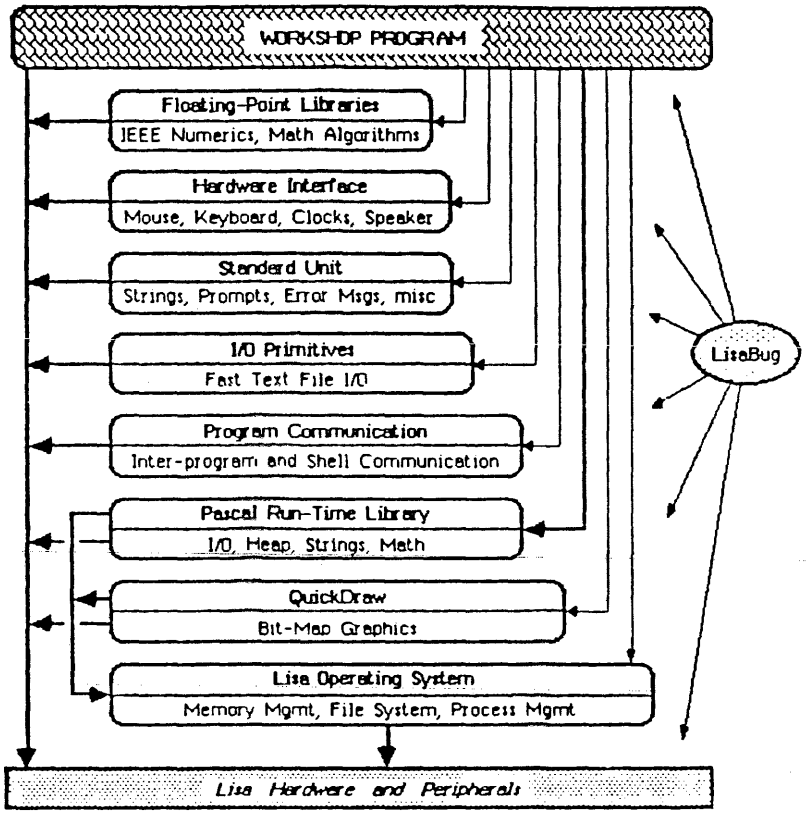
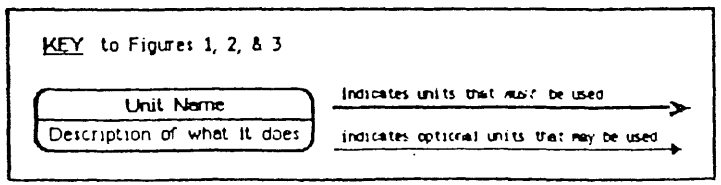


Figure 1
The Workshop Run-Time Environment



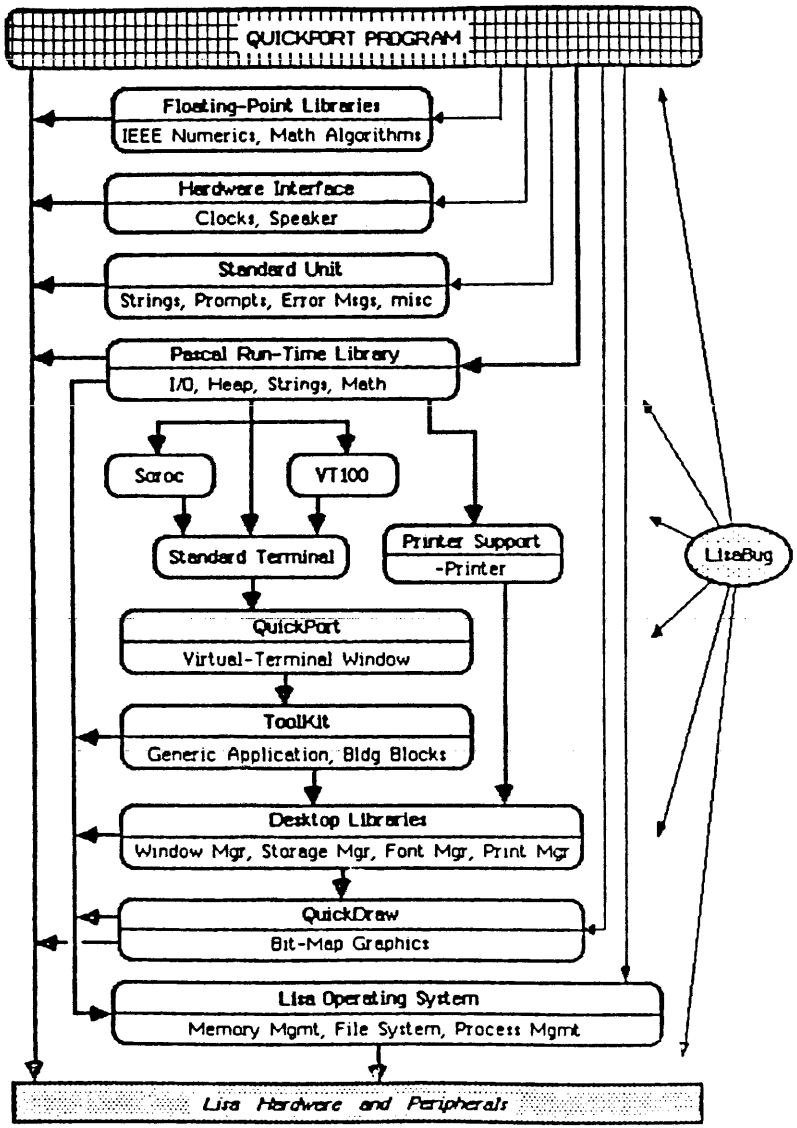


Figure 2
The QuickPort Run-Time Environment

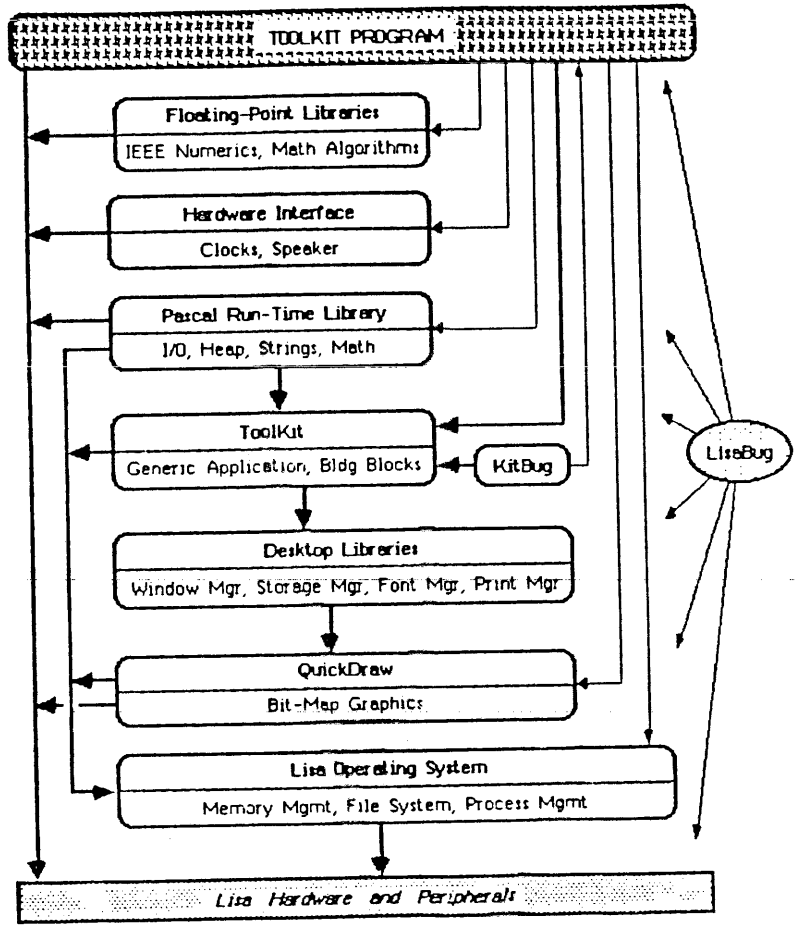


Figure 3
The ToolKit Run-Time Environment