ESV Workstation

# User's Manual

EVANS & SUTHERLAND COMPUTER CORPORATION
Salt Lake City, Utah

DOCUMENTATION WARRANTY:

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

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

ESV, ESV Series, ESV Series Workstations, ES/os, ES/Dnet, ES/PEX, ES/PHIGS, ES/PSX, Clean-Line, Fiber Link, Local Server, CDRS, and Shadowfax are trademarks of Evans & Sutherland Computer Corporation.

LAT Host Services, DEPICT, and PCONFIG are trademarks of Ki Research.

AVS is a trademark of Stardent Computer, Inc.

VAX, VMS, and DECnet are trademarks of Digital Equipment Corporation.

UNIX is a registered trademark of AT&T.

Ethernet is a registered trademark of Xerox Corporation.

Motif is a trademark of the Open Software Foundation, Inc.

SunPHIGS is a registered trademark of Sun Microsystems, Inc.

CrystalEyes is a trademark of StereoGraphics Corporation.

Spaceball is a trademark of Spatial Systems Pty Limited.

Kodak is a trademark of Eastman Kodak Company.

Part Number: 517940-102 AA
April, 1991

Copyright © 1991 by Evans & Sutherland Computer Corporation.
All rights reserved.

Printed in the United States of America.

# 1. Reader's Guide

# Table of Contents

**1. Reader's Guide ........ 1-1**

Typographical Conventions ........ 1-1

Software and Documentation Releases ........ 1-1

ESV Workstation Document Set ........ 1-2
- *ESV Workstation User's Manual* ........ 1-2
- *ESV Workstation Reference Manual* ........ 1-3
- *ESV Workstation Applications and Options* ........ 1-3

Software Options Documentation ........ 1-4
- Development Kit ........ 1-4
- RISCompilers ........ 1-5
- Documenter's Workbench ........ 1-5
- ES/PSX ........ 1-5
- AVS ........ 1-5

Optional Documentation ........ 1-6
- X Window System Documentation ........ 1-6
- OSF/Motif Documentation ........ 1-6
- Introductory Texts ........ 1-6

# 1. Reader's Guide

The ESV Workstation documentation is divided into the following categories.

- ESV Workstation Document Set
- Software Options Documentation
- Optional Documentation

Evans & Sutherland part numbers are shown in parentheses after the document title.

## Typographical Conventions

Following are the typographical conventions used in the *ESV Workstation Document Set* and the documentation for the ESV Workstation Options.

- *Times italics* indicate a new term that is being defined.
- TIMES SMALL CAPS represent the names of keys on the keyboard.
- **Helvetica bold** represents any command, function, option, data type, or filename.
- ***Helvetica bold italics*** represent a variable element which must be filled in with a specific value when actually used.
- `Courier normal` represents anything a user sees on the screen, the contents of a file, or sample code.
- **`Courier bold`** represents anything typed on a command line in response to a system prompt.

## Software and Documentation Releases

ESV Workstation software and documentation releases are numbered **X.Y.Z**, where,

- **X** = major functional changes and bug fixes with a general distribution to customers,
- **Y** = minor functional changes and/or bug fixes with a general distribution to customers, and
- **Z** = minor functional changes and/or bug fixes with a limited distribution to customers (generally through a customer request).

## ESV Workstation Document Set

The *ESV Workstation Document Set [2.0]* (217940-101) consists of the following three volumes and is shipped with all ESV Series Workstations.

- *ESV Workstation User's Manual*
- *ESV Workstation Reference Manual*
- *ESV Workstation Applications and Options*

### *ESV Workstation User's Manual*

This volume consists of the following chapters.

- "1. Reader's Guide"

  This chapter contains a description of, and part numbers for, all of the ESV Workstation documentation.
- "2. Product Overview"

  This chapter contains overviews of the ESV Workstation hardware and software.
- "3. Getting Started"

  This chapter contains information about booting the ESV Workstation, logging on, and the default system configuration.
- "4. Customizing the System"

  This chapter describes ways the user can change the ESV Workstation from its default configuration.
- "5. Video Output Guide"

  This chapter describes the ESV Workstation's video output. It contains video output specifications, timing formats, and describes techniques for recording screen images.
- "6. Editors"

  This chapter contains an overview of the **vi** and **emacs** editors.
- "7. Customer Support"

  This chapter contains site preparation, installation, and preventative maintenance information, and describes the warranty and hardware/software customer support plans.
- "8. Porting Guide"

  This chapter contains information for porting programs to the ES/os environment.
- "9. Local Server"

  This chapter contains information about the ESV Local Server which consists of a second CPU card and a shared memory module.

*ESV Workstation Reference Manual*

This volume consists of the following chapters.

- "1. ES/PEX"

  This chapter describes graphics standards, PHIGS and PHIGS PLUS functions, and contains sample PHIGS programs.

- "2. X Extensions"

  This chapter describes the X Input, X Picking, and E&S extensions to the X11 server. The E&S extension contains the X Overlay, X Multiscreen, X Video Timing Formats, and X Miscellaneous Traversal functionality.

- "3. X Clients"

  This chapter lists the X Clients supported by the ESV Workstation and contains the manual reference pages for the **X**, **Xesv**, **xcm**, **csm**, and **mwm** clients.

*ESV Workstation Applications and Options*

This volume consists of the following chapters.

- "1. Release Notes"

  This chapter is reserved for release note documentation which is distributed with software releases.

- "2. Helpful Hints"

  This chapter is reserved for "helpful hints" documentation which is distributed with software releases.

- "3. Application Notes"

  This chapter is reserved for "application notes" documentation which is distributed with software releases.

The remainder of this volume is reserved for the following documents which are provided when you purchase the ESV Workstation option(s).

*Stereo User's Manual [2.0]* (217941-101)

*Spaceball User's Manual [2.0]* (217941-201)

*Fiber Link User's Manual [2.0]* (217941-401)

*ES/Dnet User's Manual [2.0]* (217941-501)

*LAT Host Services User's Manual [2.0]* (217941-601)

*Diskless Node User's Manual [2.0]* (217941-701)

## Software Options Documentation

### Development Kit

The following optional documents are recommended for purchase with the Development Kit option. A minimum of one copy per site is recommended.

***SunPHIGS Document Set* (217982-100)**

This set consists of the following four volumes, available only as a set.

- *Getting Started with SunPHIGS*
- *SunPHIGS 1.1 Programming Guide*
- *SunPHIGS 1.1 Reference Manual*
- *SunPHIGS 1.1 Extensions Reference Manual*

***ES/PHIGS Reference Manual* (217960-100)**

***RISC/os Document Set* (400400-100)**

This set consists of the following seven documents, available as a set or individually.

- *RISC/os Programmer's Reference Manual* (400401-100)
- *RISC/os User's Reference Manual* (400402-100)
- *RISC/os System Administrator's Reference Manual* (400403-100)
- *RISC/os System Administrator's Guide* (400404-100)
- *RISC/os Programmer's Guide* (400405-100)
- *RISC/os User's Guide* (400406-100)
- *RISC/os Streams Primer and Programmer's Guide* (400407-100)

***MIPS Programmer's Document Set* (400410-100)**

The set consists of the following three documents, available as a set or individually.

- *RISCompiler Languages Programmer's Guide* (400411-100)
- *MIPS Assembly Language Programmer's Guide* (400412-100)
- *MIPS RISCompiler Porting Guide* (400413-100)

**RISCompilers**

The following documents are provided with the purchase the RISCompiler options.

- *MIPS-Pascal Programmer's Guide and Language Reference* (400421-100)
- *MIPS-FORTRAN Programmer's Guide and Language Reference* (400422-100)
- *MIPS-Ada 3.0 Programmer's Guide* (400423-102)

**Documenter's Workbench**

The following document is provided with the purchase of the Documenter's Workbench option.

- *MIPS RISC/os Documenter's Work Bench* (400424-100)

**ES/PSX**

The following document is provided with the purchase of the ES/PSX option.

- *ES/PSX Document Set* (217950-100)

**AVS**

The following document is provided with the purchase of the AVS option.

- *AVS Document Set* (217951-100)

## Optional Documentation

The following X Window System and OSF/Motif documentation is recommended for graphics program development.

### X Window System Documentation

The following books are published by O'Reilly & Associates and are available individually.

- *Xlib Programming Manual* (400450-100)
- *Xlib Reference Manual* (400451-100)
- *X Window System User's Guide, Motif Ed.* (400452-101)
- *X Toolkit Intrinsics Programming Manual, Motif Ed.* (400453-101)
- *X Toolkit Intrinsics Reference Manual* (400454-100)

### OSF/Motif Documentation

The following five books are published by Prentice-Hall, Inc., and are available individually.

- *OSF/Motif User's Guide, Revision. 1.0* (400460-100)
- *OSF/Motif Style Guide, Revision 1.1* (400461-101)
- *OSF/Motif Programmer's Guide, Revision 1.1* (400462-101)
- *OSF/Motif Programmer's Reference Manual, Revision 1.1* (400463-101)
- *AES User Environment Volume, Revision A* (400464-100)

### Introductory Texts

The following three introductory texts are available individually.

- *A Practical Guide to the UNIX System* by Mark G. Sobell, The Benjamin/CummingsPublishing Company, Inc. (400440-100)
- *Computer Graphics – Principles and Practice, Second Edition* by Foley, van Dam, Feiner, and Hughes, Addison-Wesley Publishing Company (400441-100)
- *The X Window System, Programming and Applications with Xt OSF/Motif Edition* by Douglas A. Young, Prentice-Hall, Inc. (400442-100)

# 2. Product Overview

# Table of Contents


**2. Product Overview ........ 2-1**

ESV Series Workstation ........ 2-1
Hardware Overview ........ 2-3
- Monitor ........ 2-3
- Hard Disks and Tape Drive ........ 2-4
- Interactive Devices ........ 2-4

Software Overview ........ 2-8
- Operating System ........ 2-8
- Software Development Environment ........ 2-8
- Graphics Environment ........ 2-10

ES/os Features ........ 2-11
- What the ES/os System Does ........ 2-11
- How ES/os Works ........ 2-12

Window Environment ........ 2-15
- X Toolkit ........ 2-15
- Motif ........ 2-16

# 2. Product Overview

The ESV Series Workstations are UNIX-based, high-performance, 3D graphics workstations, engineered to support applications in molecular modeling, industrial/automotive design, and design engineering and analysis. The ESV Series Workstations support the image quality and performance requirements of these applications.

The ESV Series Workstations are available in several models that can be tailored to meet individual customer's needs. Model numbers reflect various graphics performance configurations.

## ESV Series Workstations

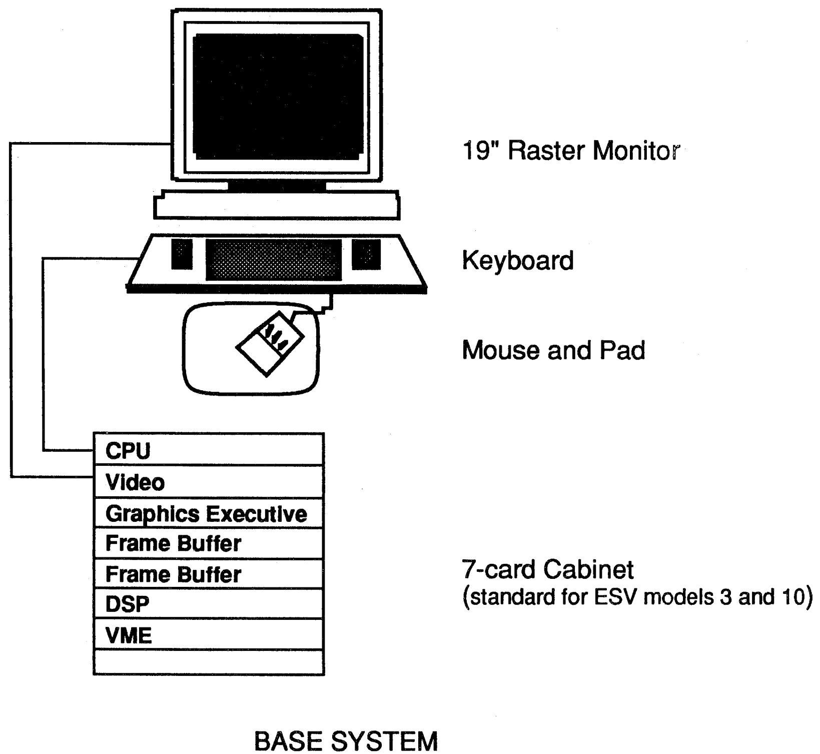
The ESV Series Workstations are available in several different configurations. Each configuration, defined by a model number, represents both a physical configuration and a level of performance outlined below:

| Model | 5-pixel vec/sec | 10-pixel vec/sec | 3-sided poly/sec | 4-sided poly |
|---|---|---|---|---|
| ESV 3/32 & 3/33 | 360,000 | 360,000 | 33,000 | 19,000 |
| ESV 10/32 & 10/33 | 525,000 | 525,000 | 50,000 | 28,000 |
| ESV 20/32 & 20/33 | 860,000 | 680,000 | 82,000 | 45,000 |
| ESV 30/32 & 30/33 | 1,060,000 | 680,000 | 115,000 | 62,000 |
| ESV 40/32 & 40/33 | 1,100,000 | 680,000 | 147,000 | 78,000 |
| ESV 50/32 & 50/33 | 1,100,000 | 680,000 | 172,000 | 100,000 |

| Model | CPU | MFLOPs | Model | CPU | MFLOPs |
|---|---|---|---|---|---|
| ESV 3/32 | 25MHz | 4 | ESV 3/33 | 33MHz | 5.28 |
| ESV 10/32 | 25MHz | 4 | ESV 10/33 | 33MHz | 5.28 |
| ESV 20/32 | 25MHz | 4 | ESV 20/33 | 33MHz | 5.28 |
| ESV 30/32 | 25MHz | 4 | ESV 30/33 | 33MHz | 5.28 |
| ESV 40/32 | 25MHz | 4 | ESV 40/33 | 33MHz | 5.28 |

| Dhrystone-MIPS/VAX-MIPS | Models |
|---|---|
| 24/20 | ESV 3/32, ESV 10/32, ESV20/32<br>ESV 30/32, ESV 40/32, ESV 50/32 |
| 32/26 | ESV 3/33, ESV 10/33, ESV 20/33<br>ESV 30/33, ESV 40/33, ESV 50/33 |

| DHRYSTONES (version 1.1) | Models |
|---|---|
| 41,000 | ESV 3/32, ESV 10/32, ESV20/32<br>ESV 30/32, ESV 40/32, ESV 50/32 |
| 52,000 | ESV 3/33, ESV 10/33, ESV 20/33<br>ESV 30/33, ESV 40/33, ESV 50/33 |



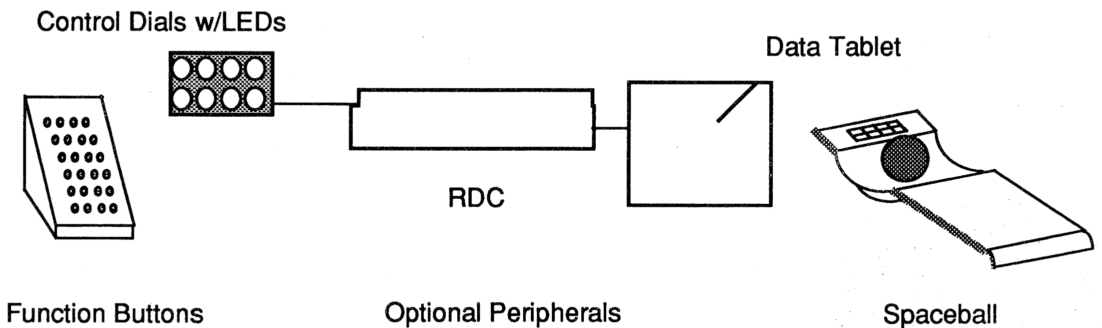




**Figure 2-1. Base system configuration and optional peripherals**

## Hardware Overview

The ESV Workstation's CPU is implemented with a 25 MHz MIPS R3000 RISC microprocessor or a 33 MHz MIPS R3000A RISC microprocessor. The CPU is central to the system, and it ties together the system's two global busses: the VMEbus and the Gbus (a proprietary bus used by the graphics subsystem). The CPU is capable of accessing devices on either the Gbus or the VMEbus. Both the graphics subsystem and the CPU are expandable for performance. ESV models 3 and 10 come standard in the small cabinet (7-slot). Other models come standard in the large cabinet which has 14 card slots including four VME and four graphics expansion slots. Figure 2-1 shows the base system configuration and optional peripherals.

The three major subsystems in the ESV Workstation (CPU, Graphics, and VME I/O) are all clock-independent. There is a total of 128 Kbyte of high-speed cache memory. System memory ranges from 8 Mbytes up to 128 Mbytes.

The color raster monitor has a viewing format of 1280 x 1024 pixels. The custom keyboard maintains communication via a synchronous serial link. The ESV Workstation mouse is a three-button optical mouse.

The standard hardware configuration is listed below:

- MIPS R3000 processor (25 MHz or 33 MHz),
- Ethernet controller with TCP/IP,
- Two additional RS-232 ports,
- One keyboard port,
- One mouse port,
- SCSI support for internal and external peripherals,
- 19-inch raster monitor,
- ESV graphics card set,
- Optical mouse,
- Alphanumeric keyboard,
- 8 or 16 Mbyte memory,
- VMEbus.

### Monitor

The ESV Workstation's monitor has no particular operating instructions other than turning it on and off, and adjusting the brightness and contrast of the screen.

The monitor should be turned on before you turn on power to the control unit. The monitor is turned on using the ON/OFF switch located on the lower

right-hand side of the monitor front. The ON/OFF switch is the bottom right of the two switches. The switch to the left of the ON/OFF switch is the degauss switch. The two thumbwheel knobs to the left of the switches are used to control contrast and brightness to reach a desired intensity. The right thumbwheel regulates the contrast; the left regulates brightness. Intensity is controlled by the use of both controls.

The main area of the screen is used for viewing text or graphics. The terminal emulator viewing area is 80 columns wide and 40 lines high. The screen space used for viewing depends on the application.

### Hard Disks and Tape Drive

Several sizes of hard disks and a tape drive are available as options on the ESV Workstation. Hard disk drives (5-1/4-inch) are available in sizes 180 Mbyte, 380 Mbyte, 760 Mbyte, and 1.2 Gbyte. A system may have up to two hard disks and one 1/4-inch 150 Mbyte tape drive.

A disk chassis, the "Data Shuttle," is available to facilitate the removal of two disk drives without disconnecting any wires or cables. The disk drives are mounted in individual canisters which slide easily in and out of the chassis.

The ESV Workstation has a SCSI interface permitting the use of CD-ROM storage devices which are a type of media for read-only mass storage. The availability of the CD-ROM driver allows ESV Workstation users to access purchased data bases.

An external SCSI connection permits the connection of up to seven internal and external peripheral devices.

### Interactive Devices

The following are interactive devices that can be configured with the ESV Workstation:

- Keyboard,
- Mouse,
- Reprogrammable data concentrator (RDC),
- Control dials with LEDs,
- Data tablet,
- Function buttons,
- Spaceball.

Only the keyboard and the mouse are standard peripherals. Others are optional.

## Input Protocol

Input from the keyboard and mouse is defined by the core X protocol. The mouse has two major functions: to generate motion information that application programs can detect, and to indicate the current location and focus of the keyboard.

Input from the other input devices is defined by an extension to the X11 server, which includes functions and events analogous to the core functions and events. This allows extension devices (dials, tablet buttons, *etc.*) to be individually distinguishable from each other and from the core devices (mouse and keyboard).

## Keyboard

The keyboard has standard alphanumeric keys, plus cursor-control (arrow) keys and a numeric keypad at the right. It also has a row of function keys across the top, including six defined function keys, twelve undefined function keys, and the escape key. The keyboard has three LEDs at its upper-right corner which indicate the status of the following keys:

- NUM LOCK
- CAPS LOCK
- SCROLL LOCK

The keyboard communicates with the workstation using synchronous serial protocol. It is activated when the system is booted. The keyboard connector cable should be plugged into the appropriate port on the control unit or the RDC if there is one.

The keyboard keys fall into seven categories:

- *Keyboard Function Control Keys.* These keys are the CTRL, SHIFT, CAPS LOCK, and ALT keys. They are local control keys that modify the signal generated by other keys when struck in combination with them.
- *Alphabetic Keys*, *Standard Numeric*, and *Special Character Keys.* These keys all generate standard ASCII character codes and are used to display uppercase and lowercase characters. The keys may be struck alone, or in combination with the keyboard function control keys.
- *Terminal Function Keys.* These keys are ESC, TAB, BACKSPACE, DELETE, ENTER, and the space bar. These keys produce codes used by a standard terminal.
- *Numeric Keypad Keys.* The function of these keys is determined by the X client.
- *Function Keys.* These keys are interactive keys that are user defined. They can be used for internal control or for communication to the application program.

### Mouse

The optical mouse consists of a three-button mouse unit with a reflective pad. The mouse transforms *x* and *y* axis position information to a digital form acceptable to the workstation. The cursor moves around on the screen in response to movement of the mouse across the pad.

The mouse uses red and infrared LEDs reflecting off the pad to provide directional information to the control logic in the mouse. This movement is translated into absolute *x* and *y* position information. This data is transmitted serially to the workstation.

### RDC

The RDC consists of a circuit card with six interactive device ports, one debug port, and one host computer port. It gives you two main advantages:

- It allows you to keep your display monitor (and RDC) a large distance from the control cabinet (for example, in another room) without having to run several long cables from each input device to the control unit. You can run cables from input devices to the RDC and a single cable from it to the control unit.
- It provides you with an additional RS-232 port.

The RDC receives input data from interactive devices connected to the interactive device ports and multiplexes the data to the host port. It also accepts data at the host port and demultiplexes it to the various interactive devices. It supports one keyboard at port A. Ports B through F support other RS-232 asynchronous devices.

### Control Dials

The control dials unit consists of eight dials with LED displays. The LEDs provide an 8-character label for each dial. The dials communicate dynamic, incrementing, and decrementing data to the workstation. In typical applications, the control dials can be used to perform the following operations:

- Rotate objects about the *x*, *y*, or *z* axis, each type of rotation typically using a different dial.
- Zoom in or out.
- Translate objects in *x*, *y*, and *z*, each translation typically using a different dial.

### Data Tablet

Four sizes of data tablets are available: 6 x 9 inches, 12 x 12 inches, 15 x 15 inches, and 18 x 25 inches. The data tablet consists of a tablet and a four-button cursor called a *puck* which sends position information to the workstation in digital form that expresses a 2D coordinate value (*x*,*y*).

### Function Buttons

The function buttons unit gives an expanded capability for program selection by providing 32 programmable function buttons in addition to the 12 function keys on the keyboard. The function buttons are lighted by incandescent bulbs. As with the function keys, pressing a function button results in a user-specified action.

### Spaceball

Spaceball is an interactive device consisting of a stationary ball mounted on a base. By pushing and twisting the ball, Spaceball senses the forces along, and the torques around, the *xyz* axes of its coordinate system. Eight programmable buttons, located on the upper face of the base, can be used as function keys; and another programmable button, located on the front of the ball, can be used as a pick button.

## Software Overview

The software on the ESV Workstation comes in several major sections. These include:

- Operating system,
- Software development environment,
- Graphics environment.

### Operating System

The ESV's operating system (ES/os) is UNIX System V with BSD extensions. The operating system source code is a modified version of MIPS RISC/os. Binary code compatibility for all non-graphical programs that execute on a MIPS R3000 system is maintained.

### Software Development Environment

The ESV's software development environment is the typical workstation environment. It provides the following:

- Standard UNIX shells

  The shells offered are **sh** and **csh**.

- Standard UNIX editors

  Available editors include **emacs, vi** and **ed**.

- Support for C and other programming languages

  Various programming languages are supported. C is the standard language distributed with the system. Additional compilers, including FORTRAN, Pascal and Ada, are offered as options. Other compilers may be supported as they become available.

- Source code debugger

  **dbx** is the source code debugger of choice. This is standard on a UNIX system.

- Utility programs

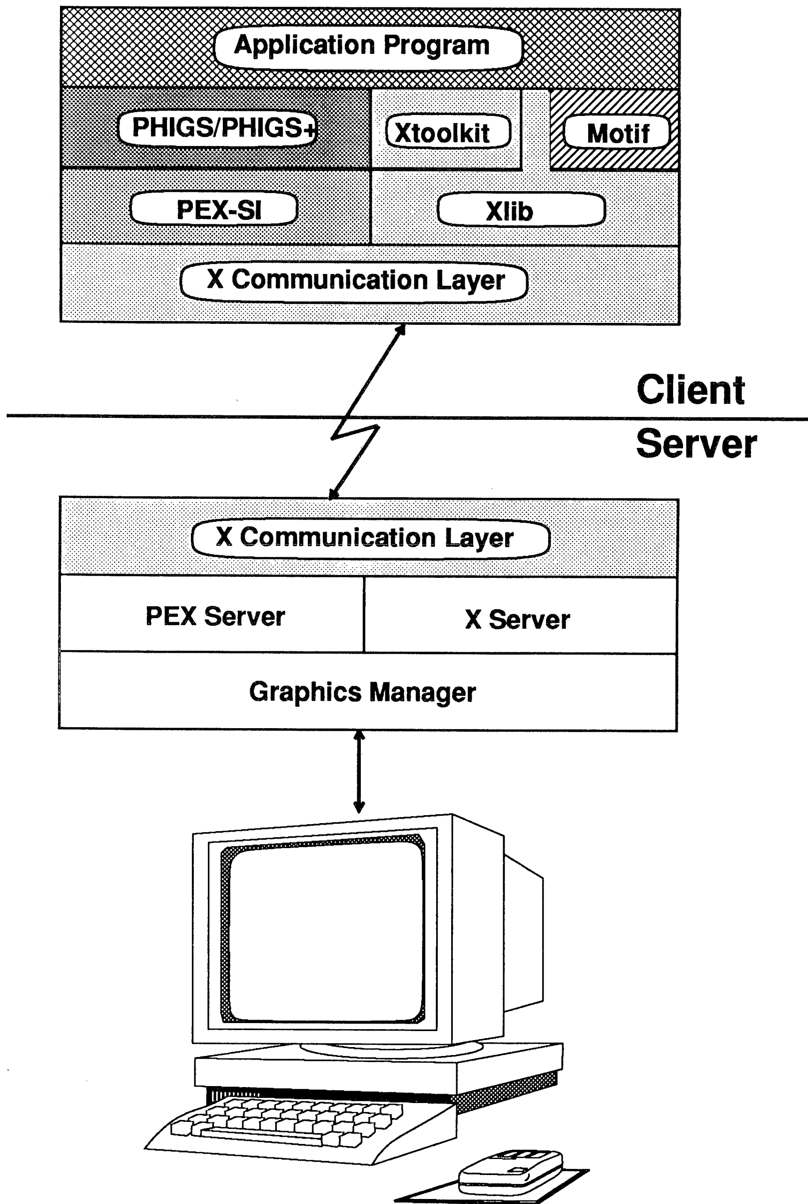



**Figure 2-2. Software overview**

## Graphics Environment

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

The X Window System supports 2D graphics. It consists of a *protocol* definition and an *implementation* of that protocol. The protocol is maintained by the X Consortium, which consists of members from different areas of industry and education, and the implementation is maintained by MIT. The current level of the X Window System is X11R4, and this is the version currently supported on the ESV Workstation.

The X Window System specification allows for extensions to be added. There are several extensions that come with the release from MIT, and PEX is one of these. PEX allows X to support 3D graphics and allows a user application access to PHIGS and PHIGS PLUS functions. A PEX Consortium defines its protocol. Its implementation is currently in alpha release to the X Consortium members.

An application program on the ESV can use the following graphical libraries:

- PHIGS
- PHIGS PLUS
- **Xlib**
- **Xt** (X toolkit)
- Motif (Open Software Foundation's X toolkit)

Figure 2-2 shows the software layers of the ESV Workstation.

### PEX Server

The PEX server is the device-dependent layer of the PHIGS extension to X. This layer builds and edits the PHIGS structures and maintains all the table information associated with a PHIGS workstation. The body of the data in the structures is the PEX protocol data, hence requires no translation.

### X Server

The X server is the device dependent layer of the X Window System. This layer takes advantage of the raster graphic capabilities of the pixel processors, and it converts basic X graphics operations into graphics commands handled by hardware. It also has the administrative task of coordinating the use of all graphics resources by the various application programs. These resources are the frame buffer, the color lookup tables, and the hardware cursor. The X server allocates windows and adjusts their visual priority on the screen.

## ES/os Features

### What the ES/os System Does

The ES/os operating system is a set of programs that controls the computer, acts as the link between you and the computer, and has various tools to help you do your work. The computing environment is uncomplicated, efficient, and flexible. It provides the following specific advantages:

- A general purpose system for performing a wide variety of jobs or applications;
- An interactive environment that allows you to communicate directly with the computer and receive immediate responses to your requests and messages;
- A multi-tasking environment that enables you to execute more than one program simultaneously;
- A multi-user environment that allows you to share the computer's resources with other users without sacrificing productivity. This technique is called timesharing. The ES/os system interacts with users on a rotating basis, but it appears to be interacting with all users simultaneously.

The organization of the ES/os system is based on four major components:

the *kernel* — The *kernel* is a program that constitutes the nucleus of the operating system; it coordinates the functioning of the computer's internals (such as allocating system resources). The kernel works invisibly; you need never be aware of it while doing your work.

the *file system* — The *file system* provides a method of handling data that makes it easy to store and access information.

the *shell* — The *shell* is a program that serves as the command interpreter. It acts as a liaison between you and the kernel, interpreting and executing your commands. Because it reads input from you and sends you messages, it is described as interactive.

*commands* — *Commands* are the names of programs that you request the computer to execute. The ES/os system provides tools for jobs such as creating and changing text, writing programs and developing software tools, and exchanging information with others via the computer.

## How ES/os Works

Figure 2-3 is a model of the ES/os system. Each circle represents one of the main components of the ES/os system: the kernel, the shell, and user programs or commands. The arrows suggest the shell's role as the medium through which you and the kernel communicate.

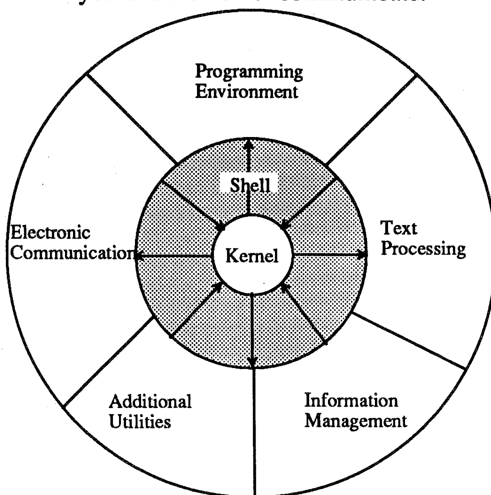



**Figure 2-3. Model of ES/os**

### The Kernel

The nucleus of ES/os is called the kernel. The kernel controls access to the computer, manages the computer's memory, maintains the file system, and allocates the computer's resources among users. Figure 2-4 is a functional view of the kernel.

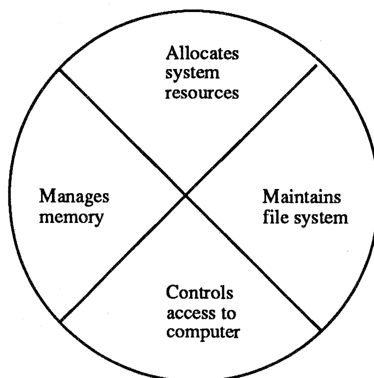
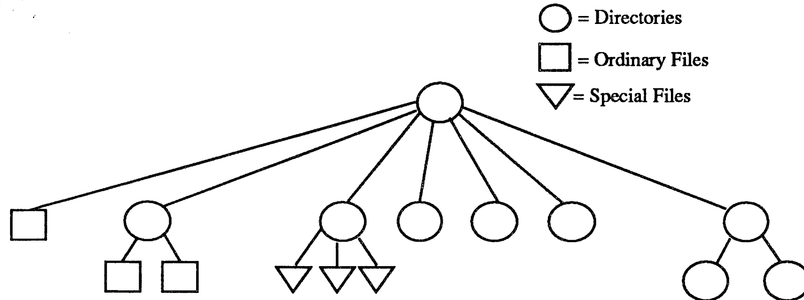



**Figure 2-4. Functional view of the kernel**

## The File System

The file system is the cornerstone of ES/os. It provides a logical method of organizing, retrieving, and managing information. The structure of the file system is hierarchical; if you could see it, it might look like an organization chart or an inverted tree.




The file is the basic unit of ES/os, and it can be any one of three types: an ordinary file, a directory or a special file.

An ordinary file is a collection of characters that is treated as a unit by the system. Ordinary files are used to store any information you want to save. They may contain text for letters or reports, code for the programs you write, or commands to run your programs. Once you have created a file, you can add material to it, delete material from it, or remove it entirely when it is no longer needed.

A directory is a super-file that contains a group of related files. You can create directories, add or remove files from them, or remove directories themselves at any time.

All the directories that you create and own will be located in your home directory. This is a directory assigned to you by the system when you receive a recognized login. You have control over this directory; no one else can read or write files in it without your explicit permission, and you determine its structure.

The ES/os system also maintains several directories for its own use. These directories, which include **/unix** (the kernel) and several important system directories, are located directly under the root directory in the file hierarchy. The root directory (designated by /) is the source of the ES/os file structure; all directories and files are arranged hierarchically under it.

Special files constitute the most unusual feature of the file system. A special file represents a physical device such a terminal, magnetic tape drive, or communication link. The system reads and writes to special files in the same way it does to ordinary files. However, the system's read and write requests do not activate the normal file access mechanism; instead, they activate the device handler associated with the file.

Some operating systems require you to define the type of file you have and to use it in a specified way. In those cases, you must consider how the files are stored since they might be sequential, random-access, or binary files. To the ES/os system, however, all files are alike. This makes ES/os file structure easy to use. For example, you do not need to specify memory requirements for your files since the system automatically does this for you. Or if you or a program you write needs to access a certain device, such as a printer, you specify the device just as you would another one of your files. In ES/os, there is only one interface for all input from you and output to you; this simplifies your interaction with the system.

Figure 2-5 shows an example of a typical file system. Notice that the root directory contains the kernel (**/unix**) and several important system directories.

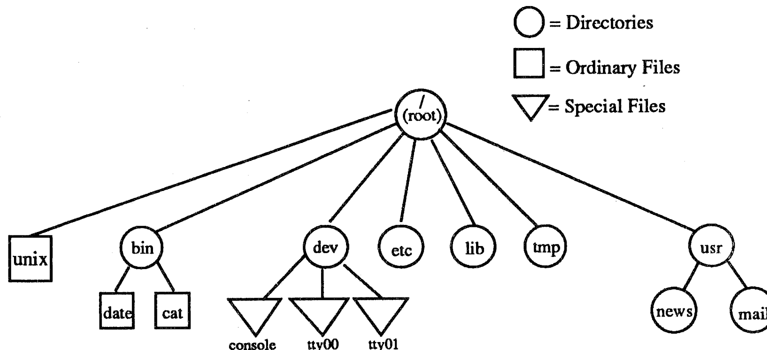



**Figure 2-5. Example of a file system**

**/bin** Contains many executable programs and utilities.

**/dev** Contains special files that represent peripheral devices such as the console, the line printer, user terminals, and disks.

**/etc** Contains programs and data files for system administration.

**/lib** Contains libraries for programs and languages.

**/tmp** Contains temporary files that can be created by any user.

**/usr** Contains other directories including **mail**, which contains files for storing electronic mail, and **news**, which contains files for storing electronic news articles.

The directories and files you create comprise the portion of the file system that is controlled by you. Other parts of the file system are provided and maintained by the operating system, such as **/bin, /dev, /etc, /lib, /tmp**, and **/usr**.

### The Shell

The shell is a unique command interpreter that allows you to communicate with the operating system. The shell reads the commands you enter and interprets them as requests to execute other programs, access files, or provide output. The shell is also a powerful programming language, not unlike the C programming language, that provides conditional execution and control flow features. The model of an ES/os system in figure 2-3 shows the two-way flow of communication between you and the computer via the shell.

### Commands

Programs that can be executed by the computer without need for translation are called *executable programs* or *commands*. As a typical user of ES/os, you have many standard programs and tools available to you.

## Window Environment

The ESV Workstation supports the X Window System. The X Window System is a network-based window system. Several versions of X have been developed; the most recent is X11.

The X11 architecture has two parts: display servers and clients. Display servers provide the network with display resources and control user input. Clients are application programs that control particular operations. Chapter 3 of this manual provides a longer description of the X Window System and how to use it on your ESV Workstation. The *ESV Reference Manual,* Chapter 4, describes the X clients which the ESV Workstation supports.

There are commercially available books on the X Window System. You may order a recommended set which is listed and described in chapter 1, "Reader's Guide," of this manual.

### X Toolkit

The X Toolkit is a programming library containing predefined components that ensure a consistent user interface in the creation of complicated applications.

## Motif

Motif is a user interface containing several development tools, one of which is a window manager. The window manager is based on the Motif toolkit which provides it with a standard graphical interface. The Motif window manager (**mwm**) provides functions for the following:

- Moving and resizing windows,
- Reducing windows to icons,
- Restoring windows from icons, and
- Arranging windows on the screen.

A description of the **mwm** client is provided in the *ESV Workstation Reference Manual,* Chapter 3. There are books by the Open Software Foundation about Motif and available by order from Evans & Sutherland. These are listed and described in chapter 1, "Reader's Guide," of this manual.

# Table of Contents


**3. Getting Started ....... 3-1**

Introduction ....... 3-1
Getting Started as an Administrator ....... 3-1
- Preparation for Booting ....... 3-1
- Debug Terminal ....... 3-3
- General Behavior ....... 3-3

Test Descriptions ....... 3-4
- Non-visible Confidence Tests ....... 3-5
- Visible Confidence Tests ....... 3-5

Booting UNIX ....... 3-7
Getting Started as a User ....... 3-9
- Logging on Using **xdm** ....... 3-9
- Logging on Without Using **xdm** ....... 3-10

UNIX Environment ....... 3-10
- C-shell (**csh**) ....... 3-11
- Bourne Shell ....... 3-12
- Directory **/usr/src/samples** ....... 3-12
- Path ....... 3-13

X Window System Overview ....... 3-15
- X Server ....... 3-15
- Window Manager ....... 3-15
- X Clients ....... 3-16

Running an X Session ....... 3-17
- Running an X Server with **xdm** ....... 3-17
- Running an X Server with **startesvx** ....... 3-17
- Default Clients Started at Login ....... 3-18
- Ending an X Session ....... 3-21

Customizing Resources and Default Clients ....... 3-21
- Resource Configuration ....... 3-23
- Setting Resources ....... 3-23
- Customizing Your **.Xdefaults** File ....... 3-24

Emergency Terminating ....... 3-25

# 3. Getting Started

## Introduction

The first part of this chapter is intended for the ESV manager. It describes the behavior of the ESV Workstation when it is not running the X Window System. The description covers system behavior from the moment the workstation is turned on, to the time at which a graphics application takes over the screen, then after the graphics application exits. The description is organized with the general behavior first – what you expect when the system is functioning normally. Next, each step of the start-up process is described. There is a description of the debug terminal and how it interfaces to the system.

The second part of this chapter is intended for the ESV user. It describes the ES/os (UNIX) environment and the X Window System including the window manager that runs on your ESV Workstation and other X application programs that are available to you.

## Getting Started as an Administrator

### Preparation for Booting

Booting the workstation is a simple procedure. Before starting the booting process, please check the following:

1) Make sure that the control unit and display are turned off.
2) Check that the control unit and display power cords are plugged into active wall outlets of adequate capacity.
3) The mouse should be connected to the control unit CPU I/O control panel port 1 or to the RDC. If the system includes an RDC, the RDC connects to the CPU I/O control panel on port 1. This port is also used for the debug terminal.
4) The keyboard and all other peripherals should be connected to the correct ports. Where the system expects each device to be located is part of the configuration information in the device file. All peripherals can be configured to work through the CPU's I/O port or through the RDC. The default configuration includes keyboard, monitor, and mouse, but does not include an RDC.

   Figure 3-1 shows the default configuration for plugging peripherals into a system without an RDC. Figure 3-2 shows which RDC ports the peripherals use.

   **Note:** In systems with RDCs, the mouse must connect to the port on the RDC; it cannot plug into the port on the cabinet.

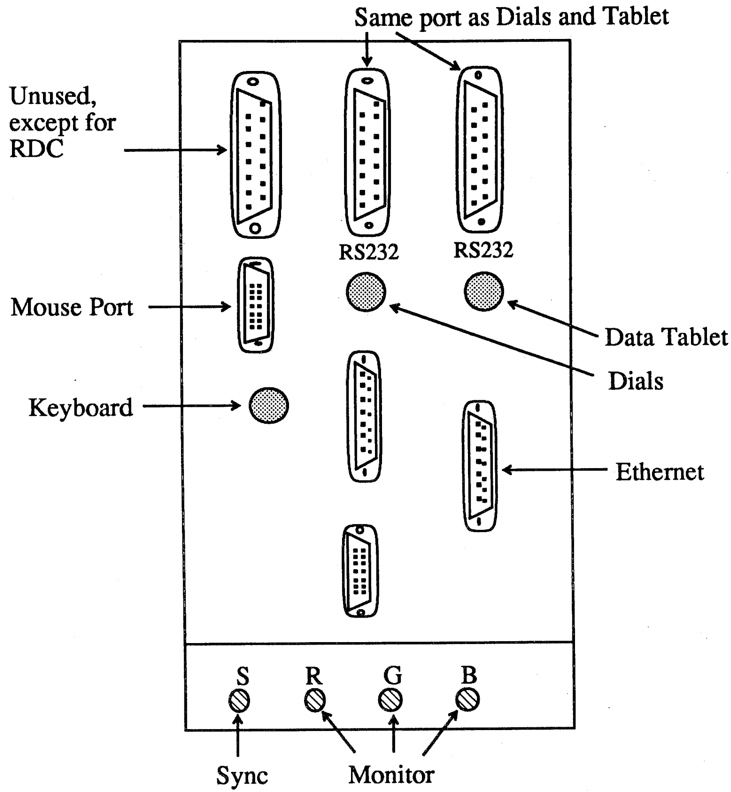



**Figure 3-1. Default cabinet port configuration (without RDC)**

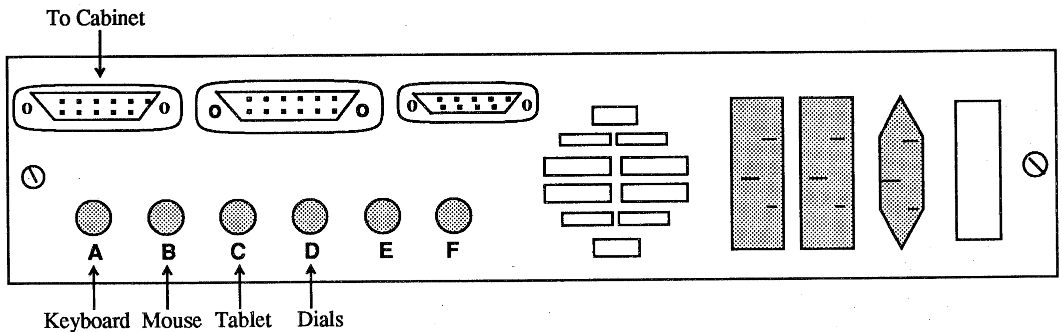



**Figure 3-2. RDC port configuration**

5) The display is connected to coax connectors marked RGB on the cabinet.

**Debug Terminal**

The debug terminal is an optional port to get diagnostic and bootup messages. The debug terminal should be handy to use if there is a problem that prevents the graphics subsystem from being used. Its terminal should be an ASCII terminal that plugs into the mouse/RDC port on the I/O panel. The PROM defaults to 1200 baud for the debug terminal.

Normally the system does not print anything to the debug port. It can be enabled by generating a "break" signal any time after power up as long as there are no graphics applications running. Also, the debug terminal is automatically enabled if any of the graphic subsystem tests fail.

Once enabled, everything that is printed on the graphics display is also sent to the debug terminal. The debug terminal can also be used for input as well as the system keyboard. To disable the debug terminal, you must change the environment variable ***console*** to either **l** or **s**. It should be **l** for a machine with graphics and an **s** for a machine without graphics (*i.e.*, server).

It is not possible to use the debug terminal while running a graphics application. The application needs the mouse or RDC to be hooked up instead of the debug terminal. The primary use of the debug terminal is for running diagnostics on the graphics subsystem.

**General Behavior**

The start-up process from the console consists of four steps, and the entire process is complete in approximately two minutes. A slight variance in the time is due to the number and size of the disks that the program must check. The steps in the process are:

1) Running non-visible confidence tests,

2) Running visible confidence tests,

3) Booting UNIX, and

4) Starting a graphics application.

Turn on the display, then the control unit. Turning on the workstation's power switch begins the non-visible confidence tests. These tests occur before the graphics subsystem is initialized to print messages. They perform quick checks for major problems in the graphics subsystem. If these confidence tests pass, the graphics subsystem is initialized to print out further messages, and visible confidence tests begin running automatically.

When visible confidence tests begin, a system banner is printed on the graphics display indicating that the power on confidence tests are being run. Each test prints out a title on the console terminal and a PASSED or FAILED

status upon completion. After all the tests are run, memory is cleared and the boot PROM prompt is displayed as shown:

```
>>
```

You can use the following command to see the value of all the PROM environment variables:

```
>>printenv
```

At the boot PROM prompt, you can start UNIX booting by typing **auto**. If the PROM environment variable **bootmode** is set to *c*, typing **auto** is unnecessary. In this case, the PROM automatically tries to boot UNIX. The failure of any confidence test, however, sets **bootmode** to *e*, and the environment variable requires reset to allow automatic booting again. The variable is reset by entering the following command:

```
>>setenv bootmode c
```

Then UNIX is loaded from disk and started. When the UNIX initialization is complete, a login prompt is displayed. At this point you can login and use the system as an ANSI terminal. You can also start a graphics application that controls the display. The terminal emulator remains dormant as long as the graphics application has control of the display. As soon as the application exits, the display is cleared and the terminal emulator starts.

The boot time is also the opportunity for the system administrator to load release tapes and change system configuration information. For detailed information on system administration, consult the following documents:

- *RISC/os System Administration Reference Manual*

  (Part No. 400403-100)
- *RISC/os System Administrator's Guide*

  (Part No. 400404-100)

If there are failures while this program is running, contact Evans & Sutherland.

## Test Descriptions

Each confidence test writes a specific LED pattern when the test is entered. If an error occurs, a specific LED failing pattern is flashed for about six seconds and a failing message is written to the console (if it's a visible test). The failing code is also written to the DIAG byte location in the NVRAM. The tests proceed regardless of a failure (except an early exception). If another error occurs, the LEDs flash but the NVRAM DIAG location is only written over if the new error is of a higher priority than the previous error.

## Non-visible Confidence Tests

The purpose of power-on testing is to provide you with reasonable confidence that your system functions correctly. An outline of the process is as follows:

1) When you turn the system on, the CPU begins executing at the start of PROM.
2) It initializes status and CPU registers.
3) It calls the memory configuration routine, **lmem_config()**.

   The routine **lmem_config()** performs the "soft" configuration of the memory boards. As it runs, it calls other routines and writes various LED patterns to indicate its progress. Successful completion of this routine indicates the following is true:

   - The CPU can execute instructions.
   - The write buffers are functional.
   - Some CPU I/O decode logic is functional.
   - The VME bus is communicating.

   The following is a list of the non-visible confidence tests that the system runs:

   - Enable Hardware
   - Test UART
   - Initialize UART
   - Register Test of Video Card
   - Initialize Video Card
   - Register Test of GSE Card
   - Initialize GSE Card
   - Initialize Pixel Processors
   - Initialize Frame Buffer
   - LED Test

## Visible Confidence Tests

This section contains information about the visible confidence tests which are the next diagnostic tests run on the system. These are still low-level tests. If an exception occurs at this level, the system is unusable. Should an exception occur, the LED value indicates which test was being run at the time of the failure.

Each test is followed by the PASSED or FAILED indication. The listing below shows what is displayed on the screen, beginning with the diagnostic banner, as these tests are run and passed.

```
Running Power-On Diagnostics. . .
Cache Test #1. . .PASSED
Cache Test #2. . .PASSED
Data Cache MATS+ Test. . .PASSED
Instruction Cache MATS+ Test. . .PASSED
Data Cache Block Refill Test. . .PASSED
Instruction Cache Block Refill Test. . .PASSED
Write Buffer Test. . .PASSED
Memory Test . . .PASSED
TBL Test. . .PASSED
Exception Test. . .PASSED
Instruction Streaming Test. . .PASSED
EDC Test. . .PASSED
Battery Check Test. . .PASSED
NVRAM Test. . .PASSED
Timer Test. . .PASSED
Time-of-Day Clock Test. . .PASSED
Duart Port Tests. . .PASSED
FP Test #1. . .PASSED
FP Test #2 . . .PASSED
DPR Test . . .PASSED                    (for LSC systems only)
Lance Slave Register Test. . .PASSED
Lance Master Test. . .PASSED
```

These are the names of the confidence tests.

- Print Power On Diagnostic Banner
- Cache Test #1
- Cache Test #2
- Data Cache MATS+ Test
- Instruction Cache MATS+ Test
- Data Cache Block Refill Test
- Instruction Cache Block Refill Test
- Write Buffer Test
- TLB Test

- Exception Test
- Streaming Instruction Test
- EDC Test
- Battery Check Test
- NVRAM Test
- Timer Test
- Time-of-day Clock Test
- Duart Port Tests
- FP Test #1
- FP Test #2
- Lance Slave Register Test
- Lance Master Test

## Booting UNIX

Booting is the process of reading the UNIX system kernel, usually stored in **/unix** on System V, into the system memory and starting it running.

When all the power on confidence tests have executed, the system clears memory, initializes the system keyboard, and displays the boot prompt. At the boot prompt (>>) either UNIX begins booting automatically, or you type **auto** to start UNIX booting. The listing below is a sample boot prompt.

```
Autoboot:  Waiting to load dkis(0,0,8)sash (CTRL-C
to abort, RETURN to expedite) loading

118400+21680+170416 entry: 0xa0300000

MIPS Standalone Shell Version 4.10 MIPS OPT Sat Jan
20 13:22:32 MST 1990
```

Optional action that you can take at the boot prompt includes loading and running programs from disk, tape or Ethernet. In the example above, the program, **sash**, is run. This is the program that boots UNIX. Usually, commands are issued at the boot prompt only when installing a new system.

All input comes from the system keyboard (or debug terminal) and is displayed on the graphics display. The display is treated like a simple ANSI terminal with 80 columns and 40 lines.

The system keyboard can be hooked up in two ways. It can either plug into the keyboard port on the I/O panel of the system, or plug into the RDC. When the system inititalizes the keyboard, it first checks on the I/O panel and, if the keyboard doesn't respond there, the system checks the RDC. If no key-

board is found, a message is printed to that effect and the boot PROM does nothing. Otherwise the keyboard will be initialized and used as the console input device.

As UNIX is booting, it prints messages, then prints a login prompt at the completion of booting.

The listing below is an example of the messages that UNIX prints on the screen as it boots. Several items in these messages vary from system to system, such as addresses, memory sizes, number of files, IDs, and system names.

```
Loading dkis(0,0,0)/UNIX
718576+85328+497648 entry: 0x80021000
CPU: MIPS R3000 Processor Chip Revision: 2.0
FPU: MIPS R3010 VLSI Floating Point Chip Revision: 2.0

ES/OS Release 1_0 ESV Version R_100
Total real memory = 25165824
Available memory = 23425024

root on dev 0x1000 (fstyp is ffs)
Available memory  = 22564864

Checking root file system () automatically.
The system is coming up.  Please wait.
*****Normally all file systems are fscked.
*****To fsck only dirty ones, type 'yes' within 5 seconds:
*****All file systems will be fscked.
mountall: fscking /dev/usr (/usr).
** /dev/usr
** Last Mounted on
** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference counts
** Phase 5 - Check Cyl groups
5632 files, 58523 used, 23249 free (201 frags, 2881 blocks,
0.2% fragmentation)
```

```
*****FILE SYSTEM WAS MODIFIED *****
/dev/usr/mounted on /usr
sadc:  Permission denied
Saving system core dump
Internet daemons:  routed portmap inetd rwhod.
NFS daemons:  nfsd biod.
systemname: /dev/dsk/isc0d1s4 mounted on /usr1
systemname: /dev/dsk/isc0d1s6 mounted on /usr2
The system is ready
```

## Getting Started as a User

### Logging On Using xdm

At this point the system automatically starts the application **xdm**, the display manager. **xdm** performs several tasks:

- Prompts you for a login/password,
- "Authenticates" you as the user,
- Runs a "session," which is simply defined as the duration of a given UNIX process.

You will see a window similar to that shown in figure 3-3.

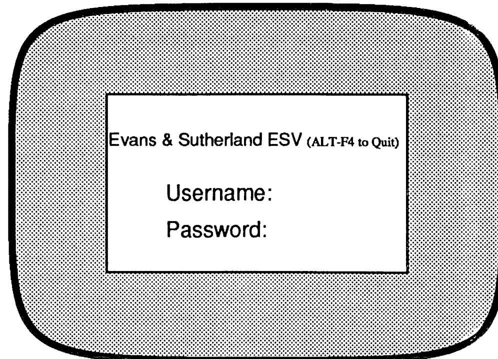



**Figure 3-3.** **xdm** Login Window

You enter your login name and password. After the login is authenticated one of two things happens:

1) If you have an **.xsession** file in your home directory, **xdm** starts the applications indicated in this file.

2) If you do not have an **.xsession** file, **xdm** starts an **xterm**, the **mwm** window manager, and the **xcm** client manager by default.

   **xterm** is a terminal emulator. You can run other clients from within the **xterm** window.

### Logging On Without Using xdm

If you don't want to use **xdm**, the key indicated on the **xdm** banner can be pressed to terminate **xdm**, and you can use the console.

You can use the machine as a simple terminal. The console behaves like an ANSI terminal and can be used with screen based editors.

After you log in, the system banner and prompt shown below is displayed.

```
ES/os 2.0 your_system's_name
Copyright 1991, Evans & Sutherland Corporation

**************************************************
*                                                *
*              Evans & Sutherland                *
*               ESV Workstation                  *
*                                                *
**************************************************

console>
```

You are now logged in to console mode. You can use UNIX commands from this prompt in this mode, but screen functionality is limited. For most purposes other than system management, you will want to start an X application to take advantage of the screen versatility that it provides.

When you start X or some other graphics application, the application takes over the graphics and peripherals. The terminal code remains disabled until the application exits. When the application exits, the screen is cleared and reinitialized. You can continue to use it as an ANSI terminal, or you can log out.

## UNIX Environment

Your environment on a UNIX system includes a set of operating system commands, system utilities such as editors, compilers, and mail, and a *shell*. The *shell* is a utility that acts as an interface between you and the operating system. It is an interactive command interpreter. The shell interprets what you type on the command line and translates it to execute the appropriate system command(s). Each user runs under a separate invocation of the shell. Requests to the shell can be a simple command, such as asking to view a file, or a complex request to compile a program.

The shell provides a number of convenient features for interactive use. For example, the shell can be made to keep track of the location of your home directory, your terminal type, your preferred text editor and preferred printer. It can remember the correct directory path to search for command programs, and your invented abbreviations for system commands. The command **env** will produce a listing of your current environment variables.

The shell can also process groups of commands in files called *shell scripts*. When the shell processes a script file, it executes all the commands in the file without requiring further input. The most commonly used shells are the C-shell (**csh**) and the Bourne shell (**sh**). For ease of operation and general friendliness, we encourage users, especially new UNIX users, to run the C-shell (**csh**) rather than the Bourne shell (**sh**) on their ESV Workstation.

## C-shell (csh)

The C-shell (**csh**) interface between you and UNIX is an interactive command interpreter and a high level programming language. As a programming language, its syntax is like that of the C programming language.

The primary purpose of **csh** is to translate command lines typed at a terminal into system actions, such as invocation of other programs.

An instance of **csh** begins by executing commands from the file **.cshrc** in your home directory. When **csh** is started by the login process, it also executes commands from the **.login** file. After login, the shell reads commands from the terminal or processes a shell script at your request.

Your system manager can set up your **/etc/passwd** file to execute C-shell for you by default on login. If this is not done, you can execute the C-shell by entering the command **csh** at the prompt.

### .cshrc

This is a file that should exist in your home directory. Whenever you start the C-shell, it performs commands from this file. There are many possible initial C-shell instructions that may be contained in this file to tell the C-shell how you want your commands and environment customized.

For example, you might use the command **setenv PRINTER lp2** to set your default printer to one named **lp2**. When you give the command **lpr**, your printing will go to printer **lp2** (if it exists).

You could use the command **set path= ($path /usr/newpath)** to add the directory **/usr/newpath** to the end of your existing command search path. (The system sets up a basic path for you at login.)

You also might use the command **alias lo logout** to make the shorthand command **lo** to stand for the real command **logout**. When you type **lo**, the C-

shell executes the **logout** command for you. **alias** can be very useful and is not available in the Bourne shell.

You may review C-shell options for inclusion in **.cshrc** by looking at the manual page for **csh** (type `man csh`).

### .login

This file is also used by the C-shell and should exist in your home directory. The **.login** file contains commands for the shell program. When the C-shell starts from a login, it performs the instructions in the **.cshrc** file, then looks for the **.login** file and performs the instructions contained there. This file is usually used for setting up special variables called environment variables. These variables are passed to other programs that you may start, such as another C-shell. They do not need to be set each time you run a new program.

You can set an environment variable in the **.cshrc** or **.login** file or from the command line using the **setenv** command. When you set variables from the command line, they are temporary and their effect disappears when you logout.

## Bourne Shell

Like the C-shell, the Bourne shell is both a command interpreter and a high-level programming language. When you use the Bourne shell as a programming language, it processes groups of commands stored in files called shell scripts. A shell script is a file that contains commands to be executed by the shell.

### .profile

In the Bourne shell, this file performs similar functions to the **.cshrc** and **.login** files in the C-shell. If you log in under the Bourne shell, you should have a **.profile** file in your home directory to set up your own environments.

The Bourne shell executes the commands in the **.profile** to customize the environment each time you log in. Each user has a different **.profile** file. Typically, it specifies a terminal type, establishes terminal characteristics and performs other housekeeping functions.

You may review Bourne-shell options for inclusion in **.profile** by looking at the manual page for **sh** (type `man sh`).

## Directory /usr/src/samples

The directory **/usr/src/samples** exists on your ESV Workstation and contains the following sample files among others:

- **.cshrc**
- **.login**
- **.profile**

The sample files should be copied into your home directory. From there, you can modify them as you wish. Your system manager may copy the files for you when your user account is created, or you can copy them yourself using the **cp** command as follows:

```
cp /usr/src/samples/.cshrc    yourhomedir/.cshrc   (C-shell only)
cp /usr/src/samples/.login    yourhomedir/.login   (C-shell only)
cp /usr/src/samples/.profile  yourhomedir/.profile (Bourne only)
```

The list below shows the contents of your home directory as it will be displayed with the **ls -al** command on your workstation after the sample files have been copied. Note that the script files (dot files) are hidden, and the **ls** command does not list them on the display them unless you use the **-a** option.

```
drwxrwxr-x  2  owner      Apr    9  13:15  .
drwxr-xr-x  4  bin        Apr    9  13:15  ..
-rwxr-xr-x  1  owner      Apr    9  13:15  .cshrc
-rwxr-xr-x  1  owner      Apr    9  13:15  .login
```

## Path

The *path* is a list of directories. It traces a sequential route through the file structure for the system to follow to locate a particular command or executable program. The first occurrence of a command found along the path is executed.

### BSD Extensions

The ESV Workstation is UNIX System V with BSD (Berkeley) extensions. Commands on the ESV Workstation default to System V commands, unless the directory specification **/bsd43/bin** is placed in the path. Placing **/bsd43/bin** properly in the path allows the operating system to execute the BSD extension commands.

Because some System V commands have the same name as BSD extensions, the position of **/bsd43/bin** in the path determines which one is executed.

### The C-shell and BSD Extensions

The default path is determined at login by the system. An example of the default path for the **csh** is set as follows:

```
set path = (~/bin  /usr/net  /bsd43/bin  /usr/ucb  /usr/bin  /
bin  /usr/new  /usr/bin/X11)
```

Generally, everything in this path except the home directory specification is necessary for the proper functioning of your account on the ESV Workstation. The command **env** shows what is in your current path.

Users of the C-shell will probably prefer to use the BSD extensions, because many of the extensions have more options than corresponding System

V commands. In this case, the directory **/bsd43/bin** should appear before **/usr/bin** in the path.

You can set the path from the command line. Note that setting the path on the command line is temporary: the effect of variables set on the command line disappears when you logout. Use the command **set path** to put in a new command directory at the front of the command line in the C-shell.

**set path = (*newcommanddir* $path)**

To put a new command directory at the end of the current path enter

**set path = ($path *newcommanddir*)**

**The Bourne Shell and BSD Extensions**

The default path for the Bourne shell is as follows:

```
PATH=$HOME/bin:/usr/net:/usr/bin:/bin:/usr/ucb:/usr/new::/
usr/bin/X11:/bsd43

export  PATH
```

Users of the Bourne shell may not want the operating system to default to the BSD extensions when duplicate command names exist. As long as the **/usr/bin** appears before **/bsd43/bin** in the path, the operating system will default to System V commands.

You can set the path from the command line. Note that setting the path on the command line is temporary: the effect of variables set on the command line disappears when you logout. Use the command **PATH** to put in a new command directory at the front of the command line in the Bourne shell.

**PATH=*newcommanddir*:$PATH;export PATH**

To put a new command directory at the end of the current path enter

**PATH=$PATH:*newcommanddir*;export PATH**

## X Window System Overview

The X Window System was developed at MIT and is now accepted as an industry standard. It allows you to open several windows at once and to run a different client program in each window. The operations performed in the different windows can vary considerably. For information on the X Window System, the following five-volume series published by O'Reilly & Associates is recommended:

- *Xlib Programming Manual* (Volume 1),
- *Xlib Reference Manual* (Volume 2),
- *X Window System User's Guide* (Volume 3),
- *X Toolkit Intrinsics Programming Manual* (Volume 4), and
- *X Toolkit Intrinsics Reference Manual* (Volume 5).

### X Server

The X server is a program that manages all user input to various client programs, and accepts output requests from clients. Input from client programs includes input from devices such as the keyboard, mouse, and control dials. As the server accepts output requests and distributes input, the X server updates the appropriate window on your display through a variety of communication channels. Every display is associated with a specific server. Clients may be run on the same host as the server, or they may be sent to a particular display from a remote host.

### Window Manager

The X window manager is a client program of the X Window System. Most interactive control of window movement, resizing, stacking order, etc. is concentrated in the window manager. The window manager distributed with the ESV Workstation is called the Motif Window Manager (**mwm**). It was developed and is copyrighted by the Open Software Foundation, Inc.

The **mwm** program allows you to interactively change many window characteristics and states. The following is a non-inclusive list of examples:

- Window size,
- Window stacking order,
- Placement on screen,
- Icon display,
- Input focus ownership, and
- Window frame appearance.

The **mwm** program also controls some client management functions, such as stopping a client.

## X Clients

An X client is an application program. Clients perform many kinds of tasks from information display, such as a clock showing the time of day on your screen, to terminal emulation and others.

The X clients supported on the ESV Workstation are listed below.

| | | |
|---|---|---|
| **appres** | **atobm** | **bdftosnf** |
| **bitmap** | **bmtoa** | **csm** |
| **editres** | **esvipc** | **ico** |
| **listres** | **maze** | **mkfontdir** |
| **muncher** | **mwm** | **oclock** |
| **pexscope** | **plaid** | **puzzle** |
| **resize** | **screen** | **sessreg** |
| **showrgb** | **showsnf** | **startesvx** |
| **startx** | **uil** | **xauth** |
| **xbiff** | **xcalc** | **xclipboard** |
| **xclock** | **xcm** | **xcutsel** |
| **xdm** | **xdmshell** | **xdpyinfo** |
| **xedit** | **xev** | **xeyes** |
| **xfd** | **xfontsel** | **xhost** |
| **xinit** | **xkill** | **xload** |
| **xlock** | **xlogo** | **xlsatoms** |
| **xlsclients** | **xlsfonts** | **xlswins** |
| **xmag** | **xman** | **xmh** |
| **xmodmap** | **xprop** | **xrdb** |
| **xrefresh** | **xset** | **xsetpointer** |
| **xsetroot** | **xstdcmap** | **xterm** |
| **xwd** | **xwdrle** | **xwininfo** |
| **xwud** | | |

Not all of the above clients are loaded onto the system by default. Some are optional and must be loaded by the system administrator. See the release notes for more details.

## Running an X Session

There are two ways to start the X server. The default is by using **xdm**. When **xdm** is running, the X server has been started. Then, you simply log in and out without having to start or stop the X server.

If you don't want to use **xdm**, you can exit from it by typing ALT F4. Then you can start the X server by typing the command **startesvx**.

Your X environment is determined by several files in your home directory. If these are not there, the system uses system default files. Your system manager may have copied the default files from **/usr/src/samples** into your home directory for you. You can determine if this has been done by using the **ls -a** command. You can copy them yourself using the **cp** command.

```
cp /usr/src/samples/.Xdefaults      yourhomedir/.Xdefaults
cp /usr/src/samples/.mwmrc          yourhomedir/.mwmrc
cp /usr/src/samples/.xsession       yourhomedir/.xsession
cp /usr/src/samples/.xinitrc        yourhomedir/.xinitrc
```

The **.xsession** file is used when **xdm** is running. The **.xinitrc** file is used when X is started via the **startesvx** command.

Once these files are in your home directory, you may modify them to suit your personal tastes.

### Running an X Server with xdm

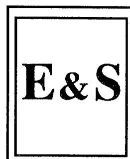
When you login under the **xdm** window, the X server has already been started. The **xdm** client looks in your home directory for the **.xsession** file to determine which clients to start.

### Running an X Server with startesvx

If **xdm** is not running, you will have a login prompt from the console. You can log in when in console mode. After you log in, enter the command

```
startesvx
```

This command starts the ESV X server and several X clients. While the server is starting, a flashing cursor is displayed on the screen. The cursor looks approximately like this:



When this cursor disappears, an hourglass is displayed at the center of your screen. The appearance of the hourglass signifies that the server is finished loading, and that now the default clients are being started.

## Default Clients Started at Login

Logging in via **xdm** or the **startesvx** command causes several default clients to start. As the default clients are loaded, the background color of the screen changes from gray to dark gray. The default clients are the following:

- **mwm** - the Motif window manager
- **xterm** - the terminal emulator
- **xcm** - the X client manger
- icon manager.

Figure 3-4 shows the default position of the windows.

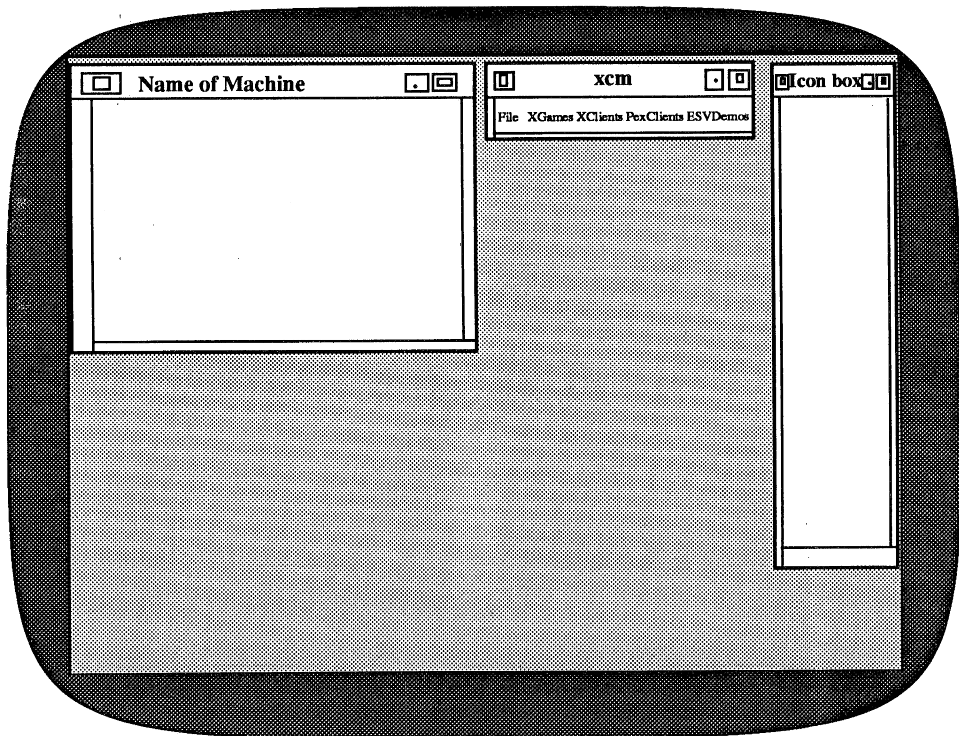



**Figure 3-4. Default window placement**

### mwm

The window manager allows interactive control of window size and placement on the screen. It controls the appearance of the window's title bar, whether the window is displayed normally or as an icon, how windows are stacked, where input focus lies, and other session management functions.

Most windows have a border around them, and a title and control bar at the top. These are normally light gray, but when the window is active (*i.e.*, has the input focus), they change to red. Also, when the window is active, the client name in the title area changes to yellow, and the block cursor changes from outline to solid yellow.

The window's title area displays the client's name. It is also used to move the window. To move a window, place the pointer in the title area, hold down the left button on the mouse (button 1), drag the window to the new location, then release the button.

You can resize a window by pressing button 1 when the pointer is over one of the resize handles on the window border. The cursor changes to an arrow indicating this. You can also expand the window into its maximum size by placing the cursor and clicking button 1 on the box at the far right of the title bar with the large square on it (the maximize button). Clicking on that box again reduces the window to its original size.

An icon is a small graphic representation of a window. You can turn a window into an icon by placing the cursor on the frame box with the dot in it toward the right side of the title bar (the iconify button) and clicking on button 1. There is an icon box window at the top of your screen. Each default window has its corresponding small flat-looking graphic representation in the icon box. You may not see all of these at once, because the icon box isn't large enough, but you can use the scroll bars at the sides of the icon box to scroll through its contents. Move the scroll bar by placing the cursor on the bar, and holding down button 1 while you drag the bar. When you turn a window into an icon, its graphic in the icon box changes to a more 3D appearance. Click twice on the icon to restore it to a window.

If you have copied the sample **.mwmrc** file from the sample directory into your home directory, **mwm** will display pull-down menus when you press one of the three mouse buttons.

Pressing button 1 when the cursor is in the root background window provides a menu with the following selections:

- Screen 0 (default) - Move to the default screen.
- Screen 1 (stereo) - Move to the stereo screen.
- Screen 2 (pseudo) - Move to the screen of visual class PseudoColor.

- Screen 3 (direct) - Move to the screen of visual class PseudoColor.
- Xterms

  Large Font Window - Create a new xterm window with a large font.

  Small Font Window - Create a new xterm window with a small font.
- Systems - Create a new **xterm** window and login to selected machines (by default local only).

For a description of the multiple screens, refer to the *ESV Workstation Reference Manual*, chapter 2, "X Extensions."

Pressing button 2 while the cursor is in one of the window borders produces a menu with the following selections:

- Refresh screen - Redraw the screen.
- Move window - Resize this window.
- Raise window - Raise this window to be in front of all other windows.
- Lower window - Lower this window to be behind all other windows.
- Destroy window - Destroy this window and its processes.

Pressing button 3 while the cursor is in the root background window produces a menu with the following selections:

- Restart - restart **mwm**.
- Kill **mwm** - kill **mwm**.

### xterm

This X client program is a terminal emulator for the X Window System. It is important because X itself supports only bitmapped displays. The windows created by **xterm** allow you to run applications designed for use in a standard alphanumeric terminal. You can bring up multiple terminal emulator (**xterm**) windows at the same time.

The default background color for **xterm** windows is blue. The **xterms** have a scroll bar at the left edge. When you place the cursor on this area, it takes on the shape of an arrow with pointers both up and down. If you hold down the middle mouse button and drag the double-pointed arrow up and down, it scrolls through text that has moved off the **xterm**. Release the middle button to stop the scrolling mode.

The **xterm** can be used to select and paste text. Position the mouse cursor at the start of the text you wish to select. Hold down mouse button 1 and drag the cursor over the text you want selected. Release button 1. The selected text remains highlighted. Move the mouse cursor to the spot where you want to

paste the selected text and click mouse button 2 to place it there. Click button 1 a second time to end the selection mode.

**xcm**

**xcm** is an X client program that allows you to find and run X clients through a menu system. By default, the program has several lists of clients that are selected via four pull-down menus, plus a menu to get help and to quit. These menu titles are displayed in your default **xcm** window and shown below. An example of the client names each pulldown menu contains are shown below:

| Files | XGames | XClients | PexClients | ESVDemos |
|---|---|---|---|---|
| **Help** | **maze** | **xclock** | **pexclock** | **pmv** |
| **Quit** | **xtetris** | **xload** | **vrt** | **courses** |
| | **xeyes** | **xcalc** | | **esvcad** |
| | **ico** | **xbiff** | | |
| | **puzzle** | **xedit** | | |
| | **xchomp** | **xman** | | |
| | **xmines** | **xmag** | | |
| | **xlogo** | **csm** | | |
| | **xterm** | | | |
| | **xlock** | | | |

The menu system is configurable by making changes to your **.xcmrc** file in your home directory. If this file does not exist, you can access the file **/usr/lib/X11/system.xcmrc** for menu configuration information. For further information see the **xcm** man page.

### Ending an X Session

To end the session, move the focus to the *exit* window and type **exit**. This window by default is the **xterm** in the upper left of the screen. This takes you back to the **xdm** login window (if you're in **xdm**), or back to the console window from which you may log off the system.

## Customizing Resources and Default Clients

When X is started by the **startesvx** command, the ESV X server starts a set of default clients. The default set of clients is determined by a file called **.xinitrc**. If there is no **.xinitrc** file in your home directory, the **startesvx** command loads the clients specified by the file **/usr/lib/X11/system.xinitrc**. However, when you have an **.xinitrc** file in your home directory, the **startesvx** command loads the clients specified in your file. This allows you to determine your own set of default clients.

Displays managed by **xdm** use the same mechanism but the files are **.xsession** and **/usr/lib/X11/system.xsession.**

The following is taken from the **/usr/src/samples/.xinitrc**:

```
# Initialization file to bring up a representative set of
# useful clients,in the absence of a $HOME/.xinitrc startup
# file. If the user has a $HOME/.Xdefaults file, it will take
# precedence over any specifications in /usr/lib/X11/app-
# defaults files.
#
# If the user gets here, s/he does not have a .xinitrc file
# therefore we start up some useful clients and windows.
#

# Start up a window manager - Motif
mwm -multiscreen &
sleep 2
xcm &
# Set Mouse Parameters
xset m 2 4

# Set Root Window Color
xsetroot -solid Grey5
# xsetroot -solid \#093f26 color used prior to release 2.0

# X Applications (commented out)
#xbiff  -geometry 80x80-105+5 -bw 0 &
#xclock -geometry 80x80-5+5 &
#xcalc  -geometry 186x222+878+5 &
#xman   -geometry -5+105 &

# XTERMS
# Note that the last xterm created is done in the foreground
# and is therefore the controlling window. Exiting this window
# will close down the entire display.

# Commented out xterms used prior to release 2.0
#xterm -rw -T `hostname | /bsd43/bin/tr a-z A-Z` -fn 9x15 -
geometry 80x40+525+325 &
#xterm -rw -T `hostname | /bsd43/bin/tr a-z A-Z` -geometry
80x34+2-2 &

# only default xterm used in this file. Also an EXIT window
xterm -rw -T "`hostname | /bsd43/bin/tr a-z A-Z` Exit Window"
-geometry 80x32+2+2
```

**Note:** With the exception of the last **xterm**, all clients are run in the background (see the ampersand **&** at the end of the lines). The last **xterm** is the exit window and must be run in the foreground.

The placement and appearance of the clients loaded by **startesvx** or **xdm** is controlled by the *resources* specified for them. When each client starts up, it takes on the resource characteristics specified in the **.Xdefaults** file in your home directory (and any additional resources specified on the start-up line of the **.xinitrc** or **.xsession** file). A resource specified on the start-up line will override the specification in the **.Xdefaults** file.

### Resource Configuration

Resources are variables that determine the appearance of various client features such as color, geometry, size, font, and so on. Each client running on the workstation may have resources associated with it that allow you to customize the way the client looks on your screen. To change a client's appearance, you change the *value* associated with the resource.

Each client has different resources available to it. Many of these are set up as defaults in the **.Xdefaults** file. You can change the client's default behavior by modifying the resource statements in the **.Xdefaults** file in your home directory. The following are the resource definitions for the default **xterm**.

```
xterm*jumpScroll          on
xterm*Scrollbar           on
xterm*saveLines           400
xterm*font                6x13
xterm*TitleBar            on
xterm_DeiconifyWarp:      on
xterm*Background:         #00294d
xterm*Foreground:         #ffffff
xterm*border:             #ff1493
xterm*cursorColor:        #ffff00
xterm*pointerColor:       #ff0000
```

Note that the additional resource `-fn 9x15` specified on the start-up line for the first **xterm** in the **.xinitrc** file overrides the `xterm*font 6x13` in this file.

### Setting Resources

A resource is a variable. It has a value attached to it. By changing the value, you change the behavior or appearance of an X client. Values can be Boolean

```
xterm*jumpScroll: on
```

or they can be numbers or strings

```
xterm*saveLines: 400
```
or
```
xclock*Mode: analog
```

Resources can be associated with an application's objects, *e.g.*, window, scrollbars, buttons. The resource can be associated with a single *instance* of the object, or with the *class* of objects. For example:

```
xcalc*Buttons:  gray
xcalc*Buttons.enter:  yellow
```

The first resource specification makes all buttons (class **Buttons**) in the client **xcalc** gray, and the second resource specification makes a single button (the enter button) yellow.

**Syntax**

The most basic resource specification consists of a line with the name of a client followed by a period or asterisk, the name of an object, followed by a colon and white space, and a value assigned to this resource. For example,

```
xcalc*enter:  gray
```

In this example, **xcalc** is the client; the asterisk is a wildcard representing any omitted classes (like **Buttons**); **enter** is the name of the object; **gray** is the value associated with the resource.

By convention, instance names begin with a small letter and class names begin with a capital letter. Any name (instance or class) that is made of more than one word (*e.g.*, **simpleMenu** or **SimpleMenu**) is written with all succeeding words capitalized and concatenated without a space.

Because resource names are hierarchical (they become more specific moving from left to right), they require separators to mark hierarchical divisions. A period is the separator for a fully specified name; an asterisk is a wildcard separator that stands for intervening classes or objects not written.

Note that trailing spaces at the end of a line can have a disastrous effect in resource specifications, and note that capitalization is important. Because of the different capitalization in the two lines below, the first works properly, but the second does nothing:

```
xterm*font:  10x20
xterm*Font:  10x20
```

## Customizing Your .Xdefaults File

Your **.Xdefaults** file must be located in your home directory. It can contain resources and resource values associated with any of the default clients that run on the ESV Workstation. If you do not have one (use the **ls -a** command to check) copy the example from **/usr/src/samples** to your home directory. You can customize the default appearance of your clients by modifying this

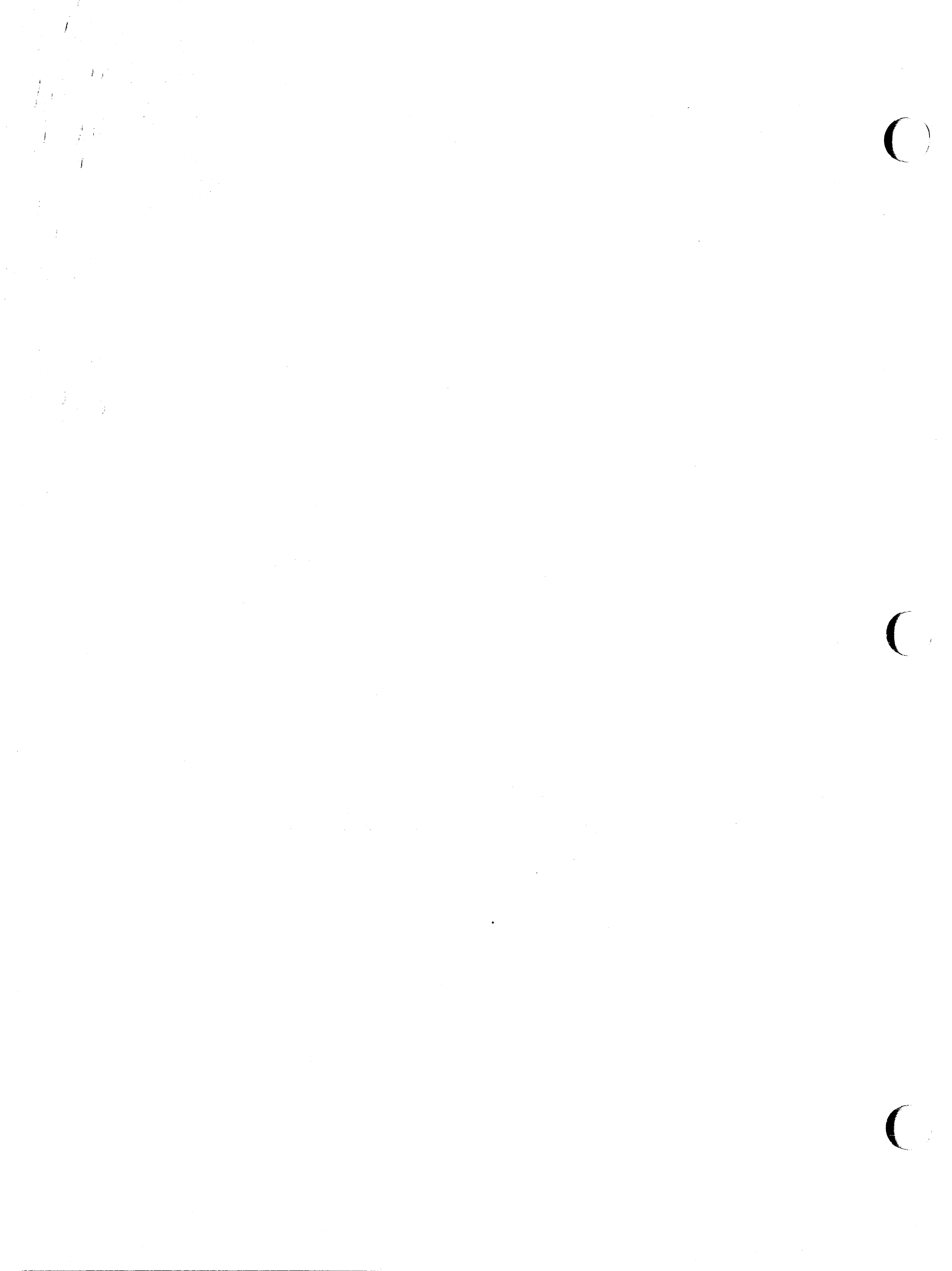
file. The **/usr/src/samples/.Xdefaults** file is meant to provide an easy method for customizing your own clients. Many alternative configuration resources are listed in this file, but most are commented out by a pound sign (**#**) at the beginning of the line.

You can easily change the default configuration by editing the **.Xdefaults** file to uncomment some resource value and comment out what is currently active. Note that changes to this file may not take effect until you restart **mwm**. To become familiar with the possibilities for changing the resource configuration, you should look at the **.Xdefaults** file and consult the reference pages for the particular client. The manual pages supply you with further information on each client's resources.

## Emergency Terminating

The key sequence discussed here should be used for emergency exits only. It does not provide a neat or clean exit. If the X server was started from **startesvx**, the key sequence will kill the X server. If **xdm** is running, the key sequence will kill all the user's clients and bring the system back to the **xdm** login window.

Press the left CONTROL key and the left ALT key simultaneously, then press the PAUSE/BREAK key (top row, right-most key).



# Table of Contents


**4. Customizing the System .......... 4-1**

Introduction .......... 4-1

Managing System Parameters .......... 4-1

Modifying Lookup Tables .......... 4-1

Configuring Graphics Memory .......... 4-5

Example **gm_config.dat** File .......... 4-7

# 4. Customizing the System

## Introduction

This section contains examples of ways that you can customize your workstation's system parameters.

For information on setting up your workstation with system administration tasks, consult the *RISC/os System Administration Reference Manual*. System administration tasks include, but are not limited to, the following:

- Controlling system security,
- Setting up logins and passwords,
- Setting user and group IDs,
- Managing disk device(s),
- Administering the file system,
- Backing up the file system.

## Managing System Parameters

When you have successfully logged on to the workstation, you may want to modify the terminal environment or other system parameters. This section contains examples of ways in which you may want to manage your system. The first example is how to modify the workstation's lookup tables. The second example is about configuring the system's shared global memory.

### Modifying Lookup Tables

The file **pex_config.dat** is a data file you may use to configure the workstation's lookup tables at server start-up. It is opened as the file name specified by your environment variable **PEX_CONFIG**, or if this variable is not set, the file is opened from the directory **/usr/lib/X11**. You may configure the lookup tables to any size, except for the light table which is constrained internally to twelve entries, and the view table which is constrained to 32 entries. Only the number of entries in the lookup tables may be changed by the user. Changing the other entries has unpredictable results. In order for changes to have an effect on lookup table size, the server must be restarted after making any modifications to **pex_config.dat**.

The format of **pex_config.dat** is rigid and may not be changed without a resulting crash of the X/PEX server. The format consists of twelve records of five fields each. The records define the size and number of predefined entries for the following tables:

**LINE_BUNDLE_TABLE**

**MARKER_BUNDLE_TABLE**

**TEXT_BUNDLE_TABLE**

**INTERIOR_BUNDLE_TABLE**

**EDGE_BUNDLE_TABLE**

**PATTERN_TABLE**

**TEXT_FONT_TABLE**

**COLOUR_TABLE**

**VIEW_TABLE**

**LIGHT_TABLE**

**DEPTH_CUE_TABLE**

**COLOUR_APPROXIMATION_TABLE**

The contents of each of the five fields are given below:

**START_INDEX** The starting index for the table. Usually given as 1 or 0.

**MAX_TABLE_ENTRIES** The maximum number of entries allowed in the lookup table. This is the only field that the user may change.

**NUMBER_PREDEFINED** The number of predefined entries in the lookup table.

**FIRST_PREDEFINED_INDEX** The starting index of the predefined entries in the lookup table.

**LAST_PREDEFINED_INDEX** The ending index of the predefined entries in the lookup table.

The **config** file must start with a C-style comment (delimited by /* and */) and, while the comment itself can be multi-lined, there can only be one set of comment delimiters. Comments can appear in the body of the file (see the entries for view and light table) either on a new line or at the end of an existing line. As with the first comment, comments can be multi-lined but there can only be one set of comment delimiters.

**Note:** Once the server is running, these table sizes cannot be changed.

The following example shows the contents of the default configuration file (**/usr/lib/X11/pex_config.dat**)

```
/* PEX Configuration File:
   Only MAX_TABLE_ENTRIES can be changed by the user. */
```

```
LINE_BUNDLE_TABLE
        START_INDEX 1
        MAX_TABLE_ENTRIES 20
        NUMBER_PREDEFINED 1
        FIRST_PREDEFINED_INDEX 1
        LAST_PREDEFINED_INDEX 1
MARKER_BUNDLE_TABLE
        START_INDEX 1
        MAX_TABLE_ENTRIES 20
        NUMBER_PREDEFINED 1
        FIRST_PREDEFINED_INDEX 1
        LAST_PREDEFINED_INDEX 1
TEXT_BUNDLE_TABLE
        START_INDEX 1
        MAX_TABLE_ENTRIES 20
        NUMBER_PREDEFINED 1
        FIRST_PREDEFINED_INDEX 1
        LAST_PREDEFINED_INDEX 1
INTERIOR_BUNDLE_TABLE
        START_INDEX 1
        MAX_TABLE_ENTRIES 20
        NUMBER_PREDEFINED 1
        FIRST_PREDEFINED_INDEX 1
        LAST_PREDEFINED_INDEX 1
EDGE_BUNDLE_TABLE
        START_INDEX 1
        MAX_TABLE_ENTRIES 20
        NUMBER_PREDEFINED 1
        FIRST_PREDEFINED_INDEX 1
        LAST_PREDEFINED_INDEX 1
PATTERN_TABLE
        START_INDEX 1
        MAX_TABLE_ENTRIES 0
        NUMBER_PREDEFINED 0
        FIRST_PREDEFINED_INDEX 0
        LAST_PREDEFINED_INDEX 0
TEXT_FONT_TABLE
        START_INDEX 1
        MAX_TABLE_ENTRIES 20
        NUMBER_PREDEFINED 2
        FIRST_PREDEFINED_INDEX 1
        LAST_PREDEFINED_INDEX 2
COLOUR_TABLE
        START_INDEX 0
```

```
        MAX_TABLE_ENTRIES 256
        NUMBER_PREDEFINED 8
        FIRST_PREDEFINED_INDEX 0
        LAST_PREDEFINED_INDEX 7
VIEW_TABLE
        START_INDEX 0
        MAX_TABLE_ENTRIES 20 /* This size cannot be set
                                higher than 32 */
        NUMBER_PREDEFINED 6
        FIRST_PREDEFINED_INDEX 0
        LAST_PREDEFINED_INDEX 5
LIGHT_TABLE
        START_INDEX 1
        MAX_TABLE_ENTRIES 12 /* This size cannot be set
                                higher than 12 */
        NUMBER_PREDEFINED 1
        FIRST_PREDEFINED_INDEX 1
        LAST_PREDEFINED_INDEX 1
DEPTH_CUE_TABLE
        START_INDEX 0
        MAX_TABLE_ENTRIES 6
        NUMBER_PREDEFINED 2
        FIRST_PREDEFINED_INDEX 0
        LAST_PREDEFINED_INDEX 1
COLOUR_APPROXIMATION_TABLE
        START_INDEX 0
        MAX_TABLE_ENTRIES 0
        NUMBER_PREDEFINED 0
        FIRST_PREDEFINED_INDEX 0
        LAST_PREDEFINED_INDEX 0
```

## Configuring Graphics Memory

When the X server is started, its first task is to establish the shared global memory, called *structure memory*. The size of this memory, plus other system parameters, are kept in a file named **gm_config.dat**. This file can be changed for each ESV Workstation, allowing a small amount of tuning to meet individual needs. The server looks first for the file named by the environment variable ***GM_CONFIG***. If this is not set, the server looks for **/usr/lib/X11/gm_config.dat**.

The **gm_config.dat** file contains keyword commands and arguments. The commands do not need to come in any special order. C-style comments may be included in the file, delimited by **/* */**, and with a space before and after each delimiter.

The following key words are allowed in the **gm_config.dat** file:

- **SM_SIZE *size***

  This keyword, with its argument ***size***, tells the structure walker how large in bytes to make structure memory. The default size is 16 Mbytes. For efficiency, sizes should be multiples of 4096. The user might want to increase ***size*** from its default if the system starts returning error messages to the effect that processes are running out of structure memory.

- **NUM_PROCESSES *number***

  This keyword, with its argument ***number***, tells the structure walker the maximum number of simultaneously running graphics systems. The graphics systems that are counted are 1) the ES X/PEX server, and 2) each ES/PSX application. This number is used to reserve space for structure memory management data structures. The structure walker must reserve space for memory management bookkeeping before any applications can allocate structure memory. Each graphics application needs 128 words of structure memory for this purpose. The default number of maximum processes is 20. The smaller the number entered here, the more structure memory is available for use.

  Note that no matter how many PHIGS applications are running, they count in a group as one with the X server.

- **SW_STACK_SIZE *number***

  This keyword, with its argument ***number***, specifies the size of the structure walker's traversal stack in long words. This stack is used by the structure walker as it traverses graphics hierarchical structures. The stack size limits the number of levels that a graphics structure can have. The default for this item is 500.

- **SW_ATTR_STACK_SIZE *number***

  This keyword, with its argument ***number***, specifies the size of the structure walker's attribute stack in long words. This stack is used by the structure walker as it traverses graphics hierarchical structures. Graphical attributes that change in a structure are pushed onto this stack. The attribute stack size limits the number of levels that a graphics structure can have. The default for this item is 1000.

- **SW_MATRIX_STACK_SIZE *number***

  This keyword, with its argument ***number,*** specifies the size of the structure walker's matrix stack in long words. This stack is used by the structure walker as it traverses graphics hierarchical structures. The matrix stack size limits the number of levels that a graphics structure can have. The default for this item is 4000.

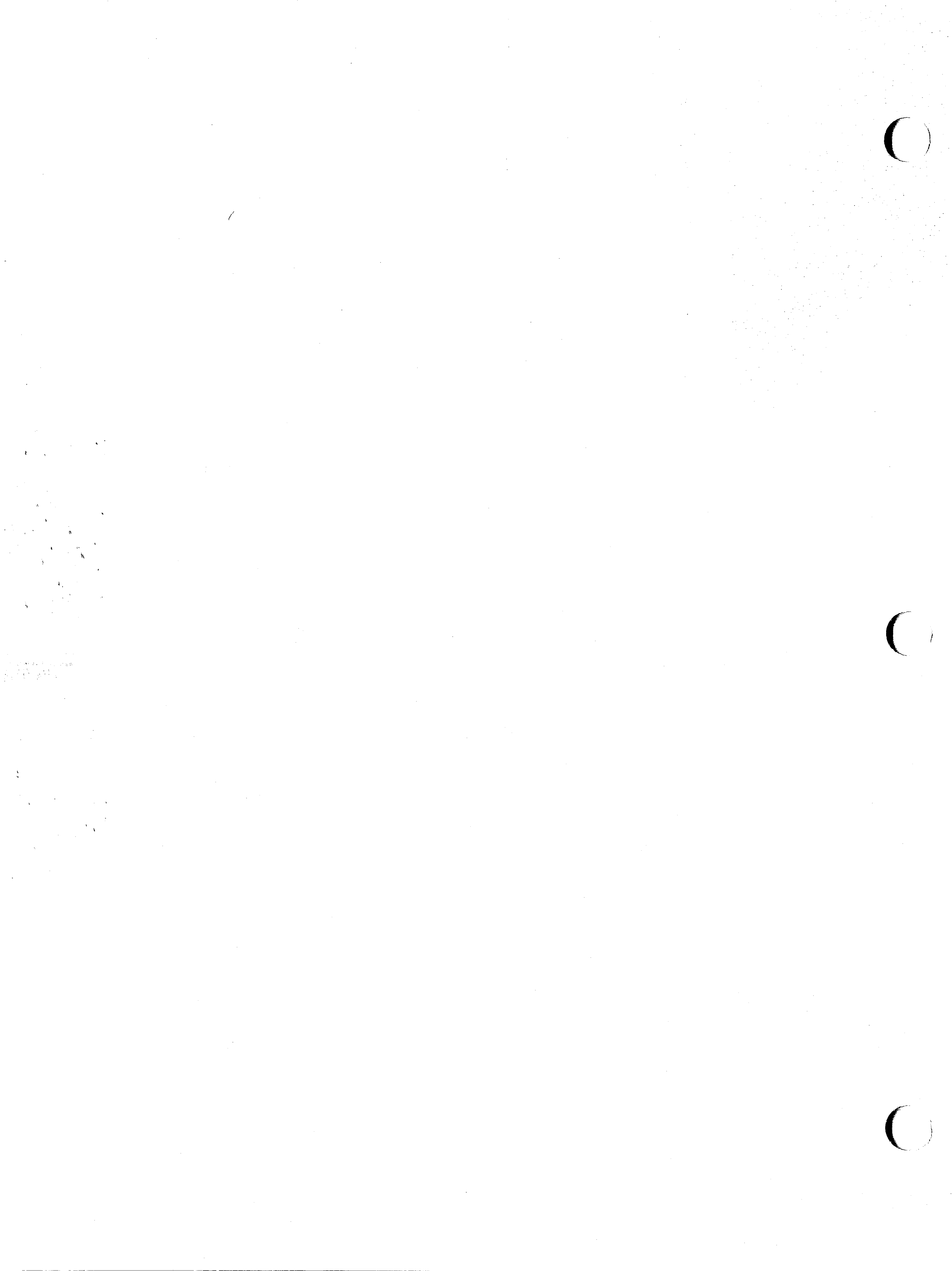
- **DEFAULT_RGB_CLUT *table***

  This keyword, with the 256 color lookup table values listed in ***table***, specifies a default lookup table for the main RGB color lookup table. The table is generally gamma-corrected and designed to fit specific color monitor characteristics. The 256 values should be given in C-style hex notation and should be separated by spaces or by a new line.

### Example gm_config.dat File

```
/* CONFIG file for the Graphics Manager */
/* ---------------------------------- */
SM_SIZE 4194304    /* 4 Meg */
NUM_PROCESSES 20   /* 20 processes allowed to alloc structure memory */
/* Structure Walker stack sizes in Longwords */
SW_STACK_SIZE 500
SW_ATTR_STACK_SIZE 1000
SW_MATRIX_STACK_SIZE 4000
/* Default Gamma-corrected RGB table. */
DEFAULT_RGB_CLUT
0x000000 0x0d0d0d 0x131313 0x181818 0x1c1c1c 0x202020 0x232323 0x262626
0x292929 0x2b2b2b 0x2e2e2e 0x303030 0x333333 0x353535 0x373737 0x393939
0x3b3b3b 0x3d3d3d 0x3f3f3f 0x414141 0x424242 0x444444 0x464646 0x474747
0x494949 0x4b4b4b 0x4c4c4c 0x4e4e4e 0x4f4f4f 0x515151 0x525252 0x545454
0x555555 0x565656 0x585858 0x595959 0x5b5b5b 0x5c5c5c 0x5d5d5d 0x5e5e5e
0x606060 0x616161 0x626262 0x636363 0x656565 0x666666 0x676767 0x686868
0x696969 0x6b6b6b 0x6c6c6c 0x6d6d6d 0x6e6e6e 0x6f6f6f 0x707070 0x717171
0x727272 0x737373 0x747474 0x767676 0x777777 0x787878 0x797979 0x7a7a7a
0x7b7b7b 0x7c7c7c 0x7d7d7d 0x7e7e7e 0x7f7f7f 0x808080 0x818181 0x828282
0x838383 0x848484 0x848484 0x858585 0x868686 0x878787 0x888888 0x898989
0x8a8a8a 0x8b8b8b 0x8c8c8c 0x8d8d8d 0x8e8e8e 0x8f8f8f 0x8f8f8f 0x909090
0x919191 0x929292 0x939393 0x949494 0x959595 0x959595 0x969696 0x979797
0x989898 0x999999 0x9a9a9a 0x9a9a9a 0x9b9b9b 0x9c9c9c 0x9d9d9d 0x9e9e9e
0x9f9f9f 0x9f9f9f 0xa0a0a0 0xa1a1a1 0xa2a2a2 0xa3a3a3 0xa3a3a3 0xa4a4a4
0xa5a5a5 0xa6a6a6 0xa6a6a6 0xa7a7a7 0xa8a8a8 0xa9a9a9 0xa9a9a9 0xaaaaaa
0xababab 0xacacac 0xacacac 0xadadad 0xaeaeae 0xafafaf 0xafafaf 0xb0b0b0
0xb1b1b1 0xb2b2b2 0xb2b2b2 0xb3b3b3 0xb4b4b4 0xb5b5b5 0xb5b5b5 0xb6b6b6
0xb7b7b7 0xb7b7b7 0xb8b8b8 0xb9b9b9 0xb9b9b9 0xbababa 0xbbbbbb 0xbcbcbc
0xbcbcbc 0xbdbdbd 0xbebebe 0xbebebe 0xbfbfbf 0xc0c0c0 0xc0c0c0 0xc1c1c1
0xc2c2c2 0xc2c2c2 0xc3c3c3 0xc4c4c4 0xc4c4c4 0xc5c5c5 0xc6c6c6 0xc6c6c6
0xc7c7c7 0xc8c8c8 0xc8c8c8 0xc9c9c9 0xcacaca 0xcacaca 0xcbcbcb 0xcccccc
0xcccccc 0xcdcdcd 0xcdcdcd 0xcecece 0xcfcfcf 0xcfcfcf 0xd0d0d0 0xd1d1d1
0xd1d1d1 0xd2d2d2 0xd3d3d3 0xd3d3d3 0xd4d4d4 0xd4d4d4 0xd5d5d5 0xd6d6d6
0xd6d6d6 0xd7d7d7 0xd7d7d7 0xd8d8d8 0xd9d9d9 0xd9d9d9 0xdadada 0xdbdbdb
0xdbdbdb 0xdcdcdc 0xdcdcdc 0xdddddd 0xdedede 0xdedede 0xdfdfdf 0xdfdfdf
0xe0e0e0 0xe0e0e0 0xe1e1e1 0xe2e2e2 0xe2e2e2 0xe3e3e3 0xe3e3e3 0xe4e4e4
0xe5e5e5 0xe5e5e5 0xe6e6e6 0xe6e6e6 0xe7e7e7 0xe7e7e7 0xe8e8e8 0xe9e9e9
0xe9e9e9 0xeaeaea 0xeaeaea 0xebebeb 0xebebeb 0xececec 0xededed 0xededed
0xeeeeee 0xeeeeee 0xefefef 0xefefef 0xf0f0f0 0xf0f0f0 0xf1f1f1 0xf2f2f2
0xf2f2f2 0xf3f3f3 0xf3f3f3 0xf4f4f4 0xf4f4f4 0xf5f5f5 0xf5f5f5 0xf6f6f6
0xf6f6f6 0xf7f7f7 0xf8f8f8 0xf8f8f8 0xf9f9f9 0xf9f9f9 0xfafafa 0xfafafa
0xfbfbfb 0xfbfbfb 0xfcfcfc 0xfcfcfc 0xfdfdfd 0xfdfdfd 0xfefefe 0xfefefe
```

# 5. Video Output Guide



# Table of Contents


**5. Video Output Guide .......... 5-1**

Introduction .......... 5-1
Raster Monitors .......... 5-1
Video Signals .......... 5-1
Multiple Video Hookups .......... 5-2
Video Output Specifications .......... 5-3
- Connections .......... 5-3
- Voltage Levels .......... 5-3
- Video Timing Formats .......... 5-3
- Changing Video Timing Formats .......... 5-4
- Video Timing Specifications .......... 5-5
- Nomenclature .......... 5-5
- Video Output Signals .......... 5-13

Recording Screen Images .......... 5-15
- Overview .......... 5-15
- Still Camera .......... 5-15
- Film Recorder .......... 5-15
- Plotter .......... 5-16
- Video Camera Recording .......... 5-16
- Monitor Characteristics .......... 5-16

# 5. Video Output Guide

## Introduction

This chapter describes the video output from the ESV Workstation. It contains the following information:

- A discussion of raster video theory,
- A summary of the video output specifications for the ESV Workstation, and
- A description of techniques for recording screen images.

## Raster Monitors

Raster monitors are either *interlaced* or *non-interlaced.* Non-interlaced monitors draw all of the horizontal lines each time they draw from the top to the bottom of the screen. All the lines drawn on the screen make up a *frame*. The default ESV Workstation video timing format is non-interlaced.

Commercial television is an example of an interlaced monitor. Interlaced monitors draw the first scan line on the top of the screen, then draw the third scan line, then the fifth, and so on to the bottom of the screen, drawing all the odd numbered scan lines. The beam then goes back to the top of the screen and draws all of the even numbered scan lines. The odd numbered scan lines are called the *odd field* and the even numbered scan lines are called the *even field.* The two fields make up one frame.

Interlaced monitors can have the same resolution as a non-interlaced monitor with only about 60% of the performance of the non-interlaced monitor because the interlaced monitor draws a given pixel only one-half as often as the non-interlaced monitor. The field rate is still 60 Hz, so there is not a problem with flicker. Interlaced monitors produce some peculiarities (*artifacts*) when there is motion in the picture. If the eye follows an object that is moving vertically, the object can move one scan line per field, and the eye will see the two fields superimposed on each other. The object appears to have horizontal black lines drawn through it. This is one of the class of artifacts known as *temporal aliasing*. One familiar example of temporal aliasing from western movies is a wagon wheel appearing to turn backwards.

## Video Signals

The ESV Workstation generates three video signals: red, green, and blue. There is also a composite sync signal at TTL levels

The ESV Workstation controls the monitor by regulating how bright the electron beam is for each point of the picture and when the beam draws each

scan line. The information that controls the brightness of the beam is called *active video*. The information that tells the beam when to go back and forth in the horizontal direction is called *horizontal sync*. The information that tells the beam to go back and forth in the vertical direction is called *vertical sync*. Vertical sync and horizontal sync are often combined to form *composite sync*. The information is separated in the monitor by special filters.

The composite sync signal does not need to convey any information while active video information is being sent. This allows the composite sync signal to be included with one of the video signals. The resulting signal is called *composite video*. Most color video terminals have one video signal for each of the primary colors of light: red, green, and blue (RGB). The composite sync signal is usually included with the green video signal. The ESV Workstation uses this RGB video system with composite sync carried on the green video signal.

The video signal has defined voltage levels which convey information about brightness, composite sync, and more to the monitor. The Electronic Industries Association (EIA) RS-343-A standard is one common standard. The signal voltage levels from the ESV Workstation conform to this standard.

## Multiple Video Hookups

The RGB video signals are carried on three coaxial cables bundled together in a large shielded cable. The coaxial cable has a characteristic impedance of 75 ohms. The connectors are standard BNC.

The ESV Workstation generates only one set of video signals, so multiple output devices, such as monitors and cameras, must be connected in a *daisy chain*. The ESV Workstation's monitor does not have *loop-thru* capability, so the first device hooked up to the ESV Workstation must have both inputs and *loop-thru* outputs. The next device should have its inputs connected to the loop-thru outputs of the first device. Each additional device along the chain works the same way. All devices except the device at the end of the chain must have termination removed or turned off. The last device in the chain must terminate the video signals. This is usually done with a built-in 75 ohm switchable resistor, although termination may be done with BNC caps. Coax-t connections are not recommended.

Picture quality may degrade if more than one device is connected to the ESV Workstation. In such cases, Evans & Sutherland recommends the use of wide-band video distribution amplifiers with a minimum bandwidth of 110 MHz to implement active loop-thru. When using these active loop-thru devices, all devices should be terminated at 75 ohms.

## Video Output Specifications

### Connections

There are four BNC connectors, one each for red, blue, green and TTL composite sync. Composite sync is also on green.

### Voltage Levels

Voltage levels are given referenced to *earth* ground. All video signals must be terminated in 75 ohms to ground.

**Table 5-1. Voltage levels**

| Value | RGB w/o Sync | Green with Sync |
|---|---|---|
| Reference White | 0.714 V | 1.000 V |
| Reference Black | 0.054 V | 0.340 V |
| Blanking | 0.000 V | 0.286 V |
| Sync | N/A | 0.000 V |

These levels conform to the EIA RS-343-A standard. All TTL output signals have the following drive:

- Low level output < 0.1V DC at 24 mA.
- High level output > 4.7V DC at 24 mA.

### Video Timing Formats

The ESV Workstation software supports the following seven video timing formats:

- 1280 by 1024, 60 Hz, non-interlaced (default)
- 640 by 1024, 56 Hz, interlaced
- 1280 by 1024, 56 Hz, interlaced
- RS-343-A 1280 by 1024, 30 Hz, interlaced
- RS-343-A 1260 by 946, 30 Hz, interlaced
- PAL/SECAM RGB, 768 by 574, 25 Hz, interlaced
- RS-170-A RGB, 640 by 484, 30 Hz, interlaced

The 1280 by 1024 non-interlaced format is the default ESV Workstation video timing format. This format does not conform to any established standard video timing format.

Different video timing formats are used to display pictures. Every monitor is configured to work with certain video timing format(s). The important features of a video timing format are:

- *Pixel rate* is the rate at which video information can change. The default ESV Workstation video timing format has a pixel rate of approximately 107 MHz.
- *Horizontal frequency* is the number of times a horizontal line is drawn across the screen each second. The default ESV Workstation video timing format has a horizontal frequency of 64 KHz.
- *Field rate* is the rate at which the electron beam goes from the top to the bottom of the screen. The default ESV Workstation video timing format has a field rate of 60 Hz.
- *Frame rate* is the rate at which the entire screen is redrawn. In non-interlaced monitors, the frame rate is the same as the field rate.

All timing information is conveyed on the composite sync signal. The ESV Workstation generates only one composite sync signal, so only one video timing format can be generated at a time. A different video timing format requires a different composite sync signal.

Monitors with different video timing formats can be hooked up at the same time, but only monitors that are configured for the video timing format which is generated by the ESV Workstation at that time will produce a good picture.

## Changing Video Timing Formats

### ES/PSX Option

This section discusses how to accomplish video format changes with the ES/PSX option. Switching from one video timing format to another is done by sending values to input <5> of the ES/PSX function **PS390ENV**. There are three values which the function will accept on this input:

- Sending 0 to <5> turns on a format that is non-interlaced, single-field, 1024 x 1280 pixels, consecutive memory scanlines. This is the default monoscopic format.
- Sending 2 to <5> turns on a stereo format that is interlaced, dual-field, each 512 x 1280 pixels. The odd field consists of memory scanlines 0 to 511, and the even field consists of memory scanlines 512 to 1023. This format gives twice the resolution in the *x*-axis as the stereo format described in the next paragraph.
- Sending 3 to <5> turns on a stereo format that is interlaced, dual-field with each field 512 x 640 pixels. The odd field consists of memory scanlines 0 to 511, and the even field consists of memory scanlines 512 to 1023. This format does not display pixels 640 to 1279.

The video timing format changes on the subsequent video frame without regard for the structure walker's position. This can produce momentary visual

oddities. Because the routine does its work by setting bits in sync memory, graphics must previously be initialized by the application to allow access to sync memory.

**X Clients**

An X client can change the video mode (video output format) of the ESV workstation by calling the **XVideoMode** function:

```
#include <X11/extensions/XVideoMode.h>
int  XVideoMode(display, mode)
   Display  * display;
   int        mode;
```

***display*** Specifies the connection to the X server.

***mode*** Specifies the video mode to switch to.

**Note:** In order to include the **X11/extensions/XVideoMode.h** file, you must also include the **X11/Xmd.h** file to get the appropriate typedefs.

Possible constants to use for ***mode***:

| | |
|---|---|
| Monoscopic | Normal video for ESV. |
| RS_343_A_1280x1024 | Special video format. |
| RS_343_A_1260x946 | Special video format. |
| PAL_SECAM_768x574 | Special video format. |
| RS_170_A_640x480 | Special video format. |
| Stereo60KHzIntStor1280x1024 | Special video format. |
| Stereo60KHzSplitStor1280x1024 | Special video format. |
| Stereo60KHzIntStor640x1024 | Special video format. |
| Stereo60KHzSplitStor640x1024 | Special video format. |

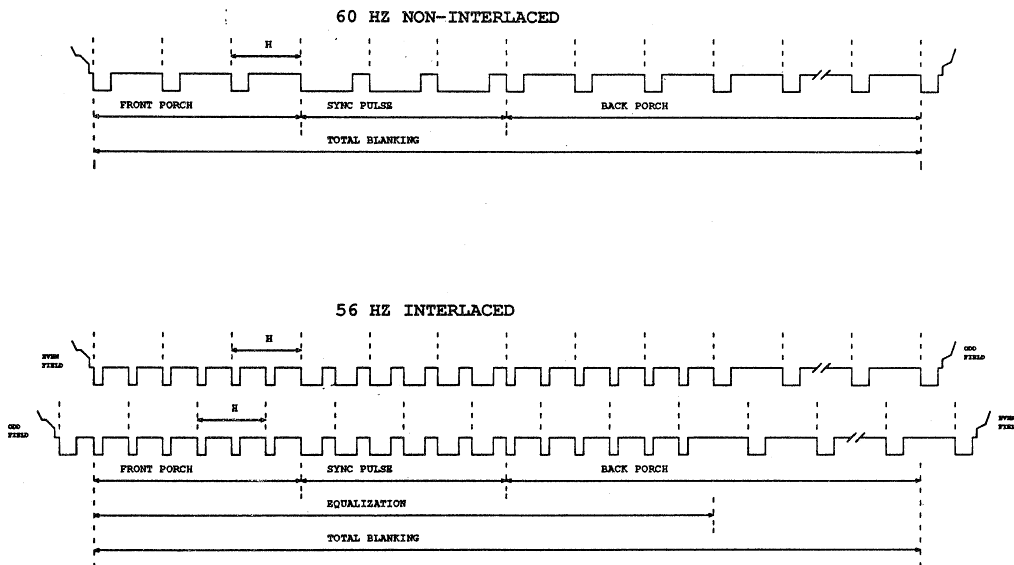
## Video Timing Specifications

Tables 5-2 through 5-8 show the video timing specifications for the video timing formats supported by the ESV Workstation.

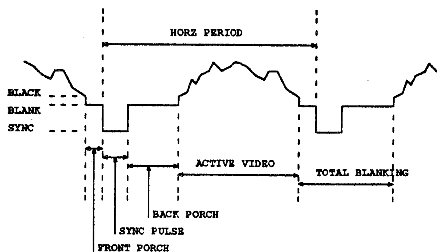
## Nomenclature

Figure 5-1 illustrates the timing terms defined below.

- *Front Porch* refers to the time interval between the end of active video and the beginning of sync, during which the video is at Blank level.
- *Back porch* refers to the time interval between the end of sync and the beginning of active video, during which the video is at Blank level.
- The symbol *H*, when used in the vertical timing specification, means one horizontal period.




**Vertical blanking intervals**




**Horizontal timing key**

**Figure 5-1. Video timing diagrams**

**Table 5-2. 1280 by 1024, 60 Hz, non-interlaced**

| | |
|---|---|
| Aspect Ratio, H:V: | 5:4 |
| Horizontal timing: | |
| Frequency: | 63.96 KHz |
| Front Porch: | .4095 µs |
| Sync Pulse: | 1.712 µs |
| Back Porch: | 1.601 µs |
| Total Blanking: | 3.723 µs |
| Active Video: | 11.91 µs |
| Horiz. Period: | 15.63 µs |
| Pixels Displayed: | 1280 |
| Vertical Timing: | |
| Frequency: | 60.00 Hz per field |
| Front Porch: | 3H |
| Sync Pulse: | 3H |
| Back Porch: | 36H |
| Total Blanking: | 42H (.6567 ms) per field |
| Active Video: | 1024H (16.01 ms) per field |
| Total Vertical Time: | 1066H (16.67 ms) |
| Pixel Frequency: | 107.4528 MHz |
| Pixel Period: | 9.3064 ns |

**Table 5-3. 640 by 1024, 56 Hz, interlaced**

| | |
|---|---|
| Aspect Ratio, H:V: | 5:4 |
| Horizontal timing: | |
| Frequency: | 60.50 KHz |
| Front Porch: | .8190 µs |
| Sync Pulse: | 1.712 µs |
| Back Porch: | 2.085 µs |
| Total Blanking: | 4.616 µs |
| Active Video: | 11.91 µs |
| Horiz. Period: | 16.53 µs |
| Pixels Displayed: | 640 |
| Vertical timing: | |
| Frequency: | 55.66 Hz |
| Front Porch: | 3H |
| Sync Pulse: | 3H |
| Back Porch: | 25H |
| Total Blanking: | 31H (.5124 ms) per field |
| Active Video: | 512.5H (8.471 ms) per field |
| Total Vertical Time: | 1087H (17.97 ms) |
| Pixel Frequency: | 53.7264 MHz |
| Pixel Period: | 18.6128 ns |

Note: All vertical frequencies denote frame rate, not field rate, and all interlaced formats are 2:1 interlaced.

**Table 5-4. 1280 by 1024, 56 Hz, interlaced**

| | |
|---|---|
| Aspect Ratio, H:V: | 5:4 |
| Horizontal timing: | |
| Frequency: | 60.50 KHz |
| Front Porch: | .8190 μs |
| Sync Pulse: | 1.712 μs |
| Back Porch: | 2.085 μs |
| Total Blanking: | 4.616 μs |
| Active Video: | 11.91 μs |
| Horiz. Period: | 16.53 μs |
| Pixels Displayed: | 1280 |
| Vertical timing: | |
| Frequency: | 55.66 Hz |
| Front Porch: | 3H |
| Sync Pulse: | 3H |
| Back Porch: | 25H |
| Total Blanking: | 31H (.5124 ms) per field |
| Active Video: | 512.5H (8.471 ms) per field |
| Total Vertical Time: | 1087H (17.97 ms) |
| Pixel Frequency: | 107.4528 MHz |
| Pixel Period: | 9.3064 ns |

**Table 5-5. RS-343-A 1280 by 1024, 30 Hz, interlaced**

| | |
|---|---|
| Aspect Ratio, H:V: | 5:4 |
| Horizontal timing: | |
| Frequency: | 33.21 KHz |
| Front Porch: | .7962 µs |
| Sync Pulse: | 2.751 µs |
| Back Porch: | 3.402 µs |
| Total Blanking: | 6.949 µs |
| Active Video: | 23.16 µs |
| Horiz. Period: | 30.11 µs |
| Pixels Displayed: | 1280 |
| Vertical Timing: | |
| Frequency: | 60.00 Hz per field |
| Front Porch: | 3H |
| Sync Pulse: | 4H |
| Back Porch: | 34H |
| Total Blanking: | 41H (1.235 ms) per field |
| Active Video: | 512.5H (15.43 ms) per field |
| Total Vertical Time: | 1107H (33.33 ms) |
| Pixel Frequency: | 55.2614 MHz |
| Pixel Period: | 18.0958 ns |

**Table 5-6. RS-343-A 1260 by 946, 30 Hz, interlaced**

| | |
|---|---|
| Aspect Ratio, H:V: | 4:3 |
| Horizontal timing: | |
| Frequency: | 30.69 KHz |
| Front Porch: | .8105 µs |
| Sync Pulse: | 2.756 µs |
| Back Porch: | 3.485 µs |
| Total Blanking: | 7.052 µs |
| Active Video: | 25.53 µs |
| Horiz. Period: | 32.58 µs |
| Pixels Displayed: | 1260 |
| Vertical Timing: | |
| Frequency: | 60.00 Hz per field |
| Front Porch: | 3H |
| Sync Pulse: | 4H |
| Back Porch: | 31H |
| Total Blanking: | 38H (1.238 ms) per field |
| Active Video: | 473.5H (15.43 ms) per field |
| Total Vertical Time: | 1023H (33.33 ms) |
| Pixel Frequency: | 49.3495 MHz |
| Pixel Period: | 20.2636 ns |

**Table 5-7. PAL/SECAM RGB, 768 by 574, 25 Hz, interlaced**

| | |
|---|---|
| Aspect Ratio, H:V: | 4:3 |
| Horizontal timing: | |
| Frequency: | 15.63 KHz |
| Front Porch: | 1.627 µs |
| Sync Pulse: | 4.610 µs |
| Back Porch: | 5.695 µs |
| Total Blanking: | 11.93 µs |
| Active Video: | 52.07 µs |
| Horiz. Period: | 64.00 µs |
| Pixels Displayed: | 768 |
| Vertical Timing: | |
| Frequency: | 50.00 Hz per field |
| Front Porch: | 2.5H |
| Sync Pulse: | 2.5H |
| Back Porch: | 20H |
| Total Blanking: | 25H (1.600 ms) per field |
| Active Video: | 287.5H (18.40 ms) per field |
| Total Vertical Time: | 625H (40.0 ms) |
| Pixel Frequency: | 14.7500 MHz |
| Pixel Period: | 67.7966 ns |

**Table 5-8. RS-170-A RGB, 640 by 484, 30 Hz, interlaced**

| | |
|---|---|
| Aspect Ratio, H:V: | 4:3 |
| Horizontal timing: | |
| Frequency: | 15.73 KHz |
| Front Porch: | 1.638 μs |
| Sync Pulse: | 4.914 μs |
| Back Porch: | 4.586 μs |
| Total Blanking: | 11.14 μs |
| Active Video: | 52.42 μs |
| Horiz. Period: | 63.56 μs |
| Pixels Displayed: | 640 |
| Vertical Timing: | |
| Frequency: | 59.94 Hz per field |
| Front Porch: | 3H |
| Sync Pulse: | 3H |
| Back Porch: | 14H |
| Total Blanking: | 20H (1.271 ms) per field |
| Active Video: | 242.5H (15.41 ms) per field |
| Total Vertical Time: | 525H (33.36 ms) |
| Pixel Frequency: | 12.2098 MHz |
| Pixel Period: | 81.9014 ns |

## Video Output Signals

**REDVIDEOOUT:**

Red video output (analog signal). Signal conforms to RS-343-A standard voltage levels when terminated in 75 ohms to ground. Signal is intended to drive a 75 ohm coax and be terminated at the end of the coax with a 75 ohm resistance to ground. Full brightness is 0.714 V, black is 0.054 V, and blank is 0.000 V.

**GRNVIDEOOUT:**

Green video output (analog signal). Signal conforms to RS-343-A standard voltage levels when terminated in 75 ohms to ground. Signal is intended to drive a 75 ohm coax and be terminated at the end of the coax with a 75 ohm resistance to ground. Composite sync is also carried on this signal, but may be optionally disabled. If composite sync is enabled, full brightness is 1.000 V, black is 0.340 V, blank is 0.286 V, and sync is 0.000 V. If composite

sync is not enabled, full brightness is 0.714 V, black is 0.054 V, and blank is 0.000 V.

**BLUVIDEOOUT:**

Blue video output (analog signal). Signal conforms to RS-343-A standard voltage levels when terminated in 75 ohms to ground. Signal is intended to drive a 75 ohm coax and be terminated at the end of the coax with a 75 ohm resistance to ground. Full brightness is 0.714 V, black is 0.054 V, and blank is 0.000 V.

***CSYNCOUT:**

Composite synchronization output (TTL, active low). Composite sync signal to video peripherals, such as monitors. Its active edge is the falling edge, but most video peripherals call this same polarity CSYNC.

## Recording Screen Images

### Overview

There are a variety of ways to capture ESV Workstation screen images, including still photography, film recording, hardcopy plotters, and video camera recording.

This section provides an overview of the different methods to generate photographs, hardcopy plots, film recordings, and video camera recordings of screen images.

### Still Camera

The easiest way to record a workstation image is to take a picture with a still camera. The workstation takes one sixtieth of a second to draw a complete picture on the screen, so the exposure must be longer than one sixtieth of a second. One fifteenth of a second exposure is preferable. If you are filming both wireframe and rendered images on the same screen, you should use a double exposure, one for the wireframe, and one for the rendered object. At slow exposure speeds, mount the camera on a tripod, and use a remote shutter release.

The face of the system monitor curves out towards the camera, so depth-of-field may be a problem. Kodachrome 25 or 64 color slide film is recommended for 35 mm cameras. The photography should be done in a totally dark room. Opaque tape should be placed over the power indicator light on the monitor. Do not use flash.

### Film Recorder

Several companies make special camera systems for filming computer-generated images. The system consists of a high resolution monochrome (black

and white) monitor, a set of interchangeable red, green, and blue filters, a camera back, and control circuitry. The camera system puts the green video on the black and white monitor, takes a picture through the green filter, then puts the red video on the black and white monitor, takes a picture through the red filter, and then repeats the process for blue. Each picture requires three exposures. These camera systems generally accept a high resolution interlaced format.

This method yields the highest quality image recording. This technique is not well suited to filming dynamic motion. However, this method can be used to animate motion sequences. Most cameras offer a movie camera film back designed for use in animation.

**Plotter**

Several companies make plotters which connect to the RGB video outputs from the ESV Workstation. The plotter has a video interface which samples and digitizes the video signals over about 30 seconds. The plotter then prints the image onto paper. Unfortunately, these plotters have poor color resolution, so spatial resolution is usually sacrificed to give better color resolution using a technique known as *dithering*. The quality of the hardcopy image will be inferior to the image quality seen on the monitor.

**Video Camera Recording**

It is possible to record video directly from the ESV Workstation screen using a video camera. The recording should be done in a totally dark room. Be sure to cover the power indicator on the monitor with opaque tape. This direct method of filming the workstation screen using a video camera may not work well due to a difference between the screen refresh rate of the workstation and the television system being used. The workstation refreshes the screen at 60 Hz. NTSC television refreshes the screen at 59.94 Hz. PAL/SECAM television refreshes the screen at 50 Hz.

Unfortunately, the NTSC encoding system used for color television in the U.S.A. gives poor spatial resolution of colors in the horizontal direction. Vertical color boundaries between different colors such as red and blue show a moving *chroma crawl* (or *zipper*) pattern on the recording. The VHS system of recording has resolution that is even worse, so the zipper pattern will not be as noticeable.

Pastel colors provide a better quality recording than the saturated RGB colors used most often in computer graphics applications. The BetaCam system provides better color resolution.

### Monitor Characteristics

The ESV Workstation monitor uses medium-persistence B22 phosphor. The International Commission on Illumination (Commission Internationale de l'Elairage, or CIE) standard coordinates of the phosphors are shown in table 5-9.

**Table 5-9. CIE phosphor coordinates**

| Color | x | y |
|---|---|---|
| Red | 0.62 | 0.35 |
| Green | 0.29 | 0.60 |
| Blue | 0.15 | 0.065 |

For white, the color temperature of the monitor at screen center is:

9300 degrees Kelvin + 7 M.P.C.D.

where, M.P.C.D. = minimum perceptible color difference

The color coordinates are:

$x = 0.283 \pm 0.020$

$y = 0.297 \pm 0.020$

6. Editors

# Table of Contents

**6. Editors ........................................................................................ 6-1**

Introduction ........................................................................................ 6-1
vi Editor ........................................................................................ 6-1
Create a File or Edit an Existing File ............................................ 6-1
Command Mode/Input Mode ............................................ 6-1
Moving Around the Screen ............................................ 6-2
Text Operations ............................................ 6-2
Quitting/Saving Files ............................................ 6-3
Search Text ............................................ 6-3
GNU Emacs ........................................................................................ 6-4
Starting Emacs ............................................ 6-4
Starting the Tutorial ............................................ 6-4
Getting Help ............................................ 6-4
Leaving Emacs ............................................ 6-5
Files ............................................ 6-5
Error Recovery ............................................ 6-5
Incremental Searches ............................................ 6-5
Cursor Movement ............................................ 6-6
Screen Movement ............................................ 6-6
Deleting ............................................ 6-6
Formatting ............................................ 6-7
Changing Case ............................................ 6-7

# 6. Editors

## Introduction

There are three editors available on the ESV Workstation: **ed, vi**, and **emacs** (GNU Emacs). The **vi** editor is more versatile than the **ed** editor, and is easier to get started with than the **emacs** editor. The **emacs** editor is more versatile than **vi**. This chapter discusses basics of **vi** and **emacs.**

## vi Editor

The **vi** editor allows you to create, edit and save files.

### Create a File or Edit an Existing File

A new file is made by entering: **vi [*newfilename*]**

Edit an existing file by entering: **vi [*oldfilename*]**

### Command Mode/Edit Mode

**vi** uses a *Command Mode* and *Edit Mode*. Command mode accepts keystrokes as commands, but does not display the characters as you type. Edit mode accepts keystrokes as text and will display it as it is entered. **vi** does not tell you which is the current mode unless showmode is set.

It is advisable to always turn **showmode** on when using **vi**. From command mode enter:

:set showmode

#### Command Mode

Command mode controls the screen, characters, words and line movement. To invoke the command mode enter an ESC (The escape key).If you are already in command mode, the terminal will beep. In command mode, no mode is displayed on the screen.

#### Edit Mode

The edit mode is entered by inputting one of the characters listed below under "Text Operations." This then allows text to be entered at the specified position. With showmode set, the mode will be displayed at the bottom of the screen. The various modes are:

- INSERT
- OPEN
- APPEND
- CHANGE

- REPLACE
- REPLACE 1 CHAR

### Moving Around the Screen

From command mode, you can move around the screen with the following:

| | |
|---|---|
| **h** or ← | Move left one character. |
| **j** or ↓ | Move down one line. |
| **k** or ↑ | Move up one line. |
| **l** or → | Move right one character. |
| **^f** | Page forward one page. |
| **^b** | Page backward one page. |
| ***n*G** | Move to line *n*. |
| **:$** | Move to last line in file. |
| **H** | Move to top line on screen. |
| **M** | Move to middle line on screen. |
| **L** | Move to last line on screen. |

Moving within a line:

| | |
|---|---|
| **$** | Move to the end of the line. |
| **^** | Move to the beginning of the line. |
| **b** | Move backward one word. |
| **w** | Move forward one word. |

### Text Operations

#### Adding New Text

From command mode, type one of the following to enter edit mode and begin adding text at the indicated position. Terminate each new line of text with a carriage return. Note that the current line is the one which contains the cursor. The mode is shown in parentheses.

| | |
|---|---|
| **a** | Add text immediately after the cursor. (APPEND) |
| **A** | Add text at the end of the current line. (APPEND |
| **i** | Insert text immediately before the cursor. (INSERT) |
| **I** | Insert text at the beginning of the current line. (INSERT) |
| **o** | Open a new line after the current line. (OPEN) |
| **O** | Open a new line above the current line. (OPEN) |

### Adding a Control Character

To add a control character, *i.e.*, a **^** (PS 390 routing byte), first type **^v** then the control character to be added. For example, to add the PS 390 command interpreter routing byte, you would type: **^v^\0**

### Editing Text

From command mode, type one of the following to enter an edit mode and begin editing text. Enter ESC to return to command mode. The mode is shown in parentheses.

| | |
|---|---|
| **cw** | Change a single word. A **$** will mark the end of the word being changed. (CHANGE) |
| **C** | Change from the cursor position to the end of the current line. A **$** will mark the end of the change area. (CHANGE) |
| **r** | Replace a single character and automatically return to command mode. (REPLACE 1 CHAR) |
| **R** | Replace text until command mode is manually reentered. (REPLACE) |
| **xp** | Transpose two characters, stay in command mode. |

### Deleting Text

From the command mode, enter one of the following to delete text.

| | |
|---|---|
| **D** | Delete from cursor to end of line. |
| **dd** | Delete the current line. |
| **dw** | Delete a single word. |
| **x** | Delete character under cursor. May use **#x** also. |
| **X** | Delete character to left of cursor. May use **#X** also. |

## Quitting/Saving Files

The following are done from the command mode. After entering the **:** (colon), a **:** is displayed at the bottom of the screen, after which you enter the desired letter.

| | |
|---|---|
| **:w** | Write the file to disk using the original filename. |
| **:w *new*** | Write the file to disk using the filename ***new***. |
| **:q** | Quits **vi** and returns to the system prompt. If the file has not been saved with **:w**, the system will ask the user to enter **q!** to discard the buffer, abandon all edits and return to the system prompt. |
| **:q!** | Abandons all edits and quits **vi**. |
| : **set all** | Display all **vi** option settings. |

**Search Text**

| | |
|---|---|
| ***/string*** | Search forward. ***string*** equals search characters. |
| ***?string*** | Search backward. ***string*** equals search characters. |

**vi defaults file**

The file **.exrc** in your home directory acts as a personal default option file for **vi**. Any **vi** option may be set in this file as a one line entry per option. **Showmode** and automatic line numbering are examples of options that may be set in **.exrc**. To set these two options, create a **.exrc** file in your home directory that contains the following two lines:

```
set showmode
set number
```

All **vi** sessions (no matter what directory you are in) will use the options set in your **.exrc** file.

## GNU Emacs

GNU Emacs is another display text editor that runs on the ESV Workstation. It provides facilities beyond simple text editing functions, and can be easily customized.

This section presents a summary of basic Emacs commands. Many commands involve holding the control (CTRL) key and pressing another key simultaneously. In this section this operation is indicated by a hyphen between the two keys, for example:

**CTRL-v**

Other commands involve pressing first the ESC key followed immediately by another key. This operation is indicated by a space between the two keys:

**ESC t**

### Starting Emacs

To start the editor after logging on to the system, type

> **`emacs`**

Emacs divides the screen into several windows. The largest of these is the text window in which you see the text you are editing. At the bottom of each text window is the *mode line* which describes information about the window, including the name of the buffer being displayed, the modes in use, and how far down the buffer you are looking. You can recognize the mode line easily because it is displayed in inverse video.

The line below the mode line is called the *echo area* and is used to *echo* (print out) the command characters that you type and small amounts of text for prompts and messages. For example, this line will echo a multi-character command and prompt you to enter the name of the file.

### Starting the Tutorial

Probably the best way to become familiar with the program is by working through the on-line tutorial. After starting Emacs you can start the tutorial by placing your pointer in the text window and entering:

**`CTRL-h t`**

You can use the key sequence **`CTRL-x CTRL-c`** to exit the tutorial.

### Leaving Emacs

| Key Sequence | Function |
|---|---|
| **CTRL-x CTRL-c** | Exit Emacs permanently. |

**Error Recovery**

| Key Sequence | Function |
| --- | --- |
| **CTRL-g** | Abort a partially typed or partially executed command. |
| **ESC-x revert-buffer** | Restore a buffer to its original contents. |
| **CTRL-l** | Redraw a corrupted screen. |
| **CTRL-x u** | Undo an unwanted change. |
| **ESC-x recover-file** | Recover a file lost by a system crash. |

**Cursor Movement**

| Key Sequence | Function |
| --- | --- |
| **CTRL-b** | Backward one character. |
| **CTRL-f** | Forward one character. |
| **ESC-b** | Backward one word. |
| **ESC-f** | Forward one word. |
| **CTRL-p** | Up one line. |
| **CTRL-n** | Down one line. |
| **CTRL-a** | Start of line. |
| **CTRL-e** | End of line. |
| **ESC-<** | Go to buffer beginning. |
| **ESC->** | Go to end of buffer. |

**Screen Movement**

| Key Sequence | Function |
| --- | --- |
| **CTRL-v** | Scroll to next screen. |
| **ESC-v** | Scroll to previous screen. |
| **CTRL-x <** | Scroll left (only if text is wrapping). |
| **CTRL-x >** | Scroll right. |

**Files**

| Key Sequence | Function |
| --- | --- |
| **CTRL-x CTRL-r** | Reads a file into Emacs in single window. |
| **CTRL-x CTRL-f** | Reads a file into Emacs in split window. |
| **CTRL-x CTRL-s** | Saves a file to disk. |

| | |
|---|---|
| **CTRL-x i** | Insert contents of another file into this buffer at cursor position. |
| **CTRL-x CTRL-w** | Write buffer to a specified file. |

**Incremental Searches**

| Key Sequence | Function |
|---|---|
| **CTRL-s** | Search forward. Use a second **CTRL-s** to search for the next occurrence. |
| **CTRL-g** | Abort current search. |
| **CTRL-r** | Search backward. |
| **CTRL-ESC-s** | Regular expression search. |
| **ESC** | Exit incremental search. |

**Deleting**

| Key Sequence | Function |
|---|---|
| **DEL** | Delete previous character. |
| **CTRL-d** | Delete next character. |
| **ESC-d** | Delete next word. |
| **ESC-DEL** | Delete previous word. |
| **CTRL-k** | Delete forward (kill) to end of line. |
| **ESC-0 CTRL-k** | Delete backward (kill) to beginning of line. |

**Changing Case**

| Key Sequence | Function |
|---|---|
| **ESC-u** | Make word uppercase. |
| **ESC-l** | Make word lowercase. |
| **ESC-c** | Capitalize word. |

**Split Screen**

| Key Sequence | Function |
|---|---|
| **CTRL-x 2** | Split screen in two windows. |
| **CTRL-x CTRL-f** | Read a file into the other window. |
| **CTRL-x CTRL-o** | Move the cursor the other window. |
| **CTRL-x 1** | Restore the screen to a single window. |

**Cut and Paste**

| Key Sequence | Function |
|---|---|
| **CTRL-SPACEBAR** | Set a mark. |
| **CTRL-x w** | Delete the text region between the mark and the cursor. |
| **CTRL-y** | Re-paste the deleted text region at cursor. |
| **ESC-w** | Extract (without deleting) the region between the mark and the cursor. |
| **CTRL-y** | Paste the extracted region at the cursor. |

**Control Characters**

| Key Sequence | Function |
|---|---|
| **CTRL-q** | Literal quote of the following character. Control keys may be typed in by preceeding them with a CTRL-q. |

**Getting Additional Information**

| Key Sequence | Function |
|---|---|
| **CTRL-h** | Enter the help system. (Use the RETURN key to exit.) |
| **CTRL-x 1** | Get rid of the help window. |
| **ESC CTRL-v** | Scroll through the help window. |
| **CTRL-h c** | Show the function a key runs. |
| **CTRL-h f** | Describe a function. |
| **CTRL-h m** | Get mode specific information. |

# Table of Contents


**7. Site Preparation and Customer Support ....................................7-1**

Site Preparation ........................................ 7-1
- Space ........................................ 7-2
- Location ........................................ 7-2
- Environmental Support ........................................ 7-2
- Power ........................................ 7-3
- Temperature and Humidity ........................................ 7-4
- Fire and Safety Precautions ........................................ 7-5
- Air Quality ........................................ 7-5
- Work Table ........................................ 7-6
- Floor Coverings ........................................ 7-6
- Cabling ........................................ 7-6
- Networking ........................................ 7-7
- Partitions ........................................ 7-7
- Acoustics ........................................ 7-7
- Lighting ........................................ 7-7
- Vibration ........................................ 7-7
- EMI ........................................ 7-8
- Static Electricity and ESD ........................................ 7-8
- Corrosive Environments ........................................ 7-9
- Altitude ........................................ 7-9

Customer Support ........................................ 7-9
- Hardware Support Plans ........................................ 7-9
- *Class A Hardware Service* Features ........................................ 7-9
- *Class B Hardware Service* Features ........................................ 7-10
- Pricing Zones ........................................ 7-10
- Software Support Plans ........................................ 7-11
- *Technical Phone Support Service* Features ........................................ 7-11
- *Software and Documentation Update Service* Features ........................................ 7-11
- Warranty ........................................ 7-11
- Educational Programs ........................................ 7-11

Safety Precautions ........................................ 7-12
- General ........................................ 7-12
- Cabinet ........................................ 7-12
- RDC ........................................ 7-12

Preventive Maintenance ........................................................................7-12
- Monitor Cleaning Instructions ........................................................7-13
- Care of the Keyboard ........................................................................7-13
- Care of the Mouse ............................................................................7-13
- Care of the Tape Unit ........................................................................7-13
- Filter Cleaning ..................................................................................7-14
- Shutdown and Powering Off ............................................................7-14

Predelivery Planning and Installation ....................................................7-15
- Delivery Constraints ..........................................................................7-15
- Equipment Packaging and Handling ................................................7-15
- Receiving Procedure ..........................................................................7-16
- Installation Procedure ........................................................................7-17
- Customer Checklist ............................................................................7-17

Frequently Asked Questions and Answers ..............................................7-17

Evans & Sutherland Field Service Organization ....................................7-19

Who to Call ..............................................................................................7-19

# 7. Site Preparation and Customer Support

## Introduction

This chapter contains site preparation, customer support, and preventive maintenance information that you received prior to the installation of your ESV Workstation. It is included for your reference.

The general information presented here should be tailored to meet individual situations. Please contact your Evans & Sutherland Service Center if you have any questions.

## Site Preparation

Adequate site planning and preparation eases the installation process and produces efficient system operation. Site planning requirements vary greatly from site to site. The location and environmental aspects of your system are as significant as the equipment itself. The system could prove to be unusable if it is placed in an awkward or inadequately supported location. Space and location are the primary considerations for site selection. Table 7-1 shows the component dimensions, and table 7-2 shows the cable lengths.

**Table 7-1. Component dimensions**

| Component | Length in (cm) | Width in (cm) | Height in (cm) | Weight lb (kg) |
|---|---|---|---|---|
| Large Cabinet | 30.0 (76.2) | 17.0 (43.2) | 25.5 (64.8) | 200.0 (90.7) |
| Small Cabinet | 30.0 (76.2) | 11.0 (27.9) | 25.5 (64.8) | 150.0 (68.0) |
| Monitor | 20.8 (52.8) | 19.4 (49.3) | 17.5 (44.5) | 68.5 (31.1) |
| RDC | 15.0 (38.1) | 15.0 (38.1) | 3.0 (7.6) | 12.0 (5.4) |
| Keyboard | 8.0 (20.3) | 19.0 (48.3) | 1.6 (4.1) | 5.0 (2.3) |
| Mouse | 3.8 (9.7) | 2.8 (7.1) | 1.0 (2.5) | 0.5 (0.2) |
| Control Dials | 9.0 (22.9) | 12.0 (30.5) | 3.8 (9.7) | 9.5 (4.3) |
| Function Buttons | 9.6 (24.4) | 9.1 (23.1) | 1.6 (4.1) | 1.8 (0.8) |
| Spaceball | 12.6 (32.0) | 6.5 (16.5) | 5.3 (13.5) | 3.3 (1.5) |
| 6x9 Tablet | 9.4 (23.9) | 12.9 (32.8) | 0.3 (0.7) | 2.1 (1.0) |
| 12x12 Tablet | 17.0 (43.2) | 17.0 (43.2) | 1.3 (3.3) | 5.5 (2.5) |
| 15x15 Tablet | 20.8 (52.8) | 20.7 (52.6) | 3.0 (7.6) | 12.2 (5.5) |
| 18x25 Tablet | 24.1 (61.2) | 30.7 (78.0) | 3.1 (7.9) | 23.2 (10.5) |

**Table 7-2. Cable lengths**

| Cable | Length - feet (meters) |
|---|---|
| Monitor to Cabinet | 10, 20, 45, 100, 150 (3.1, 6.1, 13.7) |
| RDC to Cabinet | 10, 20, 45, 100, 150 (3.1, 6.1, 13.7) |
| Cabinet to AC Power | 8 (2.4) |
| RDC to AC Power | 8 (2.4) |
| Monitor to AC Power (RDC) | 3 (1.0) |

### Space

The ESV Workstation should be positioned out of direct sunlight and away from all sources of heat, including central heating vents, with a minimum of 2 inches (5 cm) of clearance around the cabinet to allow for air flow and cabling. At least two sides of the cabinet must be left exposed to allow for adequate air flow. The back of the monitor must have at least 6-1/2 inches (16 cm) of clearance for cabling.

The actual floor space required will depend on the system itself, the length-to-width ratio of the area, and the locations of walls, partitions, windows, and doors. To determine the exact area your system requires, prepare a scaled layout that includes all features of the site location. The area allotted should provide for the following: future expansion of the system, storage of related materials, convenient system operation, and easy access for service and maintenance.

### Location

Locate your ESV Workstation site near work-related areas for efficient operation. The location of the site also depends on existing or planned facilities at the site. The location must do the following:

- Provide adequate AC power,
- Conform to environmental requirements,
- Conform to safety and fire regulations,
- Provide easy access for equipment delivery and installation, and
- Provide for the flow of work in the most efficient manner possible with respect to such considerations as related areas, human factors, storage, and noise isolation.

### Environmental Support

When selecting your site, you must plan for adequate power and for environmental support factors, such as temperature requirements and adequate air quality.

## Power

You need to provide additional power for any other equipment that will be operated in the area, such as test equipment and calculators.

- For the large cabinet, the USA wall receptacle should be a wall plug, NEMA 5-20R 20A or equivalent.
- For the small cabinet, the USA wall receptacle should be a wall plug, NEMA 5-15R 15A or equivalent.

For the 220 volt option, the ESV Workstation is supplied with an EE 7/7 ("schuko") 10A cordset. The power specifications for the components are shown in table 7-3.

**Table 7-3. Power specifications**

| Component | Voltage 120/220–240V | Max. Current 120V | Max. Current 220V | Line Freq. (Hz) | Power (Watts) |
|---|---|---|---|---|---|
| Large Cabinet | 120/220-240 | 16 A | 10 A | 60/50 | 1920 |
| Small Cabinet | 120/220-240 | 12 A | 8 A | 60/50 | 1440 |
| Monitor | 120/220-240 | 1.25 A | 0.7 A | 60/50 | 150 |
| RDC | 120/220-240 | 6 A | 3 A | 60/50 | 720 |

Following are general notes for table 7-3:

1) The line voltages shown have a tolerance of +6% to –10%.
2) The power consumption for the RDC is divided as follows: 240 Watts for the RDC and 240 Watts for each of the two convenience receptacles on the back of the RDC.
3) In Japan, ESV systems are designed to operate at 100 V, with a tolerance of +6% to –10%, and 50 Hz.
4) The ESV systems are also designed to operate at 220 to 240 V, with a tolerance of +6% to –10%, and 60 Hz.
5) The cabinet and the monitor run on two-wire-plus-ground circuits. The monitor must share a common electrical ground with its supporting cabinet.

The ESV Workstation requires one dedicated 20 amp service line for the large cabinet and one 15 amp service line for the small cabinet. The system may be damaged if adequate service is not provided. The available supply of AC power must be adequate to handle the power loads represented by the installation of the ESV Workstation as well as any anticipated future loads. The electrical system must conform to applicable national and local codes and ordinances. Check the electrical service prior to system installation to ensure that power levels are within the specified limits.

To ensure proper operation of the ESV Workstation, the following limitations are placed on AC power disturbances:

- A maximum of 10% of nominal power for 0.1 seconds occurring no more than once every 10 seconds,
- Maximum harmonic content of 5% rms, no more than 3% rms for any single harmonic,
- Maximum impulse of 300 V with rise time of 0.1 microseconds or slower, lasting no longer than 10 microseconds for total duration.

Many unconditioned AC service mains exceed these specifications, especially during periods of heavy use and/or electrical disturbances. Ensure that the input power supplied to the ESV Workstation equipment has been adequately conditioned.

### Temperature and Humidity

The best way to provide the proper air temperature is to provide a separate thermostatic control to compensate for the heat dissipated by the ESV Workstation and any other equipment and personnel in the area. The air conditioning system must provide sufficient heating and cooling to maintain the environment within the limits shown in table 7-4.

**Table 7-4. Temperature and humidity limits**

| | Operating | Non-Operating |
|---|---|---|
| Temperature | 50°F to 104°F (10°C to 40°C) | –40°F to 122°F (–40°C to 50°C) |
| Humidity | 20% to 80% (non-condensing) | 10% to 90% (non-condensing) |

In table 7-4, the operating temperature is measured at the air intake vents. The operating temperatures in the table are at sea level. The maximum operating temperature must be derated linearly 1.8°F per 1640 feet (1°C per 500 meters) increase in altitude.

Heat dissipation factors can be calculated by using the values given in table 7-5. These values should be added to any other heat generated by equipment and personnel located in the same room.

**Table 7-5. Heat dissipation factors**

| Component | Heat Dissipated (max.) | |
|---|---|---|
| Large Cabinet | 6566 BTU/hr | (1920 Watts) |
| Small Cabinet | 4925 BTU/hr | (1443 Watts) |
| Monitor | 512 BTU/hr | (150 Watts) |
| RDC | 2462 BTU/hr | (720 Watts) |
| One Person | 400 BTU/hr | (117 Watts) |

### Fire and Safety Precautions

- Existing building fire and safety codes should be adequate for the ESV Workstation installation. However, local experts should be consulted about fire prevention and extinguishing devices.
- Do not install the ESV Workstation near the use, storage, or manufacturing of flammable or explosive material.
- For safety as well as operational reasons, each interconnected piece of equipment must be provided with a properly grounded outlet.
- All power circuits must be adequately protected with fuses or circuit breakers of a suitable size.
- No metal should be exposed on the walking surface of floors.

### Air Quality

The ESV Workstation equipment is designed for use in a clean environment, where air filtration is not always possible. However, the cabinet should be placed away from high traffic areas because the build-up of potential contaminates is more concentrated in these areas. Airborne dust, dirt particles, and smoke can clog intake air filters and cause damage to the hard disk and tape drive.

Dust is usually controlled by normal heating, ventilating, and cooling equipment if adequate filters are used. Keep the system area clean and orderly to lessen the concentration of airborne particles and help maintain system reliability. Where excess dust or airborne particles are present, install an electrostatic filter to prevent damage to the system.

**Caution:** Never place the ESV Workstation in an area containing even small concentrations of corrosive chemicals.

### Work Table

The ESV Workstation monitor and interactive devices should be installed on a table or large desk of heavy construction with a durable, non-glare surface. The minimum recommended surface area is 13 square feet (1.2 square meters), and the recommended height is 29.5 inches (75 cm).

In areas where ergonomic compliance is required, the table's height should be adjustable from approximately 27 inches (68 cm) to 30.5 inches (76 cm). It is also recommended that the chair's height be adjustable.

### Floor Coverings

The most desirable flooring is a raised floor that includes tile-covered panels supported by a grid system of pedestals. These floors simplify installation and provide flexibility for subsequent layout changes or expansion. They also provide an area through which cables connecting various components of the system can be routed and kept out of the way.

If you are unable to use a raised floor, most floor surfaces are adequate for installation, with the following considerations:

- For any high-grade industrial carpeting with short, closed-loop piles, you should use minimal or no padding. The carpet should have good anti-static properties and/or a low surface resistivity. Shag rug, deep pile and other such carpets are not recommended and can cause serious operational difficulties. These rugs have loose fibers and collect dust particles which can clog cooling inlet filters and generate unacceptable levels of static electricity.
- Most tiles provide a suitable surface, however, specific attention should be given to underlay. There is a tendency for some tiles to build up static charge. This can be minimized by proper application of low-resistivity sealer and polish. This application will need to be repeated at appropriate intervals.
- Wax is not recommended as a protective coating for floors in a computer area as it tends to build up surface resistivity and increase static charge.
- Other surfaces should be evaluated for surface resistivity, ease of cleaning and resistance to decomposition, durability, cost, and appearance.

### Cabling

Conduits, cable ramps, and any necessary alterations must be implemented prior to the system delivery. All customer-supplied cables must be shielded.

### Networking

If you plan to make your ESV Workstation part of an Ethernet network, you must have an Ethernet transceiver connection to your Ethernet backbone ready when the system is installed.

### Partitions

Floor-to-ceiling partitions are an effective way of controlling noise and dust. Partitions must be positioned to avoid blocking air flow to the equipment and to allow for equipment access and cabling restraints.

### Acoustics

The ESV Workstation is designed to operate with a minimum amount of noise. Cooling fans within the cabinets are a possible source of audible sound, but in most environments ambient sound will be louder than the ESV Workstation.

If several ESV Workstations are to be operated in close proximity, acoustical damping of the ceiling, floors, and walls might be considered.

### Lighting

The ESV Workstation is designed to operate in a normal lighting environment. The optimum lighting for a graphics CRT monitor should be subdued, indirect incandescent lighting. To reduce operator fatigue, avoid lighting that produces glare on the face of the CRT.

### Vibration

Vibration can cause slow degradation of mechanical parts and contacts. It should be avoided whenever possible. In cases where structure-borne vibration is negligible, no problems should arise. If there is any unusual or prolonged vibration anticipated, consult an Evans & Sutherland Technical Support representative.

### EMI

The ESV Workstation has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the ESV Workstation is operated in a commercial environment. The ESV Workstation generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the ESV Workstation documentation, may cause harmful interference to radio communications. Operation of the ESV Workstation in a residential area is likely to cause harmful interference, in which case users will be required to correct the interference at their own expense.

EMI sources close to computer systems can affect their operation. It is difficult to predict whether or not problems will arise at a particular site. Some common sources of EMI that have been known to cause failures are:

| | | |
|---|---|---|
| appliances | industrial machines | relay contactors |
| arc welders | magnetic devices | static electricity |
| broadcast stations | mobile communications | thunderstorms |
| dielectric heaters | office machines | ultrasonic cleaners |
| fluorescent lights | power tools | vehicle ignitions |
| high voltage power lines | radar | |

Consult an Evans & Sutherland Technical Support representative if potential problems exist at a particular site.

### Static Electricity and ESD

Static electricity is the result of physical action. Vibration, friction, and separation of materials are common static generators. People and furniture are the most common static storage collectors. Static may be generated by walking, rising from chairs, moving objects, or pushing vehicles with nonconductive wheels. Voltages of 16 KV have been measured on plastic-covered metal desk chairs as a result of a person standing up. This often occurs at low relative humidities (0 to 20%).

Do not locate the ESV Workstation in an area where potentially large charges of static electricity may gather. For information on floor conductivity, see *IEEE Standard 142-1972*. Although the ESV Workstation has been engineered to resist the harmful effects of ESD, every effort should be made to reduce the possibility of ESD directly to the equipment.

### Corrosive Environments

Operation of the ESV Workstation in a corrosive environment results in damage to electronic components and circuitry. Some common corrosive substances are:

| | | |
|---|---|---|
| ammonia | nitrates | sodium chloride (table salt) |
| hydrocarbons | nitrogen oxides | sulfur dioxide |
| hydrogen sulfide | ozone | |

Consult an Evans & Sutherland Technical Support representative if any of these contaminants are present in the intended environment.

### Altitude

System operation at high altitudes may be affected by low air density. Heat dissipation problems may occur at altitudes greater than 7,000 feet (2,000 meters). If high altitude operation is anticipated, additional air flow around the cabinet should be provided.

## Customer Support

### Hardware Support Plans

Two hardware support plans are available from Evans & Sutherland.

- *Class A Hardware Service* offers the most complete hardware service plan. It is designed for customers who require the best response time for system repairs.
- *Class B Hardware Service* offers a reduced service, customer-assisted repair service plan with module exchange.

#### *Class A Hardware Service* Features

- Guaranteed next-day response from a qualified Field Service Engineer, Monday through Friday, 8:00 a.m. to 5:00 p.m. (local time).
- On-site repair by Field Service Engineer, with travel, parts, and labor included.
- System installation cost is included if a *Class A Hardware Service* contract is purchased with the ESV Workstation.
- Guaranteed customer site visits at maximum intervals of 180 days.

#### *Class B Hardware Service* Features

- Modules are shipped to the customer within one business day. Evans & Sutherland pays for the freight charges both ways.
- Customers may purchase on-site repair by a Field Service Engineer at a fixed price.

- DST (Diagnostic System Test) right-to-use license, DST updates, and DST phone support are included.

**Prerequisites**

- Customers must complete an Evans & Sutherland training class.
- Customer contacts are limited to individuals who have completed an Evans & Sutherland training class (no more than three contacts).

**Pricing Zones**

*Class A Hardware Service* is priced by service zone. The optional *Class B Hardware Service* on-site repair is also priced by service zone. Distances are determined from the center of the metropolitan city nearest an Evans & Sutherland Service Center. Following are the three zones:

- Zone 1 is within a 100 mile radius from the Evans & Sutherland Service Center.
- Zone 2 is within a 200 mile radius from the Evans & Sutherland Service Center.
- Zone 3 is everywhere else. All locations in Canada are zone 3.

Your Sales Representative will assist you in determining your zone.

### Software Support Plans

Two software support plans are available from Evans & Sutherland.

- *Technical Phone Support Service* gives customers toll-free telephone access to Evans & Sutherland Technical Support personnel.
- *Software and Documentation Update Service* enables customers to maintain their system software and documentation at the most current revision levels.

### *Technical Phone Support Service* Features

- Up to three customer contacts may call the Dispatch Hot Line for assistance in resolving problems with the operation of the ESV Workstation. A one hour call-back is guaranteed during Dispatch Hot Line hours of 7:00 a.m. to 5:30 p.m. (Mountain Time).
- Customers are provided with periodic updates to both the ESV Workstation software and documentation.
- Update service for software options, and additional copies of media and documentation, are provided at additional cost to the customer.
- Telephone support for software options is provided at additional cost to the customer.
- All systems at a customer site are required to be at the same level of support for the basic system software.

### *Software and Documentation Update Service* Features

- Customers are provided with periodic updates to both the ESV Workstation software and documentation.
- Update service for software options, and additional copies of media and documentation, are provided at additional cost to the customer.
- All systems at a customer site are required to be at the same level of support.

### Warranty

The warranty period for the ESV Workstation is 90 days after installation. Warranty service is at the *Class A Hardware Service* and *Technical Phone Support Service* levels. After the warranty period has expired, service is delivered under the terms specified on the customer's service contract.

### Educational Programs

Evans & Sutherland Field Service offers a 25% discount on all service plans for educational institutions.

## Safety Precautions

### General

1) Only Evans & Sutherland Authorized Personnel are permitted to install and service the ESV Workstation. Customers should not attempt to service any equipment, including, but not limited to, the front and back access panels on the cabinet, the RDC, and the monitor.
2) The ESV Workstation cabinet and RDC are designed to meet UL Standard 1950. Formal approval is in progress.
3) The ESV Workstation cabinet and RDC have the GS safety mark. The video card is part of the cabinet.
4) Changes or modifications to the ESV Workstation that are not expressly approved by Evans & Sutherland may void the customer's authority to operate the equipment.

### Cabinet

1) The lower front panel is not an operator accessible area and can be accessed by Evans & Sutherland Authorized Personnel only.
2) The rear access panel is not an operator accessible area and can be accessed by Evans & Sutherland Authorized Personnel only.

### RDC

1) The convenience receptacles on the back of the RDC are still energized when the power switch on the front is set to the OFF position.
2) The inside of the RDC is not an operator accessible area and can be accessed by Evans & Sutherland Authorized Personnel only.

## Preventive Maintenance

Most of the ESV Workstation preventive maintenance will be performed by your Evans & Sutherland Field Service Engineer during periodic site visits. However, you should keep the following tips in mind and make periodic checks.

- Performance and dependability of the ESV Workstation can be affected by user neglect or a poor choice of operating site.
- Cleaning intervals should be based on the amount of ESV Workstation use and the quality of the operating environment.
- Check all cables for visible damage and wear.

- Check connectors to ensure that the cables are mounted tightly, that retaining screws are tight, and that there is no strain on the connector/cable junctions.
- Make sure your ESV Workstation receives proper air ventilation.
- Always follow the proper procedure for shutting down and powering off your ESV Workstation.

### Monitor Cleaning Instructions

- Always unplug the monitor before cleaning.
- Wipe the screen and cabinet front and sides with a soft cloth.
- If the screen requires more than dusting, apply a household window cleaner to a soft cloth to clean the monitor screen.

**Caution:** Do not use benzene, thinner or any volatile substances to clean the unit as the finish may be permanently marked. Never leave the unit in contact with rubber or vinyl for an extended period of time.

### Care of the Keyboard

The keyboard is a rugged unit that should provide trouble-free service. It may be damaged if liquids are spilled on it. If an accident should happen and liquid is spilled on the keyboard, let it dry and then try its operation. If the keyboard doesn't operate properly, contact Evans & Sutherland for repair or replacement.

### Care of the Mouse

The optical mouse requires little maintenance. The mouse pad should be kept clean and dry and should be protected from scratches and dents. Clean the pad occasionally with a damp cloth. If the pad should get wet, dry it thoroughly before using. The mouse itself requires no maintenance, but can be cleaned as required with a damp cloth.

### Care of the Tape Unit

Optimal recording and readback performance of the tape unit requires proper head cleaning at frequent intervals. The manufacturer's recommended equipment for head cleaning is the Tandberg Data "TDC Cleaning Cartridge Kit." Cleaning kits other than the Tandberg have also proven satisfactory. The kits most suitable for the ESV Workstation tape unit are designed to operate by capstan motion.

**Caution:** Do not use any sharp objects when cleaning the head. Even small scratches may damage the head permanently.

The following guidelines can be used to determine cleaning intervals:

| Tape Usage | Cleaning Interval |
|---|---|
| 8 hours per day | Daily |
| Daily | Weekly |
| Weekly | Monthly |

Always clean the head immediately after using a new cartridge.

Performance of the tape unit depends on the quality of the medium used. For writing, the ESV Workstation tape drives need DC600XTD, DC6150 tapes, or equivalent, to work properly. Do not use worn or audibly noisy cartridges. Cartridges which repeatedly require rewriting should be discarded.

Tape media are very susceptible to moisture. If exposed to a high humidity environment, it may take several days to bring a cartridge back to a normal humidity condition. Running high humidity tapes over a long period of time may severely reduce the life of the tape drive head. If in doubt, let the cartridge dry out in a normal humidity environment (less than 50-65% relative humidity at 20°C) for three to four days prior to use.

### Filter Cleaning

The black "sponge-like" material visible through the slots of cabinet side panels are the air filters. The black filter material will slowly turn grey from the dust collected. This dust should be periodically removed from the filters so that the fans can maintain proper operating temperature inside the cabinet. The filters can be cleaned by vacuuming the exposed areas of the filters.

**Caution:** The cabinet power must be shut off before vacuuming the filters.

### Shutdown and Powering Off

To avoid corruption of the resident file systems, the ESV Workstation should never be turned off or unplugged without following this shutdown and powering off procedure:

- Login as **`root`**.
- **`cd /`**
- **`/etc/shutdown -y -i0 -g60 <cr>`**

  "60" can be replaced with any grace period (in seconds) you decide to allow users to log off prior to shutdown.
- Wait for the >> prompt.
- It is now safe to remove power from the ESV Workstation.

## Predelivery Planning and Installation

Predelivery planning is essential for smooth installation and acceptance of your ESV Workstation. It is important that you prepare a detailed schedule of installation activity as soon as possible after the equipment has been ordered and the site selected and prepared.

Once the installation has taken place, you are responsible for the disposal of the packaging material. Preparations should be made in advance to remove the empty packaging material from the installation site when installation is complete.

### Delivery Constraints

The largest box fits through a 36-inch (92-cm.) wide doorway. Ensure the route the equipment is to travel from the receiving area to the installation site allows the equipment to move freely. The packaged equipment must be able to fit through any halls, doorways, around any bends, or in elevators.

### Equipment Packaging and Handling

It is your responsibility to transport the system from its unloading site to the actual installation site. This should be done prior to the system installation date. Do not subject the equipment boxes to any hard bumps or shocks. Keep the boxes in a vertical position as indicated on the box surface. Do not open them.

For shipment, the ESV Workstation and peripherals are packed in a reinforced cardboard box which is attached to a pallet. The monitor box is banded to the top of the ESV Workstation box. The dimensions of the shipping box are as follows:

- English units: 35.0 in (length) x 23.5 in (width) x 34.0 in (height)
- Metric units: 89 cm (length) x 60 cm (width) x 86 cm (height)

Evans & Sutherland has adopted the *shockwatch* label and the *tip-and-tell* label as a way to safeguard the ESV Workstation during shipment. These labels are simple and effective warning devices that tell you if a shipment has been roughly handled.

The *shockwatch* and *tip-and-tell* labels help identify responsibility for products damaged during shipping. Since mishandling the product activates the devices, the presence of these labels encourages careful handling for the ESV Workstation. If the product has been mishandled, the labels indicate the following: the *shockwatch* label indicator in the center of the label turns bright red and cannot be reset; and the *tip-and-tell* label indicator turns blue and cannot be reset.

**Receiving Procedure**

You must follow this procedure when the ESV Workstation is delivered to your site:

1) Upon receipt of your shipment, note the color of the *shockwatch* and *tip-and-tell* indicators. If you receive more than one carton, check all of the labels. Do not refuse shipment.
2) If any of the labels have been triggered, note the cartons that have been mishandled on the delivery ticket and request that the carrier's driver sign a receipt acknowledging that the labels have been activated.
3) If the product is visibly damaged, note this on the delivery ticket receipt and contact Evans & Sutherland immediately by calling:
   - In the USA, 800-582-4375,
   - In Europe, your local sales office.

All boxes, with the exception of boxes containing documentation, must be opened and unpacked only by an authorized Evans & Sutherland Field Service Engineer. Unpacking by unauthorized persons may void the warranty on this equipment.

If you must open the boxes to move the system to the installation site, please contact the Evans & Sutherland Service Center and request authorization to open the boxes.

Should you unpack or inspect any of the equipment without an authorized Evans & Sutherland Field Service Engineer present, or without authorization from Evans & Sutherland, you assume all responsibility for any damage or shortage claims with the carrier.

### Installation Procedure

1) After the ESV Workstation is shipped, a representative from the Evans & Sutherland Field Service Department will contact the customer to verify delivery.
2) The customer will call the Field Service Department to initiate the installation.
3) A representative from the Field Service Department will contact the customer to verify that the site preparations have been completed and adequate electrical service is available. An installation appointment will be scheduled with the customer.
4) The Evans & Sutherland Field Service Engineer will arrive at the customer site and install the workstation.

### Customer Checklist

- Have you selected an appropriate site for your ESV Workstation?
- Does the selected site have adequate power and environmental support?
- Have the work facilities for the selected site been prepared?
- Have all of the predelivery conditions been met?

If the answer to all of the above questions is yes, then you are ready to schedule the installation of your ESV Workstation. To schedule an installation, call:

- In the USA, 800-582-4375,
- In Europe, your local sales office.

## Frequently Asked Questions and Answers

**Q: How many versions of the system software are supported?**

Evans & Sutherland supports the current version and previous version of the system software.

**Q: What is the time from the report of a serious problem until a fix is provided?**

Evans & Sutherland policy is to resolve a work stoppage situation as soon as possible. The resolution may be a work around, a patch, or a new version. Evans & Sutherland has an escalation procedure which ensures that open problems are resolved.

**Q: Can I elect to not upgrade to a new release until it is convenient for me?**

Yes, that is why we support the current version and the previous version of the system software.

**Q: What is your policy regarding upward and downward compatibility of releases?**

Evans & Sutherland tries to make system software releases as compatible as possible with previous system software and applications. However, providing new technology and fixing problems means that a new software release sometimes not be completely compatible. If possible, we will try to limit the incompatibility to a requirement to recompile and relink.

**Q: Will the same Field Service Engineer come to my site for every visit?**

Generally, our policy is to assign a primary Field Service Engineer and a back-up to each customer site. This allows the Field Service Engineer to become well acquainted with the customers needs and communicate those needs to the rest of Evans & Sutherland.

**Q: Are there any restrictions as to the number of support calls or who may call?**

The number of customer contacts who may call is limited to three. The number of calls to the Evans & Sutherland Dispatch Hot Line is unlimited.

**Q: Is there a newsletter available to provide the latest information?**

Yes, Evans & Sutherland is planning to publish a newsletter at regular intervals.

**Q: Is there any provision for on-site help?**

Yes, at our option we will send support personnel on site to resolve problems. Also, we will be happy to discuss consulting contracts to provide help in applying the system to a customers particular needs.

**Q: If I elect to not take a contract, what are my options?**

We will provide requested (time and materials) service to all owners of Evans & Sutherland equipment on an as-available basis. We will be happy to discuss contracts for systems that have not been on service for a period of time, subject to a review of the systems status. If the system is not supportable in its present state, the customer will be required to pay requested rates to restore the system to serviceable condition.

**Q: How long will your company support your product?**

Evans & Sutherland will send a letter to all customers announcing the fact that a product is no longer available for new equipment purchase and listing the support period. Generally, due to GSA requirements, that support period will be at least seven years.

**Q: Can I buy *Technical Phone Support Service* for the UNIX operating system and not buy it for software options?**

Yes, but we do not recommend it.

**Q: What is your policy regarding third-party hardware installed in an ESV Workstation?**

Customers who install third-party hardware in a ESV Workstation should do everything they can to assure themselves that a system problem is not the fault of the third-party hardware. If a Field Service Engineer makes a site visit, and it is determined that the problem is with the third-party hardware, the customer will be billed for the expenses associated with the visit.

## Evans & Sutherland Field Service Organization

| | |
|---|---|
| Corporate Headquarters, Salt Lake City, Utah | 801-582-5847 |
| Dispatch Hot Line | 800-582-4375 |
| Gordon Scott, Director of Field Service | 801-582-5847 |
| Maurice Smith, Technical Support Manager | 801-582-5847 |
| John Wallace, Eastern Field Service Region Manager | 513-692-8858 |
| Jim Blatz, Western Field Service Region Manager | 916-448-0355 |

## Who to Call

- For normal Field Service business, call the Dispatch Hot Line.
- To schedule an installation, call the Dispatch Hot Line.
- For pricing information, call your local Sales Representative or Corporate Headquarters.
- If you feel you are not being well served, call any person listed above.

**Dispatch Hot Line**

**800-582-4375**

# 8. Porting Guide

# Table of Contents


**8. Porting Guide ........ 8-1**

Introduction ........ 8-1
- Assembly Language Programs ........ 8-1
- Making a Program Portable ........ 8-1
- Information You Need to Know ........ 8-1
- Porting a Program ........ 8-3

Trouble Shooting ........ 8-5

ES/os Considerations ........ 8-10
- ES/os ........ 8-10
- Porting from BSD-Derived Systems ........ 8-10
- Porting from System V-Derived Systems ........ 8-14
- Dereferencing **nil** Pointers ........ 8-20
- Where Text and Data Lie in Memory ........ 8-20
- Changing Internal Compiler Table Size ........ 8-21
- Porting from Other Operating Systems ........ 8-21

Hardware-Related Considerations ........ 8-22
- Floating Point Arithmetic ........ 8-22
- Endianness ........ 8-25
- Alignment ........ 8-26
- Uninitialized Variables ........ 8-27

Undefined Language Elements ........ 8-28
- The Value of **nil** ........ 8-28
- Order of Evaluation ........ 8-28
- Inter-Language Interfaces ........ 8-29

C Programming Language ........ 8-29
- Supported **cc** Options ........ 8-30
- Supported **ld** Options ........ 8-31
- Using the C Preprocessor ........ 8-33
- Using the **lint** Program Checker ........ 8-34
- Memory Allocation ........ 8-35
- Signed **chars** ........ 8-36
- Bitfields ........ 8-36
- **short**, **int**, and **long** Variables ........ 8-36
- Leading "_" ........ 8-36
- **varargs** ........ 8-37

**typedef** Names ........ 8-38
Functions Returning **float** ........ 8-38
Casting ........ 8-38
Dollar Sign in Identifier Names ........ 8-39
Additional Keywords ........ 8-39
Unsigned Pointers ........ 8-39
Pascal Programs ........ 8-39
Supported **cc** Options ........ 8-40
Supported **ld** Options ........ 8-41
Runtime Checking ........ 8-43
Pascal Dynamic Memory Allocation ........ 8-43
FORTRAN Programming Language ........ 8-45
Supported **f77** Options ........ 8-45
Supported **ld** Options ........ 8-48
Static Versus Automatic Allocation ........ 8-49
Retention of Data ........ 8-50
Variable Length Argument Lists ........ 8-51
Runtime Checking ........ 8-51
Alignment of Data Types ........ 8-51
Inconsistent Common Block Sizes ........ 8-53
Multiple Initializations of Common Blockdata ........ 8-54
Endianness and **integer*2** ........ 8-55
FORTRAN and C Interface ........ 8-56
Names ........ 8-56
Invocations ........ 8-56
Arguments ........ 8-57
Array Handling ........ 8-59
Accessing Common Blocks of Data ........ 8-60
ES/os Compiler Components ........ 8-61
Debugging ........ 8-61
Program Checking ........ 8-62
Optimization ........ 8-65
The Link Editor ........ 8-66
Helpful Hints ........ 8-70

# 8. Porting Guide

## Introduction

This chapter describes a process to follow when you are porting programs to the ES/os environment.

### Assembly Language Programs

This chapter is a compilation of information that you need to port a C, Pascal, or FORTRAN program to an ESV Workstation. It does not describe how to port assembly language programs. If you are thinking about porting an assembly program and your only reason for coding in assembly language is speed, seriously consider recoding in a high-level language. The ES/os compiler reduces the need for assembly language programming because it produces highly efficient machine language code for all supported high-level languages.

### Making a Program Portable

You can make the task of porting programs easier by following the guidelines listed below when you create your program.

- Avoid breaking the rules of the source language and avoid using its non-standard features or extensions.
- Avoid using source language that is machine dependent.
- Avoid relying on anything that is operating system dependent.
- When you have to do any of the things listed above, encapsulate them in modules marked "system dependent," use conditional compilation **#ifdef** (a **cpp** function) statements around them, and explain the situation with plenty of comments.

### Information You Need to Know

Before you undertake a porting project, look over the following check list. Where applicable, cross-references are given for detailed information. Make sure that you are aware of each topic listed below to save time when porting your program.

**All programming languages:**

- How ES/os **Makefiles** work ("Overview" section).
- Floating point differences ("Hardware-Related Considerations" section).
- Differences in the address space organization ("Hardware-Related Considerations" section).
- How the **#ifdef** conditional works ("C Programming Language" section).

**C programming language:**

- How to use the **lint** program ("Using the **lint** Program Checker" subsection of the "C Programming Language" section).
- How to use variable arguments ("**varargs**" subsection of the "C Programming Language" section).
- How to determine which **#defines** are appropriate ("Using the C Preprocessor" subsection of the "C Programming Language" section).
- Characters unsigned by default ("Signed **chars**" subsection of the "C Programming Language" section).
- The different compiler options, especially **-std, -std1, -std2, -proto, -systype.**

**Pascal programming language:**

- About the precision of type **real** ("Alignment" subsection of the "Hardware-Related Considerations" section).
- How the ESV Workstation's single compilation process differs from yours.
- How memory allocation works ("Pascal Dynamic Memory Allocation" subsection of the "Pascal Programs" section).
- No two Pascals are the same, because the language has never been adequately standardized and the basic Pascal package is limited without adding extensions.

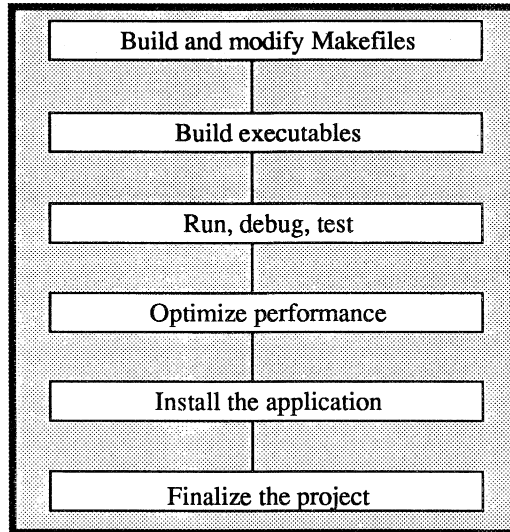
**FORTRAN programming language:**

- How static variables differ from automatic variables ("FORTRAN Programming Language" section).
- How the size of your machine's double precision may differ from ES/os.
- FORTRAN backslash escape sequences.
- FORTRAN alignment options.

## Porting a Program

The following figure shows the major steps involved in porting an application to an ESV Workstation.

The following sections describe each of the steps below in detail.




### Build/Modify **Makefiles**

Large applications are accompanied by UNIX **Makefiles**, which contain the commands that build an executable program and which are processed by the ES/os **make** facility. This facility provides a method for maintaining up-to-date versions of programs that consist of a number of files that may be generated in a variety of ways. The **Makefile** is the description file through which the **make(1)** command keeps track of the commands that create files and the relationship between files. Whenever a change is made in any of the files that make up a program, the **make** command creates the finished program by recompiling only those portions directly or indirectly affected by the change.

For more information on **Makefiles**, refer to the **make(1)** manual page and chapter 13 in the MIPS *Languages Programmer's Guide.*

Before you can build your application, you need to modify the following parts of your **Makefile**:

- Include the path names for your source program and the include files that it uses.
- Include the path names for the link libraries. See the **Intro(3)** manual page for a list of ES/os supported libraries.

- Include a path name for the directory where the application is to be accessed if the **Makefile** has an **install** target.
- Add compilation and link editor flags. See **cc(1)**, **f77(1)**, **pc(1)**, and **ld(1)** manual pages.

### Build Executables

To obtain a debugging version of your program that is not stripped or optimized:

1) Build the executable programs for the application by running the **Makefile** to compile and link the programs. Obtaining the correct compiler option settings may involve modifying the **Makefile** several times.
2) Compile and link using the **-g** debugger option for full symbolic debugging.

### Run the Application

When you run your application, more errors may occur. The trouble shooting guide in the "Trouble Shooting" section should help solve some of the runtime errors that you receive. If your program still has errors, then use debugging tool such as **dbx(1)**. Also, refer to the rest of the sections as applicable, for additional information on possible causes of errors.

Commercial applications usually include a test suite. This is the time to run the tests. If no tests are provided, you can write and automate your own tests with shell scripts **sh(1)** or **csh(1)**.

### Optimize Performance

Once your application is debugged, tested, and working, then you should recompile the final version with the proper optimization level and link edit flags. For information on optimization see chapter 4 in the MIPS *Language Programmer's Guide*, and the **cc** and **f77** manual reference pages.

All tests should be rerun at this point to further reduce the chance that machine, language, or operating system dependencies have slipped through. If the application fails unexpectedly at this point, you probably still have machine or language dependencies that the optimizer did not detect. If this is the case, isolate the area causing the problem and repair it.

Refer to chapter 4 of the MIPS *Language Programmer's Guide* for a description of optimization techniques for your application.

You may wish to profile your code (see **prof(1)** and **pixie(1)**) which will show you where the application is spending most of its time. By fine tuning the code in these areas, additional performance gains can be made. Another useful tuning tool is **cord(1)**, which helps you improve cache performance.

**Install the Application**

Once your application is tested and optimized, install it in the proper directory. You can install the program either by hand, or with the **Makefile install** target.

**Finalize the Application**

The last step is probably the most tedious, but also the most important. If the user interface has changed at all, it is important to document the changes, not only in the code itself, but in the documentation. This is also the time to finalize the source code control system to manage the code. See **sccs(1)** and **rcs(1)**.

## Trouble Shooting

This section is a composite of information taken from all of the sections in this chapter. It contains tables that summarize many problems, and their solutions, that you may encounter when porting a program.

Each of the five columns contains a characteristic of the error. The column headings and a description of the information they contain is given below:

- Column 1 – When or how did the error manifest itself.

  *Error manifested itself during: B(build), C(compile), L(Link Edit), E(execution), or O(incorrect output).

- Column 2 – The source language(s) of the program most likely to create the error.

  **Can appear in which languages: A (all), F (FORTRAN), P (Pascal), and C (C Language).

- Column 3 – Symptom (General description of the error.)
- Column 4 – Possible problem source.
- Column 5 – Action (Recommended action to correct the error. Often, this is a cross-reference to another section in this chapter.)

| * | ** | Symptom | Possible Problem Source | Action |
|---|---|---|---|---|
| L | A | Link edit fails | **gp** area overflow | Specify link editor **-G num** option. See "The **-G** Option" on page 8-65. |
| O | F | Output misformatted | VMS format extensions assumed | Specify **-vms** and recompile. |
| E | C | Memory problems | Incorrect memory allocator | See "Memory Allocation" on page 8-34. |
| E | P | Memory problems | Incorrect memory allocator | See "Pascal Dynamic Memory Allocation" on page 8-42. |
| L | A | Invalid answers | Alignment Problems | Specify **-align8** or **16** and recompile. See "Alignment" on page 8-25. |
| B | A | Source files/ Library files missing | Library paths not correctly defined in **Makefile** | See "Optimization" on page 8-64. |
| E | C | Segmentation violation | Dereferencing nil pointer | See "Dereferencing **nil** Pointers" on page 8-25. |
| E | A | Bus error | Alignment, data | Use **dbx** to locate, then correct. |
| E | A | Bus error/library routine | Alignment I/O | Copy data to aligned structure and recompile; align all data structures. |
| E | A | Range of errors | Uninitialized values | Initialize-Fix code and recompile. |
| E | A | Range of errors | Wrong assumption nil value | See "Floating Point Arithmetic" on page 8-21. |
| E | A | Out-of-range error | Floating point declaration | See "Alignment" on page 8-25. |
| E | A | Poor performance | Not optimized | See "Debugging" on page 8-60. |
| E | A | Program traverses directory tree and does not return to original place | Use of **chdir()** | Avoid using relative paths. |
| E | A | Incorrect results | Order of evaluation | See "Order of Evaluation" on page 8-27. |

| * | ** | Symptom | Possible Problem Source | Action |
|---|---|---|---|---|
| E | A | Incorrect results | Uninitialized values | See "Uninitialized Variables" on page 8-26. |
| E | F | Incorrect results | Assuming subroutine variables hold value between calls | Specify **-static** and recompile. |
| E | F | High system time | Uninitialized value | Specify **-static** and recompile. |
| E | F | Bus error | Alignment problems | See "Retention of Data" on page 8-49. |
| C | F | Link edit fails | **gp** area overflow | See "Optimization" section (general) on page 8-64 and "Variable Length Argument Lists" (FORTRAN) on page 8-50. |
| E | A | Incorrect results | Replacing double precision for extended | Analyze why your program is using extended precision. |
| E | C | Degenerate unsigned comparison | Character variables pointers wrong | Use a mask or propagate bits. See "Signed **chars**" on page 8-35. |
| L | A | **$gp** area overflow | Conflicting definitions of variables; Large numbers of small variables | Relink using suggested **-G num**. See "The **-G** Option" on page 8-65. |
| L | A | Undefined external, but external in library | Use of external occurs after scan of particular library | See "Debugging" on page 8-60. |
| L | A | **$gp** pointer not initialized | Use of non-standard startup code | See "Debugging" on page 8-60. |
| L | A | Unexpected undefined externals | Use of non-standard or machine specific library functions | Replace with ES/os equivalent functions. |
| L | F | Incompatible common block length warning | Program uses subject of common subroutines. Different variable types declared in various subprograms. | Assure full definition occurs in main routine. See "Variable Length Argument Lists" on page 8-50. |
| L | F | Illegal initialization error | Common data initialized in two or more locations | See "Runtime Checking" on page 8-50. |

| * | ** | Symptom | Possible Problem Source | Action |
|---|---|---|---|---|
| C | F | Numerous invalid/unrecognized statements | Program source in card format or uses over 72 columns | Use **-col72** or **col120** switches. |
| C | A | Undefined conditional flag | Error in use of **-D** flag | See "Using the C Preprocessor" on page 8-32. |
| L | C | Unexpected or undefined externals | Assumption compiler prepends _ | See "Leading '_' " on page 8-36. |
| C | C | Illegal variable usage | Use of **typedef** Name as an argument to function | See "**typedef** Names" on page 8-37. |
| E | A | Uninitialized data warnings | Not initializing variables | See "Uninitialized Variables" on page 8-26. |
| E | C | Wrong results | Program assumes that data types are of equal length | Use variable argument macros. See "**varargs**" on page 8-36. |
| E | C | Compute-time error | Casting on the left hand side of an assignment | Don't do it. See "Casting" on page 8-38. |
| E | F | Runtime error | Subscripts exceed the specified range | See "Static Versus Automatic Allocation" on page 8-48. |
| E | F | **$gp** area overflow | Uneven block sizes | See "Variable Length Argument Lists" on page 8-50. |
| E | F | Uninitialized local variables | **-static** is repeated | See "Alignment of Data Types" on page 8-50. |
| C | F | Illegal integer<br>Illegal space<br>Multiple defined symbols | Failure to initialize each named common block within one subprogram | Use the **fsplit** utility. See "Runtime Checking" on page 8-50. |
| L | A | Cannot put a large variable into the **gp** area | Inconsistent size declaration | See "The **-G** Option" on page 8-65. |
| L | A | **gp** register is not initialized | You use your own startup code | Use runtime startup code. See "Optimization" on page 8-64. |
| L | A | Link editor fails to obtain correct libraries | Did not use the language's default library and did not specify the user's library | See "The Link Editor" on page 8-65. |

| * | ** | Symptom | Possible Problem Source | Action |
|---|---|---|---|---|
| L | A | Include files missing or errors in include files | BSD sources | Use **-systype bsd43** or use **-I/usr/include -I/usr/include/bsd -lbsd** |
| C | C | Illegal Identifier Name | Identifier with **$** in name | See "Dollar Sign in Identifier Names" on page 8-38. |
| E | C | Segmentation or bus error | Improper assumptions with **malloc** | See "Additional Keywords" on page 8-38. |
| E | A | Very long runtime | Denormalized FP values.<br><br>Algorithm terminates based on inappropriate FP values. | Check for uninitialized values or reference of DP by SP.<br>See "Alignment" on page 8-25. |
| E O | F | Incorrect results, especially very small DP floating point answers | Equivalence making invalid assumption on storage size of variables | See "Retention of Data" on page 8-49. |
| L | A | Unable to satisfy externals when subprocedure is in second language | Assumption about compiler external names; for example, **_prefix**, **_postfix** | See appropriate language user's guide. |
| C | C | Indicates that the internal compiler tables are full | Internal compiler tables are too small. | Pass the following arguments with **cc** command to increase table size:<br>**-Wf, XNd50000**<br>**-Wf, XNp50000** |

## ES/os Considerations

This section briefly describes ES/os and the operating system dependencies that you need to be aware of when you are porting an application program from

- BSD 4.3 to ES/os BSD mode,
- BSD 4.3 to ES/os System V mode,
- System V to ES/os BSD mode,
- System V to ES/os System V mode.

### ES/os

ES/os is an AT&T System V 3.2-based kernel with BSD enhancements, including all BSD 4.3 system calls, BSD 4.3 library functions, most BSD 4.3 commands, TCP/IP networking, the NFS 4.0 remote file system, and the Berkeley Fast File System.

In addition, ES/os includes support for the IEEE Standard Portable Operating System Interface for computer environments (POSIX). Programs can be compiled using the POSIX interface by specifying **-systype posix** on the compile line. Only the library routines specified by IEEE standard 1003.1 are implemented, so a restricted set of programs will compile in this environment.

ES/os is packaged so that you can concurrently access either BSD or System V commands. System V programs are permitted to use some BSD system calls. These system calls, as noted in "ES/os Differences" (below), plus a few more are still available in ES/os for System V programmers.

**Note:** Programmers are restricted to programming in either BSD or System V environments; mixed mode programming is not fully supported.

### Porting from BSD-Derived Systems

ES/os compiles programs under System V or BSD depending on the following:

- The setting of the ***PATH*** variable in your environment,
- Use of the **-systype** option to the compile command.

In order to successfully compile programs for BSD functionality, you must do one of two things:

1) Use the compile time switch **-systype bsd43** which prepends **/bsd43** to the path for include files and libraries:

```
% cc -systype bsd43 -g -o sample sample.c
```

2) Place **/bsd43/bin** before **/bin** in the ***PATH*** variable in your **.cshrc**, **.profile**, or **.login** file. When you compile, your system goes to the **/bsd43** command directory and uses the BSD **cc** command which contains the switch **-systype bsd43**.

If you want to compile a program for System V functionality and you have placed **/bsd43** in your path prior to **/bin**, you must use the **-systype sysv** switch, as in:

```
% cc -systype sysv -g -o sample sample.c
```

The default compile time switch for **/bin/cc** is **-systype sysv** and the default compile time switch for **/bsd43/bin/cc** is **-systype bsd43**.

**Porting from 4.3 BSD to ES/os (BSD based)**

Several areas must be considered when converting a program from a regular BSD system to an ES/os BSD system.

- Include files

  Though textually different, the 4.3 BSD compilation environment include files are functionally equivalent to the 4.3 BSD include files.

  The only differences between the two are in areas where the system *cannot be made compatible. For example, the file* ***/etc/utmp*** *does not contain the field* ***ut_host*** *and the include file that describes the* ***utmp*** file (**/bsd43/usr/include/utmp.h**) contains a special marker that gives an error when compiling code using the **ut_host** field. Such code must be changed.

- Libraries

  All standard 4.3 BSD libraries are provided. In some cases, the libraries use the corresponding System V code. For example, the **libc** routines which get password and group file entries are from System V.

  In the case of **curses**, the System V.3 **curses** (based on **terminfo**) is used. Except where the programs try to use the value of the buffer returned by the **tgetent()** function, this version of **curses** provides the entire 4.3 BSD interface.

- **termcap**

  Only **terminfo** is supported. In general, programs that use **termcap** and/or **curses** work as expected. The features of **termcap** that are missing are:

  - The ability to modify the **termcap** on the fly. This is often done to set the terminal size. This can be done by setting the window size with **winsize(1)** or by setting the environment variables ***LINES*** and ***COLUMNS***. The former is the preferred method.

- The ability to add new capabilities to the database. This cannot be emulated without changing the code.

- **IOCTL** commands

  Virtually all 4.3 BSD **IOCTLs** are supported on ES/os when using the 4.3 BSD compilation environment. The only time you need to make any changes to your code is if your program does extensive tty manipulation. If this is the case, you should convert the tty handling to System V. For more information, see **termio(7)**.

- Command functionality

  If a program **execs** a BSD command, then you should verify that the command exists as a BSD command; that is, it can be found in **/bsd43/bin** and the **exec** command path should be changed accordingly. Otherwise, you should make sure that System V functionality is sufficient.

For a complete description of ES/os (RISC/os) system functionality, please see the MIPS *RISC 4.0 Release Notes* and check with your system administrator to verify that the BSD subpackage has been installed on your system.

**Porting from 4.3 BSD to ES/os (System V based)**

Many programs written in C can be ported from BSD systems without changing any code by specifying compiler and link editor compiler options as follows:

- During the compilation step, use the **-I/usr/include/bsd** and **-signed** options. The **-I/usr/include/bsd** option causes include files to be searched for in **/usr/include/bsd** before **/usr/include**, so BSD values take precedence. The **-signed** option causes all **char**-typed data to be signed.

- During the link editor step, use the **-lsun**, **-lbsd**, and **-lrpcsvc** compiler options to link in routines that are not part of the standard C library for System V, but which are needed by BSD and NFS programs.

**Note:** Use only the **-lsun** and **-lrpcsvc** compiler option for programs requiring RPC and/or XDR.

### Differences Between ES/os System V and BSD

This section describes some of the differences between ES/os and 4.3 BSD UNIX.

- **math.h**

  Programs that use **libm** should be modified to include the library **math.h**. You cannot assume that the type of these functions under ES/os is the same as on other systems.

- **longjmp()**

  If your program calls the function **longjmp()** from the signal handlers, it may need special work before being optimized with **-02** (use of **-02** is not recommended). Global variables may be placed in the registers and the values may not be restored properly. You may either explicitly declare the appropriate variable as volatile or use the **-volatile** compile option. Using this option significantly reduces the amount of optimization that is done. For a complete description of the **-volatile** option, see the MIPS *Language Programmer's Guide.*

- Variable Arguments

  The typical mechanism for passing variable argument lists on BSD systems assumes that a parameter is a pointer to an array of pointers; this does not work on ES/os. Instead you must use the **vararg.h** or **stdarg.h** macros. For a description of these macros, see appendix A of the MIPS *Language Programmer's Guide.*

- System Administration Files

  System administration files, such as **/etc/passwd**, **/etc/inetd.conf**, and the **utmp** file may differ from what is expected by some applications. Some other files, such as **/etc/ttys**, may be missing.

- Dereferencing a Pointer

  Address 0 is an invalid address. On many BSD systems, this address is addressable; C programs may depend on being able to dereference pointers with this address. Dereferencing a pointer with a value of 0 is incorrect according to all C standards.

- Pseudo-ttys

  Pseudo-ttys use a "clone device" instead of having pairs of **pty/tty.**

- COFF Format

  The ESV's object file format (COFF) is a modified UNIX System V COFF and differs markedly from the BSD object file format. Therefore, BSD programs that process object file programs as input must be modified to access information correctly from the ESV COFF.

- tty Interface

  The tty driver interface does not have a complete emulation. Programs that rely heavily on the **tty ioctls** are difficult to port.

- Load Average

  The "avenrun" (load average) kernel symbol contains items of type **FIX**, as defined in **sys/fixpoint.h**, not doubles.

- **malloc()**

  On Apollo systems, there is a system call to pre-allocate memory for **malloc()**. This call is not supported and is not needed on ESV Workstations. For more information on additional **malloc()** library calls, see the "FORTRAN and C Interface" section in this chapter.

## Porting from System V-Derived Systems

A program that runs on System V will port more easily to ES/os if you include **math.h** for math library functions. Do not assume that the type of these functions is the same as on other systems.

### libbsd.a

The library **/usr/lib/libbsd.a** is a System V library provided by Evans & Sutherland which contains some 4.3 BSD system calls and library routines. Because of file sizes when the library was introduced, the routines in this library have been renamed to approximate their 4.3 BSD routine names. For example, the **getdomnm** in **libbsd.a** is **getdomainname** in the 4.3 BSD **libc.a** library.

Table 8-1 lists the **libbsd.a** routines, the BSD **libc.a** names, and gives an explanation of the routine.

**Table 8-1. libbsd.a routines**

| **libbsd.a** file name | 4.3 BSD **libc.a** file name | Description |
|---|---|---|
| **accept** (2-BSD) | | Accepts a connection on a socket. |
| **bcopy, bcmp, bzero, ffs** (3-BSD) | | Bit and byte string operations. |
| **bind** (2-BSD) | | Bind a name to a socket. |
| **bindresvport** | | Bind a socket to a privileged IP port. |
| **connect** (2-BSD) | | Initiate a connection on a socket. |
| **ctype** | | Character type routines. |
| **dbm_open, dbm_close, dbm_fetch, dbm_store, dbm_delete, dbm_firstkey, dbm_nextkey, dbm_error, dbm_clearerr** | | Data base subroutines. |
| **ether_addr** | | Ethernet mapping operation. |
| **exportent** | | Get exported file system info. |
| **flock** | | File locking. |
| **getdomnm** (2-BSD) | **getdomainname, setdomainname** | Get/set name of current domain. |
| **getdtabsz** (2-BSD) | **getdtablesize** | Get descriptor table size. |
| **getgroups, setgroups** | | Get/set group access lists. |
| **gethostid, sethostid** (2-BSD) | | Get/set unique identifier of current host. |
| **gethostnamadr** (3N-BSD) | **gethostbyname, gethostbyaddr, gethostent, sethostent, endhostent** | Get network host entry. |
| **gethostname,** | | Get/set name of current host. |

**Table 8-1. libbsd.a routines continued**

| libbsd.a file name | 4.3 BSD libc.a file name | Description |
|---|---|---|
| **getnetent, getnetbya, getnetbynm** | **getnetbyaddr, getnetbyname, setnetent, endnetent** (3N-BSD) | Get network entry. |
| **getnetgrent** | | Get network group entry. |
| **getpeernm** (2-BSD) | **getpeername** | Get name of connected peer. |
| **getprotoe** (3N-BSD) | **getprotoent, getproatobynumber, getprotobyname, setprotoent, entprotoent** | Get protocol entry. |
| **getrlimit, setrlimit** (2-BSD) | | Control maximum system resource consumption. |
| **getrpcent** | | Get RPC entry. |
| **getrusage** (2-BSD) | | Get information about resource utilization. |
| **getservent,** | | Get service entry. |
| **getsocknm** (2-BSD) | **getsockname** | Get socket name. |
| **getsockopt, setsockopt** (2-BSD) | | Get and set options on sockets. |
| **gettimeofday,** | | Get/set date and time. |
| **getwd** (3-BSD) | | Get current working directory path name. |
| **htonl, htons, ntohl, ntohs** (3N-BSD) | | Convert values between host and network byte order. |
| **in_addr** (3N-BSD) | **inet_addr** | Internet address manipulation routines. |
| **in_lnalf** (3N-BSD) | **inet_lnalf** | Internet address manipulation routines. |
| **in_mkaddr** (3N-BSD) | **inet_mkaddr** | Internet address manipulation routines. |

**Table 8-1. libbsd.a routines continued**

| **libbsd.a** file name | 4.3 BSD **libc.a** file name | Description |
|---|---|---|
| **in_netof** (3N-BSD) | **inet_netof** | Internet address manipulation routines. |
| **in_network** (3N-BSD) | **inet_network** | Internet address manipulation routines. |
| **in_ntoa** (3N-BSD) | **inet_ntoa** | Internet address manipulation routines. |
| **index** | | Similar to **rindex**. |
| **initgroups** | | Initialize group access list. |
| **innetgr** | | Get network group entry. |
| **insque, remque** (3-BSD) | | Insert/remove element from a queue. |
| **listen** (2-BSD) | | Listen for connections on a socket. |
| **mktemp** | | Make a unique filename. |
| **ndmb** | | Data base routines. |
| **ovfork** | | Virtual forking routine. |
| **opendir, readdir, seekdir, closedir** (3-BSD) | | Directory operations. |
| **random, srandom,** | | Better random number generator; routines for changing generators. |
| **rcmd,** | | Routines for returning a stream to a remote command. |
| **recv, recvfrom, recvmsg** (2-BSD) | | Receive a message from a socket. |
| **res_comp, res_debug res_init, res_mkquery res_query, res_send** | | Resolver functions. |
| **rexec** (3-BSD) | | Return stream to a remote command. |
| **rindex** (3-BSD) | | String operations. |
| **ruserpass** | | Return remote user password. |

**Table 8-1. libbsd.a routines continued**

| libbsd.a file name | 4.3 BSD libc.a file name | Description |
|---|---|---|
| **scandir**(3-BSD) | | Scan a directory. |
| **select** (2-BSD) | | Synchronous I/O multiplexing. |
| **send, sendto, sendmsg** (2-BSD) | | Send a message from a socket. |
| **setdomnm** | | Set domain name. |
| **setegid** (2-BSD) | | Set real and effective group ID. |
| **sethostent** | | Set network host entry. |
| **setreuid** (2-BSD) | | Set real and effective user IDs. |
| **setrgegid, setreuid** | | Set user and group IDs. |
| **setsockopt** | | Set socket options. |
| **setuid, seteuid, setruid, setgid, setegid, setrgid,** (3-BSD) | | Set user and group ID. |
| **shutdown** (2-BSD) | | Shut down part of a full-duplex connection. |
| **socket** (2-BSD) | | Create an endpoint for communication. |
| **syslog** (3-BSD) | | Control system log. |
| **yp_bind, yp_enum yp_master, yp_match, yp_order yp_update, yperr_string, ypmaint_xdr, ypprot_err, ypv1_xdr, ypxdr, ypupdate_xdr yp_getgrent, yp_gethostent yp_getpent, yp_getpwent yp_getrpcent, yp_getservent yp_gnetent, yp_gnetgr, yp_innetgr, yp_service, yp_all** | | NIS (yellow pages) routines. |

**ES/os Differences**

This section describes some differences between ES/os and regular System V UNIX.

- **Symbolic Links**

  The presence of symbolic links can cause problems if you use the UNIX function **chdir()**. Programs should avoid using relative paths to change directories, because the sequence:

  ```
  chdir("./subdir"); chdir("..")
  ```
  may not set the directory back to the original place.

- **File Name Size**

  The maximum size of a file name is 255 characters, not 14. This causes problems when programs expect truncation. For example, a program could reject a user-supplied filename that is larger than 14 characters because it assumes that 14 characters is the limit. In addition, if a program declares an array to hold a filename, the array may be too small, especially if it is declared to be 15 characters long. ***MAXNAMLEN*** in **/usr/include/dirent.h** and ***MAXPATHLEN*** in **usr/include/sys/naml.h** are appropriate to use instead.

- **Directory Access**

  The ESV file system does not allow your program to directly read a directory. Programs that do this are considered to be nonportable to ES/os even though they may work on many versions of System V UNIX.

- **longjmp()**

  Programs that call the function **longjmp()** from the signal handlers may need special work before being optimized with **-02** and use of **-02** is not recommended. Global variables may be placed in the registers and the values may not be restored properly, but declaring appropriate variables as volatile solves this problem. Use the **-volatile** compiler option, which causes all objects to be treated as volatile. Remember that using this option significantly reduces the amount of optimization that is done.

- **COFF Format**

  Programs that work with object and executable files need some work, as the ESV COFF format is not completely compatible with System V.

- **Dereferencing a Pointer**

  Address 0 is an invalid address. On many SYSV systems, this address contains a 0, and C programs may depend on being able to dereference pointers with this value. Dereferencing a pointer with 0 is incorrect according to all C standards.

### Dereferencing nil Pointers

Programs that contain errors sometimes go undetected on machines where dereferencing a zero pointer yields zero. Typically, the programmer meant to write:

```
int *c;
if (c != 0 && *c) ...;
```

but actually wrote:

```
int *c;
if (*c) ...
```

On most VAX UNIX systems, the error goes undetected; on most MC68000 implementations and on the ESV Workstations, this causes a segmentation violation.

### Where Text and Data Lie in Memory

Figure 8-1 illustrates how text and data are arranged in memory on ESV machines and VAX machines. Refer to this figure as you read the paragraphs that follow it.

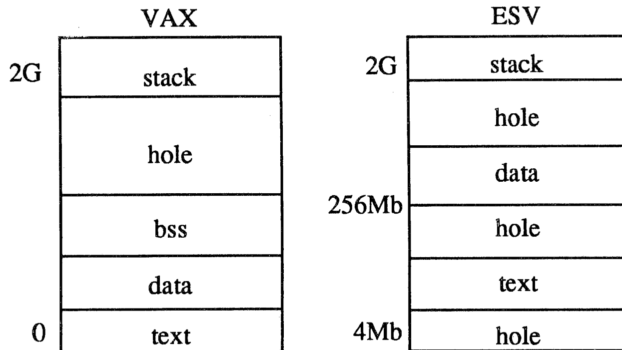



**Figure 8-1. A comparison of VAX and ESV address space**

You may encounter problems when you port a program if you assume that the link editor-defined symbol **etext** indicates the beginning of the data section as well as the end of the text section. As illustrated in figure 8-1, **etext** works on a VAX because data and text are located next to each other. This is not the case on an ESV Workstation. To solve this problem, the ESV link ed-

itor provides other symbols for the beginning and end of each text section; for more information see the **end(3)** manual page.

Some sophisticated UNIX programs such as GNU's emacs assume that the program text (that is, the executable code) or data starts at a low address in memory. The program can, for example, store tag information in the high-order bits of a pointer and mask out the tag just before dereferencing the pointer. Unless you specify otherwise with the link editor **-T** and **-D** options, program text on ESV Workstations starts at approximately 0x400000, program data approximately 0x10000000, and the program stack at approximately 0x80000000.

## Changing Internal Compiler Table Size

The ES/os **cc** command uses several different internal tables. Their sizes seem to be too small for large applications. ES/os provides an undocumented method of expanding the table sizes by passing a special argument with the **cc** command. The following arguments may resolve the problem.

```
-Wf, -XNd50000
-Wf, -XNp50000
```

## Porting from Other Operating Systems

This section discusses the issues that you need to be aware of when porting programs from systems other than UNIX.

### General

In general, if you are porting from operating systems other than UNIX, you need a working knowledge of both ES/os and the operating system that you are porting from.

Programs written with no operating system dependencies should port easily. Such programs use the standard I/O routines for the language in which they are written rather than using UNIX system calls are more likely to port with little or no modification. C programs that follow the ANSI C Standard should work as expected.

### Porting FORTRAN Programs from VAX

Library functions that provide an interface to ES/os (similar to those provided by the C library) are available to ESV-FORTRAN programs. Also, intrinsic subroutine and functions used to interface VAX systems are available to provide the same functional interface to ES/os from ESV-FORTRAN programs. Chapter 4, part I, of the *MIPS-FORTRAN Programmer's Guide and Language Reference* describes these system functions and subroutines.

It will greatly facilitate porting from a VAX environment if your program was not written to take advantage of VMS extensions to FORTRAN. While the

ES/os compiler suite follows ANSI standards, the sophistication of the compilers mandates a stricter interpretation of the standard.

## Hardware-Related Considerations

This section discusses specific ES/os implementation that you need to consider when porting programs.

### Floating Point Arithmetic

In 1985, ANSI/IEEE 754-1985 defined a standard floating point representation and arithmetic. ESV Workstations conform to this standard. While you might expect conformance to this standard to eliminate problems with porting floating point programs, there are still significant differences that can hinder an implementation's portability.

Floating point differences manifest themselves in the following ways:

- Producing slightly different results,
- Producing incorrect results,
- Slow execution,
- Faulting.

#### General IEEE 754

Table 8-2 lists the IEEE floating point format. The explicit use of extended precision formats available on some IEEE 754 floating point implementations makes programs nonportable, because there is no simple or efficient way to get the range or accuracy of IEEE extended on a machine whose highest precision is double. To avoid this problem, try using double precision, but be aware that this may yield incorrect results. Therefore, before you substitute double for extended, analyze the reasons your program uses the extended format.

Some compilers use extended precision even when your program does not specify it. This may be the case with certain hardware such as Motorola 68881, 68882, and Intel 80387, where it is the only efficient thing to do. When an expression is evaluated using extended precision, you may get a slightly different answer than if it were evaluated in double precision.

The implementation of library math functions differs from machine to machine, so you will see slightly different results when you run programs on ES/os.

**Table 8-2. IEEE floating point format**

| Format | Size | Radix | Approximate Range | Rounding | Exceptions |
|---|---|---|---|---|---|
| single format | 32 bit | 2 with 24 bits of precision | $10^{-45}$ to $10^{38}$ | round to nearest round .5 to even | for overflows, divide by zero, and so on, do not fault, but instead return special symbols. |
| double format | 64 bit | 2 with 53 bits of precision | $10^{-324}$ to $10^{308}$ | same as above | same as above |
| extended | 80 + bit | 2 with 64 bits of precision | $10^{-4951}$ to $10^{4931}$ | same as above | same as above |

## DEC VAX

Table 8-3 lists the DEC VAX floating point format. If you port a program that uses VAX floating point to ES/os, which uses IEEE floating point, remember that the IEEE floating point format is much more likely to cause problems between single and double precision in loosely typed languages like FORTRAN.

For example, the two VAX single and double precision formats are identical except that the double-precision format provides additional precision. If you reference something as single precision that is really double, your mistake has little effect on the value. Because IEEE 32-bit and 64-bit formats are different, the same mistake made on ES/os, can produce data that does not even resemble the data produced on the original machine.

Use of H-format (**REAL*16**) is not portable to ES/os, because its floating point does not have the range or accuracy of H-format. Using IEEE double precision will likely give incorrect answers.

The default, double precision, D-format has more precision but less exponent range than IEEE double, thus precision-sensitive programs may give different results.

**Table 8-3. VMS floating point format**

| Format | Size | Radix | Approximate Range | Rounding | Exceptions |
|---|---|---|---|---|---|
| F-format | 32 bit | 2 with 24 bits of precision | $10^{-38}$ to $10^{38}$ | round to nearest round .5 and up | for overflows, divide by zero, |
| D-format* | 64 bit | 2 with 56 bits of precision | $10^{-38}$ to $10^{38}$ | same as above | same as above, |
| G-format* | 64 bit | 2 with 53 bits of precision | $10^{-308}$ to $10^{307}$ | same as above | same as above |
| H-format | 128 bit | 2 with 112 bits of precision | $10^{-4933}$ to $10^{4931}$ | same as above | same as above |

* F-format is similar to IEEE single format and G-format is similar to IEEE double format.

### IBM 370

Table 8-4 lists the IBM 370 floating point formats. Programs that depend on the larger single-precision exponent range are nonportable. ESV Workstations generally provide better accuracy, and therefore different results.

**Table 8-4. IBM 370 floating point format**

| Format | Size | Radix | Approximate Range | Rounding |
|---|---|---|---|---|
| single format | 32 bit | 16 with 6 radix-16 digits precision | $10^{-73}$ to $10^{75}$ | chopped |
| double format | 64 bit | 16 with 14 radix-16 digits precision | $10^{-73}$ to $10^{75}$ | same as above |
| Real *16 | 128 bit | 16 with 30 radix-16 digits precision implemented in software | $10^{-73}$ to $10^{75}$ | same as above |

### Cray

Table 8-5 lists the Cray floating point format. Because Cray's single-precision is a 64-bit format, it is generally necessary to switch to ES/os double precision to get the same results. Also, if your program depends on a large exponent range or 128-bit precision, program modifications are required.

ES/os provides better accuracy than Cray's 64-bit format, and therefore different results occur.

**Table 8-5. Cray floating point format**

| Format | Size | Radix | Approximate Range | Rounding |
|---|---|---|---|---|
| single format | 64 bit | 48 bits of precision | $10^{-2460}$ to $10^{2460}$ | chopped or worse |
| double format | 128 bit | 95 bits of precision implemented in software | $10^{-2460}$ to $10^{2460}$ | same as above |

### Math Library Accuracy

Besides basic floating point format and accuracy issues, each implementation typically differs in the algorithms and characteristics of its math library. Even IEEE 754-1985 machines that are otherwise identical may produce different results due to differences in math libraries. See **math(3M)** for additional information on the ES/os math library. The algorithms are generally from *Cody and Waite* with some additions and replacements from 4.3 BSD.

## Endianness

ESV Workstations use the byte ordering scheme called *big-endian.* Machines that number the bytes from left to right and the least significant byte is 3 are called big-endian; machines that number the bytes from right to left and the least significant byte is zero, within a 32-bit integer, are called *little-endian.* See appendix D and "Byte Ordering" options in chapter 1 of the MIPS *Language Programmer's Guide* for more information on these byte ordering schemes. The MIPS R2000 and R3000 chip can operate either way.

You may create porting problems by placing small objects side by side to make a bigger object, or splitting a big object into small objects. For example, the following code that reads and compares a pair of shorts is machine-dependent, because on some machines the $0^{th}$ element of the array represents the high-order half of the word rather than the low-order half:

```
char carray[BUFSIZ];
err = read(0, carray, 4);
if ((carray[0] | (carray[1] << 8)) >
(carray[2] | (carray[3] << 8))) ...
```

There is never a problem if you use the correct data type and let the compiler deal with the order of the bytes:

```
short sarray[BUFSIZ];
err = read(0, (char *) sarray, 2 * sizeof(short));
if (sarray[0] >sarray[1]) ...
```

Similarly, the following code to print four characters stored within an integer is machine-dependent because it assumes the first character is at the low-order end of the integer:

```
unsigned i;
printf("%c%c%c%c\n", i & 0xff, (i >>8) & 0xff,
        (i >>16) & 0xff,(i >>24) & 0xff);
```

A better solution is the following:

```
unsigned i;
printf("%.4s\en", (char *) &i);
```

### Alignment

ES/os architecture requires that each piece of data in memory be aligned on a boundary appropriate to its size. For example, an ***n*** byte integer can be aligned on a boundary whose address is a multiple of ***n*** bytes, up to a maximum of 8 bytes. This restriction permits the memory system to run much faster.

Ordinarily alignment has no effect on correctly written programs, because the compiler inserts unused space ("padding") between variables wherever necessary to conform to the rules. Language standards almost always permit such padding, and in the rare cases where the language forbids it, the compiler conforms to the language requirements by loading and storing objects in special ways (see the "FORTRAN Programming Language" section in this chapter for information on how this applies to FORTRAN programs).

A program that follows the rules of its language usually doesn't encounter problems. To avoid alignment problems, declare the fields of a structure in descending order by size.

Even a program that follows the rules given above may have trouble when writing data on one machine and reading it on another. In fact, padding is only one of many problems: machines differ with regard to endianness, floating point formats, the size of the integer, and the width of a character or word. There are two collective solutions to these problems: if I/O speed is not important, use ASCII files rather than binary ones; otherwise, consider using the **xdr(3N)** subroutine package for external data representation.

See the "FORTRAN Programming Language" section in this chapter for a discussion of the extensions to the compiler system for dealing with misaligned data as they apply to FORTRAN programs.

You may choose one of the following three command-line arguments to deal with various degrees of misalignment:

- **-align8**

  Permits objects larger than 8 bits to be aligned on 8-bit boundaries. This option requires the greatest amount of space; however, it is the most complete solution; 16-bit padding is not inserted for **integer*2** objects within common blocks.

- **-align16**

  Permits objects larger than 16 bits to be aligned on 16-bit boundaries; 16-bit objects must still be aligned on 16-bit boundaries (MC68000-like alignment rules); 16-bit padding is not inserted for **integer*2** objects within common blocks.

- **-align32**

  Permits objects larger than 32 bits to be aligned on 32-bit boundaries; 16-bit objects must still be aligned on 16-bit boundaries, and 32-bit objects must still be aligned on 32-bit boundaries. This option requires the least amount of space, but isn't a complete solution; 16-bit padding is inserted for **integer*2** objects within common blocks.

### Uninitialized Variables

Whenever possible, initialize local variables. The **lint(1)** C program checker and ES/os compilers issue warning messages about uninitialized data in certain instances. However, because the system can see only the static characteristics of a program, it cannot warn about all instances of uninitialized data.

If your program's failures vary with the input data, but the variances are not logically related to the failing code, look for uninitialized variables.

In addition, if your program works when compiled with the default **-O1** optimization, but fails when compiled with **-O2** optimization, then the fault may be caused by uninitialized variables rather than the optimizer. In an **-O2** optimization, the optimizer may allocate an uninitialized variable to a register, creating an error that would not have occurred in an **-O1** optimization.

On an ESV Workstation, uninitialized variables can degrade the performance of a program that otherwise runs correctly. The hardware performs most IEEE operations, but software is invoked for operations on denormalized numbers. If, in performing computations on an uninitialized floating point variable, an uninitialized variable happens to be a denormalized IEEE value, then the algorithm in your program could continue to function properly even with a non-zero variable, provided it remains close to zero. This situation could deteriorate the performance and accuracy of your program.

If you suspect this problem, use the **time(1)** command. For most programs, the system CPU time is small compared to the user CPU time. If the system time is unexpectedly high but not high enough to account for the overall slowdown, that's a good indication of denormalized arithmetic. The system time does not account for the entire slowdown, because not all of the emulation time is charged against your program.

Another aid in diagnosing these problems is the **fpi(3)** floating point interrupt analyzer. The **fpi** routines count the instances of floating-point emulation and print a summary.

## Undefined Language Elements

Language standards deliberately avoid defining certain language constructs, thus causing inconsistencies among different implementations of the same language. This section explains how the ES/os compiler defines some of these constructs, which you may need to alter in the program being ported.

### The Value of nil

C and Pascal do not specify the value that the compiler must use to represent a *nil* (or null) pointer. However, C does dictate that the compiler must recognize a zero in the source program as the notation for a nil pointer and convert it into whatever value does represent nil.

The ES/os compiler uses zero to represent nil. Few UNIX programs encounter any difficulty with this, but other operating systems use other values like "-1" or "- maxint - 1." A portable program should not depend on this value.

### Order of Evaluation

The order in which program statements are evaluated can cause problems.

For example, the expression in the following Pascal statement can cause trouble if the programmer hoped that it would invoke the decrement function on both variable ***x*** and variable ***y***:

```
if (decrement(x) < 0) and (decrement(y) < 0) then ...
```

As another example of the side effects, neither language specifies the order in which the compiler evaluates an actual argument list:

```
foo(decrement(x), x+ y);
```

Another example is the C statement:

```
foo(*p++, *p++);
```

The best way to control the order of evaluation in a program being ported to an ESV Workstation is to introduce temporary variables. Because of global optimization, this usually costs nothing (apart from forcing the intended order

and degree of evaluation). The compiler attempts to allocate all of the objects to registers. See the code example below.

```
templ = decrement(x);
temp2 = decrement(y);
if (templ < 0 and temp2 < 0) then ...

templ = decrement(x);
call foo(templ, x+ y);
```

### Inter-Language Interfaces

The allocation of variables in memory, the rules of argument passing, and the mapping of source language identifiers onto assembly level symbols all pose problems that appear when you stop programming entirely within one language and start calling routines written in another language.

For example, Pascal specifies that the **ord** function must return zero for a Boolean false and one for a Boolean true; but Pascal does not specify whether a Boolean value is stored in memory as a single bit, a byte, or a full word. In fact, Pascal permits a compiler to implement true by setting the sign bit of a word, or even by setting all bits to 1, provided the **ord** function performs the appropriate conversion. As long as you program entirely in Pascal, you need never know these details, but when Pascal code passes a Boolean to a C subroutine, the latter must know whether to expect a **char**, a **short**, or an **int**, and what value constitutes true.

For information on interfaces between C and Pascal programs, see chapter 4 in the MIPS *Language Programmer's Manual.* For information on the interfaces between FORTRAN and C programs, and FORTRAN and Pascal programs, see the *MIPS-FORTRAN Programmer's Guide.*

## C Programming Language

ES/os C conforms to the *de facto* standard established by the Kernighan and Ritchie text and the AT&T portable C compiler. It provides certain extensions, such as prototype declarations, suggested by the draft ANSI C standard. See appendix A in the MIPS *Language Programmer's Guide* for more information on C extensions.

## Supported cc Options

The ES/os **cc** command supports the following options among others:

| Option | Description |
|---|---|
| **-c** | Suppresses linking with **ld** and produces a **.o** file for each source file. |
| **-C** | Prevents the C preprocessor from removing comments. |
| **-cord** | Runs the procedure-rearranger on the resulting file after linking. |
| **-cpp** | Runs the C macro preprocessor on C and assembly source files before compiling. |
| **-D*name =def*** | |
| **-D*name*** | Defines the ***name*** to the C preprocessor, as if by **#define**. If no definition is given, the name is defined as 1. |
| **-E** | Runs the source file through the C preprocessor only. |
| **-feedback *file*** | Specifies ***file*** to be used as a feedback file. |
| **-g[0-3]** | Produces additional symbol table information for **dbx** to enable bugging. 0 is none, 3 is for optimized code. |
| **-G num** | Specifies the maximum size of a data item to be accessed from the global pointer. |
| **-I** | The standard directory is never searched for **#include** files. |
| **-Idir** | **#include** files whose names do not begin with "/" are always sought first in the directory of the file argument, then in directories specified in **-I** options, finally in the standard directory. |
| **-lx** | Links with object library, **libx.a**. |
| **-j** | Compiles the specified source programs, and leaves the ucode object file output in corresponding files suffixed with **.u**. |
| **-ko *output*** | Names the output file created by the ucode loader as ***output***. |
| **-k** | Passes options that start with a **-k** to the ucode loader. |
| **-L*directory*** | Prepends ***directory*** to list of directories containing |

| | |
|---|---|
| | object-library routines (for linking using **ld**). |
| **-M** | Runs only the macro preprocessor on the named C programs. |
| **-nocpp** | Does not run the C macro preprocessor on C and assembly source files before compiling. |
| **-o *outputfile*** | Names the output file ***outputfile***. |
| **-Olimit *num*** | Specifies the maximum size of a routine that will be optimized by the global optimizer. |
| **-O[*level*]** | Optimizes the object code where ***level*** is from 0 to 3. |
| **-p[0-1]** | Compiles for profiling with **prof**. 0 is no profiling (default), and 1 is profiling turned on. |
| **-P** | Runs the source code through the C preprocessor only. |
| **-systype *name*** | Uses the named compilation environment ***name***. |
| **-S** | Assembles the program and produces an assembly source file, suffixed with **.s**. |
| **-std** | Has the compiler produce warnings for things that are not standard in the language. |
| **-U*name*** | Removes a symbol ***name*** from the C preprocessor. |
| **-v** | Prints verbose compiler information about each pass as it is executed. |
| **-V** | Prints the version of the compiler and the versions of all passes. |
| **-w** | Suppresses warning messages. |

### Supported ld Options

The ES/os **ld** command supports the following options among others. For more information on **ld**, consult the **ld** reference manual page.

| Option | Description |
|---|---|
| **-A *file*** | Specifies incremental loading. |
| **-b** | Does not merge the symbolic information for a file into one entry for that file. |
| **-B *hex*** | Sets the bss segment origin. |
| **-d** | Forces common storage for uninitialized variables and other common symbols. |

| | |
|---|---|
| **-dc** | Performs **-d** and copies initialized data referenced by program from shared objects. |
| **-dp** | Forces an alias definition of undefined procedure entry points. |
| **-D *hex*** | Pads the data segment with zero-valued bytes to make it ***hex*** bytes long. |
| **-e *entry*** | Defines the entry point. |
| **-f *fill*** | Sets the fill pattern. |
| **-G *num*** | Sets the maximum size of a variable to be allocated in the small bss section or small data section. |
| **-kl*x*** | Searches a library **lib*x*.b** where ***x*** is a string. |
| **-K*dir*** | Changes the default directories to the single directory ***dir***. |
| **-lx** | Abbreviates library name **libx.a.** |
| **-L** | Changes search algorithm to never look in the default directories. |
| **-L*dir*** | Adds ***dir*** to the list of directories in which to search for libraries. |
| **-m** | Produces a map of the input/output sections on the standard output. |
| **-M** | Produces a primitive load map. |
| **-n** | Makes the text portion read-only and shared among all users executing the file when the output file is executed. |
| **-N** | Does not make the text portion read-only or sharable. |
| **-o *name*** | Uses ***name*** as the name of the **ld** output file, instead of **a.out**. |
| **-p *file*** | Preserves the symbol names listed in ***file*** when loading ucode object files. |
| **-r** | Generates relocation bits in the output file so that it can be the subject of another **ld** run. |
| **-s** | Strips the output by removing the symbol table and relocation sections to save space. |
| **-S** | Strips the output by removing all symbols except locals and globals. |

| | |
|---|---|
| **-T [text] *hex*** | Starts the text segment at location ***hex.*** |
| **-Tdata hex** | Starts the data segment at location ***hex***. |
| **-u *name*** | Enters ***name*** as an undefined symbol. |
| **-x** | Preserves only global (non-**.globl**) symbols in the output symbol table. |
| **-y*sym*** | Indicates each file in which ***sym*** appears, its type, and whether the file defines or references it. |
| **-v** | Sets verbose mode. |
| **-V** | Prints the version of **ld** being used. |
| **-z** | Arranges for the process demand paged from the resulting executable file. |

### Using the C Preprocessor

To maintain your program on both an old system and on an ESV Workstation, consider using the **#ifdef** conditional-compilation facility provided by the **cpp** preprocessor. The C and Pascal compilers provide this feature by default; the FORTRAN compiler provides it if you either use the **-cpp** compiler option, or give your source file a name ending in **.F** rather than **.f**. Using **cpp**, you can include the following conditional statements in your program:

```
#ifdef MY_OLD_MACHINE
x := #ff5a;
#endif /* MY_OLD_MACHINE */
#ifdef MIPS
x := 16#ff5a;
#endif /* MIPS */
```

Then, you can use the **-D** option to select the appropriate version. For example, to generate an ES/os-specific version of a Pascal program, you would specify

```
pc -DMIPS myprog.p -o myprog
```

To translate **myprog.p** into a source file **myprog.i** suitable for compilation on your old machine, you would use the **-P** option as follows:

```
pc -P -DMY_OLD_MACHINE myprog.p
rcp myprog.i my_old_machine:myprog.p
```

On most machines, including ESV Workstations, the **-D** option is unnecessary if you use a name that is automatically defined for you. ESV compilers predefine the following automatically.

```
mips
host_mips
MIPSEB
MIPSEL
LANGUAGE_C
LANGUAGE_PASCAL
LANGUAGE_FORTRAN
LANGUAGE_ASSEMBLY
LANGUAGE_PL1
LANGUAGE_COBOL
unix
SYSTYPE_BSD
SYSTYPE_SYSV
```

**Note:** Typically, you use **#ifdef mips** for differences that are hardware related or os related and **#ifdef MIPS** for differences due to other programs or preferences.

### Using the lint Program Checker

The **lint** program checker tries to find areas in the source code of C programs that are nonportable or that are likely to cause errors. See the **lint(1)** manual page for reference information. Here are some guidelines to follow when using **lint**.

- Instead of running **lint** on your source files one by one, run it a single time, specifying the names of all the source files. The **lint** command detects such problems as argument-list mismatches more thoroughly when it processes the entire source program at once.
- Use the same **-D** and **-I** options (if any) as when you compile.
- Analyze **lint** error or warning messages carefully before changing your code; make sure you understand why **lint** is creating the errors. For example, suppose **lint** indicates that a function result is incompatible with its use.

  ```
  double d;
  d = atof("1.23");
  ```

  You could satisfy **lint** by putting a cast in front of the function call

  ```
  d = (double) atof("1.23");
  ```

  but in fact you would be masking the problem rather than fixing it. The correct solutions are to either include **math.h** in your program or declare **atof** so that the compiler knows that it returns a **double** value rather than an **int** as follows:

  ```
  double d;
  extern double atof();

  d = atof("1.23");
  ```

## Memory Allocation

The interface to the C library memory allocator **malloc** is standard, but the implementation varies. ES/os uses the 4.3 BSD **malloc**, rather than the System V.3 **malloc**, because the former is significantly faster.

BSD **malloc** allows for allocation errors in that it rounds up the requested block size to a power of two, thus making programs still work that write more than they allocate. This allocation is fast, but it is inappropriate for large data block sizes.

Note that UNIX memory allocators use more memory than the programs request. If you plan to allocate memory in large chunks and never free them during execution, consider using **sbrk(2)**.

If you suspect a problem caused by memory allocation, try a different allocator and see if the problem disappears or changes. The following memory allocators are available in addition to the standard **malloc** version:

- An optional **malloc**, which you can obtain by specifying the **-lmalloc** option during compile/link edit.
- An additional allocator with routines **xmalloc**, **xfree**, and **xrealloc** resides in **/usr/lib/libp.a**. You can reference the routines using the **-lp** option during compile/link edit. This allocator's interface is identical with that of **malloc**, **free**, and **realloc**.

Even if using a different memory allocator solves the problem, you should still fix it to prevent a recurrence. Here are some approaches you can take.

1) Replace all calls to **malloc** and **realloc** with a wrapper routine that initializes the newly-allocated block (or the yet-unused portion of the reallocated block) to zero. If the problem disappears, look for code that erroneously assumes that newly allocated memory is initialized to zero.

2) Replace all calls to **malloc** and **realloc** with a wrapper that calls those routines, allocating one more byte than you ask for. If the problem disappears, this experiment may hide the problem by altering the order of blocks in memory. It is also likely that (in Pascal or FORTRAN) the program is confused about whether a character array originates at 0 or 1, or that (in C) the program did not leave space for the null byte that terminates a string.

3) Replace all calls to **malloc** and **realloc** with a wrapper that calls those routines, allocating four or eight more bytes than you ask for. If the problem disappears, then a zero-origin problem with an integer, real, or double-precision array may exist.

4) Experimentally replace all calls to **free** with an empty routine. If the problem disappears, the experiment may have masked the true problem by rearranging blocks in memory. However, dangling pointers to reused space may be causing the problem. Make sure that the program does not retain pointers to any data structure whose address may change due to a call on **realloc**.

## Signed chars

Like AT&T 3B compilers, but unlike most VAX and MC68000 compilers, the ES/os compilers interpret **char** to mean **unsigned char**. The **-signed** command-line option, however, reverses this.

To understand the consequences of unsigned characters, consider that the character `0xff` is not the same as `-1`; and a loop like

```
char c;
for (c = '\ '; c >= 0; c--) ...;
```

never terminates because the variable ***c*** can never be negative.

The ES/os C compiler, and others that have adopted features of the proposed ANSI draft standard, permit you to specify either **signed char** or **unsigned char** explicitly in a declaration. Alternatively, you can use masks or shifting to eliminate or propagate bits.

**lint** detects such problems by printing the diagnostic message `degenerate unsigned comparison.`

## Bitfields

For a bit field declaration within a structure, the ES/os C compiler uses signed or unsigned bitfields depending on your declaration. The Kernighan and Ritchie definition of the language permits a compiler to ignore these attributes and always use signed arithmetic or always use unsigned arithmetic; some compilers take advantage of this.

## short, int, and long Variables

On an ESV Workstation, a **short** variable is 16 bits wide; an **int** variable is 32 bits wide; and a **long** variable is also 32 bits wide. Some microcomputer compilers allocate only 16 bits for **int** and 8 bits for **short**, and some programs may rely on this. In general, manipulating 32-bit objects with the ESV architecture is as fast as or faster than manipulating 16-bit objects.

## Leading "_"

Like AT&T 3B compilers, and unlike the BSD UNIX VAX compiler, the ESV C compiler doesn't prepend an underscore to the name of a C-compiled symbol.

**varargs**

To improve performance, ES/os compilers pass certain procedure arguments in registers. This process is normally transparent to you, except for functions that use variable length argument lists. These lists must use the macros provided in **/usr/include/varargs.h** or **/usr/include/stdarg.h**. The functions must not assume that the arguments all appear in memory and can be accessed by taking the address of the first argument and incrementing it. Both the ANSI draft standard and the Kernighan and Ritchie definition of the language warn that programs attempting to implement variable argument lists without using **varargs** may not be portable.

Even **varargs** cannot deal with a situation where the argument list varies in type as well as length. Consider the following rather common practice of assuming that all C data types are equivalent for purposes of parameter-passing:

```
error(s, a, b, c, d, e)
char *s;
int a, b, c, d, e;
{
fprintf(stderr, s, a, b, c, d, e);
}

double d;

error("Value %g should be between %g and %g\en",
d, 1.2, 6.5);
```

The problem with this routine isn't that the variable argument list is variable, but rather that the routine declares arguments **a** through **e** as integers when in fact the routine plans to supply floating point numbers. This violates both the Kernighan and Ritchie definition of the language and the ANSI draft standard. In addition, it has dire consequences, because ESV architecture uses two separate sets of registers to pass integer and floating point arguments, and also because it imposes rules on the alignment of data types. The **fprintf** can accept variably typed arguments because it determines the types at execution time and references them appropriately; but the routine in the above example tells the compiler to emit a single version of "error" that always references them all as type **int**.

The following code fragment is the best method to use for a program being ported to ESV C. Any other system that implements **fprintf** using a routine called **vprintf** can also use this routine system.

```
/* VARARGS 1 */
void
error(s, va_alist)
char *s;
va_dcl
{
va_list ap;
va_start(ap);
vprintf(s, ap, stderr);
fputs("\en", stderr);
exit(1);
}

error("Value %g should be between %g and %g\en",
d, 1.2, 6.5);
```

However, a solution using a macro makes this routine more portable.

```
#define error(_s,_a,_b,_c,_d,_e) \
fprintf(stderr, _s, _a, _b, _c, _d, _e); \
exit(1);

error("Value %g should be between %g and %g\n",
d, 1.2, 6.5, 0, 0);
```

### **typedef** Names

ANSI C provides prototypes that in one instance conflict with Kernighan and Ritchie usage. ANSI C makes it illegal for a **typedef** name to appear in the argument list for a function definition. For example, in the following code:

```
typedef int P;
function(P);
{
}
```

the occurrence of **P** in the argument list is illegal since the compiler expects an identifier after the type "**P**." ESV C conforms to the ANSI standard in this case.

### Functions Returning **float**

Functions that are declared as returning **float** actually return **float** rather than **double** as in some older implementations of C. If the result is then used in a context requiring promotion to **double**, it is promoted after returning from the function call.

### Casting

Casting is not permitted on the left hand side of an assignment. If you are porting a program that currently runs on a Sun Workstation, you may have problems with this because Sun allows casting on the left hand side.

### Dollar Sign in Identifier Names

The dollar sign ($) is not a legal character in an identifier name. Because VAX and Sun compilers allow you to use $ as a legal character, ES/os provides the command line argument

```
-Wf,-Xdollar
```

### Additional Keywords

ESV Workstations will eventually conform to the ANSI C standard; therefore, the compiler treats **const**, **signed**, and **volatile** as keywords.

### Unsigned Pointers

The ES/os compilers treat pointers as unsigned rather than signed integers. For example, the following code,

```
extern  char  * sbrk();
char *p
p = sbrk(4090)

if (p < 0) error("out of memory");
```

does not work as expected because ES/os compilers use unsigned pointer comparisons, and nothing unsigned is less than zero. The **sbrk** routine does not work because it returns -1 if it fails. The proper way to test for failure is

```
if (p == (char *) -1) error("out of memory");
```

## Pascal Programs

ESV Pascal conforms to the IEEE standard, which is similar to the original Wirth-Jensen report, rather than to the ISO standard. It also provides a number of extensions, but not UCSD string support or ISO conforming arrays. See appendix B in the MIPS *Language Programmer's Guide* for more information on Pascal extensions.

If you wish to maintain your program on both an old system and on an ESV Workstation, refer to the "Using the C Preprocessor" subsection in the "C Programming Language" section of this chapter.

### Supported **pc** Options

The ES/os **pc** command supports the following options among others:

| Option | Description |
|---|---|
| **-c** | Suppresses linking with **ld** and produces a **.o** file for each source file. |
| **-C** | Generates code for runtime range checking. |
| **-cord** | Runs the procedure rearranger on the resulting file after linking. |
| **-cpp** | Runs the C macro preprocessor on Pascal and assembly source files before compiling. |
| **-D*name =def*** | |
| **-D*name*** | Defines the ***name*** to the C preprocessor, as if by **#define**. If no definition is given, the name is defined as **1**. |
| **-E** | Runs the source file through the C preprocessor only. |
| **-feedback *file*** | Specifies ***file*** to be used as a feedback file. |
| **-g[0-3]** | Produces additional symbol table information for **dbx** where 0 is no debug info, 1 and 2 are intermediate info, and 3 is the maximum information and compiler optimization. |
| **-G num** | Specifies the maximum size of a data item to be accessed from the global pointer. |
| **-I** | The standard directory is never searched for **#include** files. |
| **-Idir** | **#include** files whose names do not begin with "/" are always sought first in the directory of the file argument, then in directories specified in **-I** options, finally in the standard directory. |
| **-lx** | Links with object library **libx.a**. |
| **-j** | Compiles the specified source programs, and leaves the ucode object file output in corresponding files suffixed with **.u**. |
| **-ko *output*** | Names the output file created by the ucode loader as ***output.*** |
| **-k** | Passes options that start with a **-k** to the ucode loader. |
| **-L*directory*** | Adds ***directory*** to list of directories containing ob- |

| | |
|---|---|
| | ject-library routines (for linking using **ld**). |
| **-nocpp** | Does not run the C macro preprocessor on Pascal and assembly source files before compiling. |
| **-o *outputfile*** | Names the output file ***outputfile***. |
| **-Olimit *num*** | Specifies the maximum size of a routine that will be optimized by the global optimizer. |
| **-O[0-3]** | Optimizes the object code. Level 0 is no optimization, 1 is the default, and 3 is the maximum. |
| **-p [0-1]** | Compiles for profiling with **prof.** 0 is no profiling, 1 is profiling. |
| **-P** | Runs the source code through the C preprocessor only. |
| **-systype *name*** | Uses the named compilation environment ***name***. Legal names are **bsd43, sysv,** and **posix**. |
| **-S** | Assembles the program and produces an assembly source file with a **.s** suffix. |
| **-std** | Has the compiler produce warnings for things that are not standard in the language. |
| **-U*name*** | Removes a symbol ***name*** from the C preprocessor. |
| **-v** | Prints the passes as they execute. |
| **-V** | Prints the version of the driver and the versions of all passes. |
| **-w** | Suppresses warning messages. |

## Supported ld Options

The ES/os **ld** command supports the following options among others. For more information on **ld**, consult the **ld** reference manual page.

| Option | Description |
|---|---|
| **-A *file*** | Specifies incremental loading. |
| **-b** | Does not merge the symbolic information for a file into one entry for that file. |
| **-B *hex*** | Sets the bss segment origin. |
| **-d** | Forces common storage for uninitialized variables and other common symbols. |
| **-dc** | Performs **-d** and copies initialized data referenced by program from shared objects. |

| | |
|---|---|
| **-dp** | Forces an alias definition of undefined procedure entry points. |
| **-D *hex*** | Pads the data segment with zero-valued bytes to make it ***hex*** bytes long. |
| **-e *entry*** | Defines the entry point. |
| **-f *fill*** | Sets the fill pattern. |
| **-G *num*** | Sets the maximum size of a **.comm** item or literal to be allocated in the small bss section. |
| **-kl*x*** | Searches a library **lib*x*.b** where ***x*** is a string. |
| **-K*dir*** | Changes the default directories to the single directory ***dir***. |
| **-l*x*** | Abbreviates library name **lib*x*.a**. |
| **-L** | Changes search algorithm so it never looks in the default directories. |
| **-L*dir*** | Adds ***dir*** to the list of directories in which to search for libraries. |
| **-m** | Produces a map of the input/output sections on the standard output. |
| **-M** | Produces a primitive load map. |
| **-n** | Makes the text portion read-only and shared among all users executing the file when the output file is executed. |
| **-N** | Does not make the text portion read-only or sharable. |
| **-o *name*** | Uses ***name*** as the name of the **ld** output file, instead of **a.out.** |
| **-p *file*** | Preserves the symbol names listed in ***file*** when loading ucode object files. |
| **-r** | Generates relocation bits in the output file so that it can be the subject of another **ld** run. |
| **-s** | Strips the output by removing the symbol table and relocation bits to save space. |
| **-S** | Strips the output by removing all symbols except locals and globals. |
| **-T [text] *hex*** | Starts the text segment at location ***hex***. |

| | |
|---|---|
| **-Tdata *hex*** | Starts the data segment at location ***hex***. |
| **-u *name*** | Enters ***name*** as an undefined symbol. |
| **-x** | Preserves only global (non-**.globl**) symbols in the output symbol table. |
| **-y*sym*** | Indicates each file in which ***sym*** appears, its type, and whether the file defines or references it. |
| **-v** | Sets verbose mode. |
| **-V** | Prints the version of **ld** being used. |
| **-z** | Arranges for the process demand paged from the resulting executable file. |

### Runtime Checking

When possible, compile your program with runtime-checking using the **-C** option. This generates code which checks that subscripts don't exceed the range specified for them in the program. Storing one byte past the end of an array of characters may be harmless on one system if the compiler decides not to use the byte for anything but may cause an execution error on another system if a compiler decides to store something such as a subroutine return address there.

### Pascal Dynamic Memory Allocation

The ESV Pascal compiler responds to the much-requested dynamic allocation extensions to IEEE Pascal. The compiler provides a new generic data type, **pointer**, which is type-compatible with any standard Pascal pointer type.

The new capability does not allow you to directly take the address of an arbitrary variable or directly dereference a generic pointer. However, you can take the address of any object in the Pascal heap, or you can use the C library function **malloc** to return a generic pointer. Once you have a generic pointer containing the desired address, you can use any Pascal pointer type as a template to dereference that pointer.

Here is an example of one approach that uses **malloc**:

```
/* Declare interface to C library function for
dynamic allocation */

function malloc(number_of_bytes: integer): pointer;
extern;

/* Declare interface to C library function for rapidly
setting a block of memory to a fixed value */
procedure memset(destination: pointer; value: char;
number_of_bytes: integer); extern;
```

```
/*Two examples:a string, and an array of real numbers*/
type
   big_char_array = packed array [0 ..  maxint] of char;
   string = record
      length: integer;
      data: ^big_char_array;
      end;
   big_real_array = packed array [0 ..  maxint] of real;
   matrix2d = record
      rows, columns: integer;
      data: ^big_real_array;
      end;

var
      s: string;
      m: matrix2d;
      i, j: integer;
      Begin

/* To read a string of length "i" from the input: */

   s.length = i;
   s.data = malloc(i * sizeof(char));
   if s.data = nil then
   ...handle allocation error here...
   for j := 0 to i - 1 do
   begin;
      s.data^[j] := input^;
      get(input);
      end;

   m.rows := 5;
   m.columns := 7;
   m.data := malloc(m.rows * m.columns *
                       sizeof(real));
   if m.data = nil then
   ...handle allocation error here...
                    /* Clear the array */
   memset(m.data, chr(0), m.rows * m.columns *
                       sizeof(real));
   for i := 0 to m.rows - 1 do
   for j := 0 to m.columns - 1 do
   m.data^[i * m.columns+ j] := 1.0;
```

You should refrain from using the generic-pointer facility with variables which lie in local or global memory rather than in the heap or the **malloc** area.

For example, while the following trick does permit you to take the address of any character array, it is unsafe when used with ordinary local or global variables.

```
In one module:

function char_addr(p: pointer): pointer;
begin
      char_addr := p;
end;

In other modules:

function char_addr(var c: char): pointer; extern;
function mung_strings(p, q: pointer); extern;
var
   x: packed array [1 ..  10] of char;
   y: packed array [1 ..  100] of char;
   p, q: pointer;

Begin
   p := char_addr(x[1]);
   q := char_addr(y[1]);
   mung_strings(p, q);
   end
```

## FORTRAN Programming Language

This section describes ESV-FORTRAN, which contains full American National Standard (ANSI) Programming Language FORTRAN (X6.9-1978) plus ESV extensions that provide partial VMS FORTRAN compatibility. ESV-FORTRAN also contains extensions that provide partial compatibility with programs written in SVS FORTRAN and FORTRAN 66.

See the *MIPS-FORTRAN Language Reference* and the *MIPS-FORTRAN User's Guide* for more information on language extensions.

If you wish to maintain your program on both an old system and on an ESV Workstation, read the "Using the C Preprocessor" subsection in the "C Programming Language" section of this chapter.

### Supported f77 Options

The ES/os **f77** command supports the following options among others:

| Option | Description |
|---|---|
| **-66** | Reports non-FORTRAN 66 constructs as errors. |
| **-automatic** | Places local variables on the runtime stack. |

| | |
|---|---|
| **-c** | Suppresses linking with **ld** and produces a **.o** file for each source file. |
| **-cord** | Runs the procedure-rearranger on the resulting file after linking. |
| **-cpp** | Runs the C macro preprocessor on any FORTRAN source files before compiling. |
| **-C** | Generates code for runtime subscript range checking. |
| **-D*name=def***<br>**-D*name*** | Defines the ***name*** to the C preprocessor, as if by **#define**. If no definition is given, the name is defined as 1. |
| **-E** | Runs only the C macro preprocessor on the files, and sends the result to the standard output. |
| **-EB** | Produces object files targeted for big-endian byte ordering. |
| **-feedback *file*** | Specifies ***file*** to be used for feedback. |
| **-F** | Applies the C preprocessor to relevant files. |
| **-g** | Produces additional symbol table information for **dbx**. |
| **-i2** | Makes the default size of integer and logical constants and variables short (two bytes). |
| **-i4** | Makes the default size of integer and logical constants and variables four bytes. |
| **-k** | Passes options that start with a **-k** to the ucode loader. |
| **-ko *output*** | Names the output file created by the ucode loader as ***output***. |
| **-Ldir** | Adds ***dir*** to the list of directories containing object-library routines. |
| **-m** | Applies the M4 preprocessor to each **EFL** or **RATFOR** source file. |
| **-nocpp** | Does not run the C macro preprocessor on any FORTRAN source files before compiling. |
| **-N[cdlnqsx] *nnn*** | Makes static tables in the compiler bigger. |
| **-o *output*** | Names the final output file ***output*** instead of |

| | |
|---|---|
| | **a.out.** |
| **-O0** | Turns off all optimizations. |
| **-O1** | Turns on all optimizations that can be done quickly. |
| **-O2, -O3** | Turns on more optimizations. |
| **-Olimit *num*** | Specifies the maximum size of basic blocks that will be optimized by the global optimizer. |
| **-p0** | Does not permit any profiling. |
| **-p1** | Sets up for profiling by periodically sampling the value of the program counter. |
| **-P** | Runs only the C macro preprocessor and puts the result for each source file in a corresponding **.i** file after being processed by appropriate preprocessors. |
| **-static** | Causes all local variables to be statically allocated. |
| **-std** | Produces warnings (from the compiler) for things that are not standard in the language. |
| **-systype *name*** | Uses the named compilation environment ***name***. This has the effect of changing the standard directory for **#include** files, the runtime libraries, and where runtime libraries are searched for. |
| **-S** | Compiles the named programs. |
| **-u** | Makes the default type of variables ***undefined***. |
| **-U** | Does not convert upper-case letters to lower-case. |
| **-U*name*** | Removes any initial definition of ***name***. |
| **-v** | Prints the name of each pass as the compiler executes. |
| **-w** | Suppresses all warning messages. |
| **-w1** | Suppress warnings about unused variables. |

## Supported **ld** Options

The ES/os **ld** command supports the following options among others. For more information on **ld**, consult the **ld** reference manual page.

| Option | Description |
|---|---|
| **-A** ***file*** | Specifies incremental loading. |
| **-b** | Does not merge the symbolic information for a file into one entry for that file. |
| **-B** ***hex*** | Sets the bss segment origin. |
| **-d** | Forces common storage for uninitialized variables and other common symbols. |
| **-dc** | Performs **-d** and copies initialized data referenced by program from shared objects. |
| **-dp** | Forces an alias definition of undefined procedure entry points. |
| **-D** ***hex*** | Pads the data segment with zero-valued bytes to make it ***hex*** bytes long. |
| **-e** ***entry*** | Defines the entry point. |
| **-f** ***fill*** | Sets the fill pattern. |
| **-G** ***num*** | Sets the maximum size of a **.comm** item or literal to be allocated in the small bss section. |
| **-I** | **#include** files are never searched for in the standard directory. |
| **-j** | Compiles the specified source programs and leaves the ucode object file output in corresponding files suffixed with **.u**. |
| **-kl*x*** | Searches a ucode library **lib*x*.b** where ***x*** is a string. |
| **-K*dir*** | Changes the default directories to the single directory ***dir***. |
| **-l*x*** | Abbreviates library name **lib*x*.a.** |
| **-L** | Changes search algorithm so it never looks in the default directories. |
| **-L*dir*** | Adds ***dir*** to the list of directories in which to search for libraries. |
| **-m** | Produces a map of the input/output sections on the standard output. |

| | |
|---|---|
| **-M** | Produces a primitive load map. |
| **-n** | Makes the text portion read-only and shared among all users executing the file when the output file is executed. |
| **-N** | Does not make the text portion read-only or sharable. |
| **-o *name*** | Uses ***name*** as the name of the **ld** output file, instead of **a.out**. |
| **-p *file*** | Preserves the symbol names listed in ***file*** when loading ucode object files. |
| **-r** | Generates relocation bits in the output file so that it can be the subject of another **ld** run. |
| **-s** | Strips the output by removing the symbol table and relocation bits to save space. |
| **-S** | Strips the output by removing all symbols except locals and globals. |
| **-T [text] *hex*** | Starts the text segment at location ***hex***. |
| **-Tdata *hex*** | Starts the data segment at location ***hex***. |
| **-u *name*** | Enters ***name*** as an undefined symbol. |
| **-x** | Preserves only global (non-**.globl**) symbols in the output symbol table. |
| **-y*sym*** | Indicates each file in which ***sym*** appears, its type, and whether the file defines or references it. |
| **-v** | Sets verbose mode. |
| **-V** | Prints the version of **ld** being used. |
| **-z** | Arranges for the process demand paged from the resulting executable file. |

### Static Versus Automatic Allocation

For fastest program execution, the FORTRAN compiler uses **-automatic** allocation by default. If your program requires static allocation, you could use the **-static** driver option when you compile, however, program execution speed is sacrificed in most cases. A better solution is to use the ANSI FORTRAN 77 **SAVE** statement to specify the particular variables that must be statically allocated to make the program work correctly.

One symptom of a program that needs to use the **-static** option is repeated program failures because uninitialized local variables are used.

Neither ANSI FORTRAN 66 nor FORTRAN 77 permits a program to assume that local variables are automatically initialized to zero, or that local variables retain their values from the time a subroutine returns until the next time that subroutine is invoked.

Many older compilers use static allocation; that is, they allocate a location in global memory for each local variable in each subroutine. Because each local variable has its own fixed location, it starts out with a value of zero and retains its value even when the subroutine that declared it is not active. Applications on various systems often make use of this inadvertently.

Automatic allocation uses a stack to implement local variables. It has several advantages.

First, because current local variables reside near the current stack pointer, the compiler can address them with short-offset load and store instructions, which execute more rapidly than large-offset instructions.

Second, local variables get popped from the stack when a subroutine returns, the total memory required for the program is less, and subroutines which are never active simultaneously can share memory for their local variables.

Third, automatic allocation permits the global optimizer to more effectively allocate local variables to registers within a subroutine. This is because the optimizer does not need to do either of the following:

- load the initial values of the variables from global memory at the start of the subroutine,
- restore their final values to memory when the subroutine returns.

### Retention of Data

ESV-FORTRAN does not support the retention of data passed as parameters in previous calls to different entry points of a subroutine. This effect is not allowed by the FORTRAN standard and is error prone. However it is supported by some FORTRAN implementations and is required by some FORTRAN programs. Consider this example: your program calls an entry point to a subprogram with certain arguments, it then calls the subprogram again to a different entry point or the subprogram itself, the second call assumes that the arguments to the first call remain valid. ESV-FORTRAN does not support this usage. However, you can do one of the following to retain the data.

- set the arguments to local variables in the subprogram and use the **-static** switch to retain the values of the local variables,
- place the variable in a global common.

### Variable Length Argument Lists

ESV-FORTRAN does not support variable length argument lists, so your program can't call a routine the first time with fifteen arguments and a second time with two arguments.

### Runtime Checking

Compile your program with runtime-checking using the **-C** option. The **-C** option generates code to check that subscripts do not exceed the range specified in the program. Performance is adversely affected. Once the program is debugged, removing the **-C** option solves this problem. Storing one byte past the end of an array of characters may be harmless on one system if the compiler decided not to use the byte for anything. However, an execution error may occur on another system if the compiler tries to store something such as a subroutine address. This does not work if array parameters are declared as one element, which is common in older programs. To get around this, use the FORTRAN 77 "*" declaration.

### Alignment of Data Types

ESV Workstation architecture imposes certain rules governing how data may be aligned in memory. Basically, a variable of size ***n*** bytes must be aligned on a boundary whose address is a multiple of ***n*** bytes, up to a maximum of 8 bytes. For example, because a half-word occupies two bytes, its address must be a multiple of two.

High-level languages also impose rules about where you can assume data appears in memory. In most cases, the language rules forbid the same things that the architecture forbids.

Occasionally, the rules conflict. For example, the ANSI X6.9-1978 standard for FORTRAN explicitly permits certain double-precision (8-byte) variables to lie on the same boundary as any real (4-byte) variable; but the ESV Workstation architecture requires the double-precision variable to be aligned on an 8-byte boundary.

The ESV compiler system supports the alignment rules imposed by each of its languages, even when they are more permissive than the architecture. FORTRAN, for example, deliberately avoids performing double-word load or store operations on certain double-precision variables.

Some extensions such as **Integer*2**, which are not part of any language standard, cause problems. For example, consider the following common block:

```
common /x/i, j, k, l
integer*2 j, l, q(6)
```

```
integer*4 i, k
equivalence (q(1), i)
```

The compiler normally inserts a half-word of padding between ***j*** and ***k*** to conform to alignment rules, but that prevents ***q(6)*** from lying atop ***l***.

Modifying your programs to align data according to the rules of the ESV architecture improves their performance. In the previous example, reversing the order of ***j*** and ***k*** within the common block eliminates the need for padding at the cost of changing the relationship between the array ***q*** and the scalar variables.

Rearranging the order of variables within a common block is not practical. However, you can use certain "hidden" options of the compiler system to generate code which tolerates misalignment but degrades performance. When uncertain if an object will be misaligned, the compiler generates slower code sequences.

You may choose one of the following three options to deal with various degrees of misalignment:

- **-align8**

  Permits objects larger than 8 bits to be aligned on 8-bit boundaries. This option requires the greatest amount of space; however, it is the most complete solution; 16-bit padding is not inserted for **Integer*2** objects within common blocks.

- **-align16**

  Permits objects larger than 16 bits to be aligned on 16-bit boundaries; 16-bit objects must still be aligned on 16-bit boundaries (MC68000-like alignment rules); 16-bit padding is not inserted for **Integer*2** objects within common blocks.

- **-align32**

  Permits objects larger than 32 bits to be aligned on 32-bit boundaries; 16-bit objects must still be aligned on 16-bit boundaries, and 32-bit objects must still be aligned on 32-bit boundaries. This option requires the least amount of space, but isn't a complete solution; 16-bit padding is inserted for **Integer*2** objects within common blocks.

You should also use the following option no matter which above option you choose, unless experimentation proves this impossible:

**-align_common** Assumes that all common blocks are aligned properly, even though objects within the common blocks may be misaligned. This option generates better code. Without it, the assembler assumes that all global objects in lan-

guages like C and FORTRAN may be misaligned, even though they appear to be aligned, because they might be aliased against initialized objects in other modules to force the link editor to misalign them.

Pass these options specifically to the FORTRAN and assembly phases of the compiler system, by preceding them with **-Wfb**, as shown:

```
f77 -Wfb,-align_common,-align16 ...
```

There are problems which are not solved by these options. For example, your program must not perform I/O directly on misaligned objects or perform any other operation which requires passing them by reference to runtime library routines, that have not been compiled with the **-align** flags.

You can circumvent this problem by copying misaligned objects to or from aligned temporaries before performing I/O. If the misaligned data is accessed only within libraries, and not by the kernel, you can circumvent the problem by using a runtime fix-up package which traps unaligned references and repairs them dynamically. See the **unaligned(3)** manual page for more information.

Keep in mind that trapping is expensive in terms of execution time.

### Inconsistent Common Block Sizes

ANSI FORTRAN requires that a named common block has the same size but not necessarily the same constituent variable each time it occurs in a program. However, programs often declare only the amount needed, thus making the length of the common block vary. For example:

```
subroutine foo
common /gdata/ theta
...
end
subroutine bar
common /gdata/ theta, omega, radius
...
end
```

The FORTRAN compiler allows uneven block sizes when possible by allocating the space required by the largest instance of the common block. If, however, the varying size causes one instance of a common block to fall below the **-G** threshold while another instance of the same common block is too big to fit into the **$gp** area, a problem results. At best the link editor prints error messages and the compiler system makes less than optimal use of the **$gp** area. At worst, a falsely small instance of the common block causes the compiler to overflow the **$gp** area.

Use the link editor to report conflicting common block sizes by taking the name of each common block, converting it to lower case, prepending a **-y**, and

appending an underscore( _ ). When you link your program, pass the above names to the link editor. The names must precede the first object file specified in the command line. For example, if the common blocks are named `gdata`, `rsb31`, and `xtrnls`, then type in

```
f77 -ygdata_ -yrsb31_ -yxtrnls_ *.o -o myprog
```

The link editor reports the size that each common block has every time it occurs in an object file. The link editor also reports additional information about each common block; however, for common block problems, only size matters:

```
stdrfl.o: definition of common rsbg31_ size 1012
arstdr.o: definition of common rsbg31_ size 4
```

### Multiple Initializations of Common Blockdata

ANSI FORTRAN 77 requires that you use a **DATA** statement on a named common block only within a block data subprogram. An ordinary subroutine may initialize only local variables, not common variables; the ESV compiler system does not enforce this restriction.

The ESV compiler does enforce the ANSI FORTRAN 77 restriction which requires that you initialize each common block within exactly one subprogram. A variety of messages appear when you violate this restriction, including error messages from the link editor citing multiply defined symbols or messages from earlier phases of the compiler citing `illegal init` or `illegal space`. For example:

```
ugen: internal : line 6345 : ../symbol.p, line 270
illegal inits
```

To diagnose such problems, use the utility **fsplit** to split your program into many small files, with one subprogram per file. Then link the program and collect the multiply defined messages in a file. For each multiply-defined symbol, prepend a **-y**, and relink the program with these options preceding the list of object files in the command line. For example, if the link editor issues error messages for `gdata_` and `rsb31_`, then relink with:

```
f77 -ygdata_ -yrsb31_ *.o -o myprog
```

The link editor uses the phrase `definition of external data` every time an object file initializes a symbol:

```
bstimr.o: definition of external data gdata_
zuxeng.o: definition of external data rsb31_
stdrfl.o: definition of common rsbg31_ size 1012
cmflol.o: definition of external data rsb31_
cmflow.o: definition of external data gdata_
arstdr.o: definition of common gdata_ size 4032_
```

The phrase `definition of common` can appear repeatedly for a particular common block, but the phrase `definition of external data` must appear only once for each common block.

Once you realize that two **.o** files are initializing the same common block, transfer the appropriate **DATA** statements from one to the other (or, preferably, to a blockdata subprogram), then recompile and relink.

## Endianness and integer*2

Special problems exist for porting FORTRAN programs between big- and little-endian machines in addition to those discussed in the "Uninitialized Variables" section of the "Hardware-Related Considerations" section. Although FORTRAN programs pass arguments by reference (they pass the address of the argument rather than the argument itself), they cannot declare the formal arguments of a subroutine. Consider the following call:

```
call foo(0.314159e1, 0.628318d1, 1234, 2468)
```

Clearly the first argument is type real or **real*4,** and the second argument is type double precision or **real*8**. But the types of the third and fourth argument, which can be either **integer*2** or **integer*4**, are unknown to the compiler. Thus, the compiler allocates four bytes for each of these variables.

On a big-endian machine such as the ESV Workstation, where the address of an integer is the address of its high-order byte, this code fails: if a four-byte integer is passed to a subroutine which expects a two-byte integer, then the subroutine recognizes only the two upper bytes of the four-byte integer. On a little-endian machine, where the address of an integer is the address of its low-order byte, this code works correctly even if subroutine **foo** expects the arguments to be **integer*2**, because the address is the same in either case.

There are two solutions:

- If all of the formal arguments in your program are two-byte integers, and you also wish the compiler to use two-byte integers wherever you have declared variables as **integer** rather than **integer*4**, then you can use the **-i2** option when you compile your program, and all literal integers will use only two bytes.
- If it is not possible to use **-i2**, then you must use temporary variables of type **integer*2** to pass literal numbers to two-byte arguments:

```
integer*2 temp1, temp2

temp1 = 1234
temp2 = 2468
call foo(0.314159e1, 0.628318d1, temp1, temp2)
```

## FORTRAN and C Interface

This section discusses items you should consider before writing a call between FORTRAN and C.

### Names

In calling a FORTRAN subprogram from C, the C program must append an underscore ( _ ) to the name of the FORTRAN subprogram. For example, if the name of the subprogram is **matrix**, then call it by the name **matrix_**. When FORTRAN is calling a C function, the name of the C function must end with an underscore.

Note that only one main routine is allowed per program. The main routine can be written in either C or FORTRAN. Following is an example of a C and a FORTRAN main routine:

| C | FORTRAN |
|---|---|
| `main()   {` | `write(6,10)` |
| `printf("hi!\n"); 10` | `format('hi!')` |
| `}` | `end` |

### Invocations

Invoke a FORTRAN subprogram as if it were an integer-valued function whose value specifies which alternate return to use. Alternate return arguments (statement labels) are not passed to the subprogram but cause an indexed branch in the calling subprogram. If the subprogram is not a function and has no entry points with alternate return arguments, the returned value is undefined. The FORTRAN statement

```
call nret(*11,*2,*3)
```

is treated exactly as if it were the computed **goto**

```
goto (1,2,3), nret()
```

A C function that calls a FORTRAN subprogram can usually ignore the return value of a FORTRAN subroutine; however, the C function should not ignore the return value of a FORTRAN function. The table below shows equivalent function and subprogram declarations in C and FORTRAN programs:

| C Function Declaration | FORTRAN Declaration |
| --- | --- |
| `double dfort_()` | `double precision function dfort()` |
| `double rfort_()` | `real function rfort()` |
| `int ifort_()` | `integer function ifort()` |
| `int lfort_()` | `logical function lfort()` |

Note the following:

- Avoid calling FORTRAN functions of the following types from C:

  **float** **complex** **character**

- You cannot return complex types between C and FORTRAN.
- A character-valued FORTRAN subprogram is equivalent to a C language routine with two extra initial arguments: a data address and a length.

  Thus:

  ```
  character*15 function g(...)
  ```

  is equivalent to:

  ```
  char result[];
  long int length;
  g_(result, length, ...)
  ```

  and could be invoked in C by:

  ```
  char chars[15];
  g_(chars, 15);
  ```

## Arguments

The following rules apply to arguments passed between FORTRAN and C:

1) All arguments must be passed by reference. That is, the argument must specify an address rather than a value. Thus, to pass constants or expressions, their values must be first stored into variables and then the address of the variable passed.

2) When passing the address of a variable, the data representations of the variable in the calling and called routines must correspond, as shown in the following table. Note that FORTRAN requires that each **integer**, **logical**, and **real** variable occupy 32 bits of memory.

| FORTRAN | C |
|---|---|
| `integer*2 x` | `short int x;` |
| `integer x` | `long int x;` or just `int x;` |
| `logical x` | `long int x;` |
| `real x` | `float x;` |
| `double precision x` | `double x;` |
| `complex x` | `struct{float real,imag}x;` |
| `double complex x` | `struct{double dreal,dimag;}x;` |
| `character*6 x` | `char x[6];` |

3) The FORTRAN compiler may add items not explicitly specified in the source code to the argument list. The compiler adds the following items under the conditions specified:

- Destination address for character functions, when called.
- Length of a character string, when an argument is the address of a character string.

When a C program calls a FORTRAN subprogram, the C program must explicitly specify these items in its argument list in the following order:

1) Destination address for character function return value (if appropriate).
2) Normal arguments (addresses of arguments or functions).
3) Length of character strings. The length must be specified as an absolute value or **integer** variable. The next two examples illustrate these rules.

Example 1 shows how a C routine must specify the length of a character string (which is only implied in a FORTRAN call). **sam_** is a routine written in FORTRAN.

FORTRAN call to **sam_**

```
external f
character*7 s
integer b(3)
. . .
call sam(f, b(2), s);
```

C call to **sam_**

```
int f();
char s[7];
long int b[3];
. . .
sam_f(f, &b[1], s, 7);
```

Example 2 shows how a C routine can specify the destination address of a FORTRAN function (which is only implied in a FORTRAN program). **f*** is a function written in FORTRAN.

```
FORTRAN Call to f_
     external f
     character*10 f,g
     g = f()
C Call to f_
     char s[10]
     f_(&s,10);
Function f_
     character*10 function f()
     f = '0123456789'
     return
     end
```

## Array Handling

FORTRAN stores arrays in column-major order with the furthest left subscript varying the fastest. C, however, stores arrays in the opposite arrangement, with the furthest right subscripts varying the fastest, which is called row-major order. Here is how the layout of FORTRAN arrays and C arrays look.

<u>FORTRAN</u>

```
integer t(2,3)
t(1,1), t(2,1), t(1,2), t(2,2), t(1,3), t(2,30)
```

<u>C</u>

```
int t[2][3];
t[0][0], t[0][1], t[0][2], t[1][0], t[1][1], t[1][2]
```

Note that the default for the lower bound of an array in FORTRAN is 1, whereas in C it is 0.

When a C routine uses an array passed by a FORTRAN subprogram, the dimensions of the array and the use of the subscripts must be interchanged, as shown in the following example.

FORTRAN caller

```
   integer a(2,3)
   call p( a, 1, 3 )
   write(6,10)a(1,3)
10 format(1x,19)
   stop
   end
```

C Called Routine

```
void
p_( a, i, j )
int *i, *j, a[3][2];
{
 a[*j-1][*i-1] = 99;
}
```

The FORTRAN caller prints out the value 99. Note the following:

- Because arrays are stored in column-major order in FORTRAN and row-major order in C, the dimension and subscript specifications are reversed.
- In FORTRAN, the lower-bound default is 1, whereas in C it is 0; therefore, 1 must be subtracted from the indices in the C routine. Also, because FORTRAN passes parameters by reference, the ***j** and ***i** pointers are used in the C routine.

### Accessing Common Blocks of Data

The following rules apply to accessing common blocks of data:

- FORTRAN common blocks must be declared by common statements; C can use any global variable. Note that the common block name in C (**sam_**) must end with an underscore.
- Data types in the FORTRAN and C programs must match unless you desire equivalencing. If so, you must adhere to the alignment restrictions for the data types.
- If the same common block is of unequal length, the largest of the sizes is used to allocate space.
- Unnamed common blocks are given the name **_BLNK_**.

The following gives examples of C and FORTRAN routines that access common blocks of data.

C

```
struct S {int i; float j;}r_;
main()   {
   sam_();
printf("%d %f\n", r_.i, r_.j);
```

FORTRAN

```
subroutine sam()
common /r/i,r
i = 786
r = 3.2
return
```

The C routine prints out 786 and 3.2.

## ES/os Compiler Components

This section discusses considerations for debugging, programming checking, compiling, and link editing your programs; the following topics are included:

- Debugging Procedures

  You compile programs for debugging using the **-g** option of the driver command that compiles your program for full symbolic debugging when executed with the **dbx** debugger.

- Programming Checking

  Several program checking tools are available to check the correctness of your program.

- Optimization

  The optimizer can significantly improve the performance of your object program. The optimizer is invoked using one of the several **-O** options of the driver command. You should consider levels of optimization higher than the standard default once your program is successfully debugged.

- Link Editor Features

  Several link editor options and techniques should be considered. These options are invoked by either a driver command (**cc, pc, f77, pl1, cob**) or the link editor **ld** command.

In addition to the information provided in this section, you may need to refer to the *Languages Programmer's Guide* and the manual page for the driver, **dbx**, or **ld** in the *RISC/os User's Reference Manual.*

### Debugging

This section gives a suggested procedure to follow when debugging your ported program. For a complete description of the debugger **dbx**, refer to the *Languages Programmer's Guide.*

If a program fails and you wish to use **dbx** to debug the failed program, do the following:

1. Recompile the program using the following compiler options:

   - the **-g** debugging option, which causes the compiler system to generate the symbol table required by **dbx**.

   - the **-O1** optimizing options (the default), which causes the compiler system to minimally optimize the resulting object. (Once the program is successfully debugged, you may want to recompile it using a higher level of optimization.)

- the **-signed** and **-varargs** options (for C programs only).
- the **-static** option (for FORTRAN programs only).

2. Execute the program.
3. If a segmentation fault, bus error, or other error causes the program to default, use **dbx** to isolate the problem. Do a stack trace using the **dbx where** command to locate the point of failure.
4. If you know the approximate location of the problem, then do the following:
   - Use the **dbx stop** command to set a breakpoint just before the suspected problem location.
   - Use the **dbx where** command to display the current values contained in the pertinent variables.
   - Use the **dbx next** or **step** command to incrementally execute the instructions after the breakpoint. Display and check the values of the variables as you execute each instruction.
5. Use binary search techniques, as discussed in step 4, when you are trying to track down the source of corrupted data. You can also make a change to data or code to see what happens, but understand the code before you do this. For example, sometimes all you need to do is to check for the symptom that results in a problem, and bypass the code that would be executed. A classic example of this is programs that get segmentation faults for doing the following:

```
if(*sp=='a'){
 ...
}
```

   If **sp** is 0, then a segmentation fault occurs, but the code works as expected if it is changed to:

```
if(sp && *sp == 'a'){
 ...
}
```

**Program Checking**

A correct program is not necessarily a portable program as it may run successfully on one system, but not another. Debugging alone does not guarantee correctness. In fact, no tool can completely guarantee the correctness of a program; however, a few tools can help check whether a program is operating correctly. These tools are appropriate to use either when porting a program from another system to an ESV Workstation, or when writing a program on an ESV Workstation intended to be portable to other systems.

### lint

One such tool is **lint**, a static program checker for the C programming language. **lint** provides the sort of checking that is typically performed by compilers in other programming languages. Its use for C programs is highly recommended. See the *Language Programmer's Reference* for more information.

### Subscript Range Checks

Another tool is subscript range checking. It is not uncommon for a program to reference an array outside of the declared bounds. An error of this sort may go undetected if, for example, the location referenced exists, but is otherwise unused. When the program is ported to another system the incorrect reference may instead access a critical location, and the program will fail to operate correctly.

To detect subscript range errors, your program may be compiled with a special option that generates extra code to verify that the indexes to array references are within the declared bounds of the array. This option is available in Pascal and FORTRAN. It is the default in Ada. For C, the language and its style of use does not make subscript range checking useful, so no compiler option is provided.

A Pascal program compiled without subscript range checking would run:

```
% pc -q -o example example.p
```

However, if you compiled the same program with subscript checks, you would receive a subscript error during run time.

```
% pc -C -q -o example
% ./example
 Trace/BPT trap (core dumped)
```

At this point, you could use **dbx(1)** to locate the source line with the subscript range error.

The **-C** compile option also works for a FORTRAN program. Older FORTRAN programs require some modification to work with subscript range checking turned on. It was once common in FORTRAN to declare array parameters to have dimension 1 when the actual size was passed as a separate parameter:

```
subroutine zero(a, n)
real a(1)
do 10 i = 1, n
10    a(i) = 0
end
```

In FORTRAN 77 the declaration could correctly be written as:

```
real a(n)
```

or

```
real a(*)
```

### Dynamic Storage Allocation

Just as programs sometimes reference outside the bounds of an array, another common error is to call a dynamic storage allocator and reference outside of the allocated block. Since the compiler often does not know the size of the block when a pointer based reference is made, it cannot generate code to verify the access, as with subscript range checking. However, a special version of the standard dynamic storage allocation routines **malloc(), free(),** and **realloc()** called **malloccheck(1)** is available to check for incorrect uses of dynamic storage. Add **-lmalloccheck** to your link command line to use this version. It checks for the following:

- Writing beyond an allocated block.

  A common error is to write beyond the end of an allocated block. The **malloccheck** allocator allocates extra space both before and after the block it returns to you and initializes this space to special bit patterns. A write outside the block will usually affect these pattern words. When the block is freed, the pattern words are checked, and if modified a warning is given.

- Freeing a block twice.

  Another error is to free a block twice. **malloccheck** does not reuse storage after it is freed, but instead simply marks it as such. A second free to the same block generates a warning.

- Referencing a block after it is freed.

  Another error is to reference a block after it is freed. This often works because the freed storage is not immediately re-used. **malloccheck**'s **free** routine overwrites the data when it is free, which usually causes subsequent references to return unexpected results, leading to a detectable program failure later.

- Initializing allocated storage to zero.

  Some programs inadvertently assume the allocated storage is initialized to zero, even though the standard **malloc()** and **free()** routines do not guarantee this. **malloccheck** initializes the allocated storage to non-zero so that such assumptions lead to program failure.

**Malloccheck**'s primary checking is done when blocks are freed. An error may go undetected if a block is never freed, or if the error occurs after it is freed. Also, an inconsistency detected by **free** may be difficult to trace to an

error made long before. For all of these reasons, **mallocheck** provides the **malloc_status()** subroutine, which checks the entire dynamic storage allocation area. Calls to **malloc_status()** can be inserted in the program as necessary to locate the source of an error. During program development a single call to **malloc_status()** at the end of the program is useful. The argument to **malloc_status()** specifies the level of checking.

```
malloc_status(0);
```

checks for errors and prints some summary statistics. A level of 1

```
malloc_status(1);
```

checks for errors, prints some summary statistics, and lists all blocks that remain in use. This is useful for finding blocks that the program failed to free. Failure to free storage can lead to eventual memory exhaustion and program failure on a large run.

## Optimization

The optimizer is vulnerable to human error. For example, incorrectly specifying the size of a variable or the nature of a formal argument can cause problems. In the following Pascal code, the optimizer may move the **if** statement to precede the loop, since **name_changer** is declared to receive only one character, therefore **name[5]** cannot change during the loop.

```
type
      array5 = packed array [1 ..  5] of char;
var
      i: integer;
      name: array5;
procedure name_changer(var c: char); extern;

for i := 1 to 10 do
      begin
      if name[5] >'9' then goto 5;
      name_changer(name[1]);
      writeln(name);
      end;
```

This assumption is true if **name_changer** is coded in Pascal and the formal argument agrees with the actual argument. If it is coded in C, and the formal argument is **char *c**, then **name_changer** may alter **name[5]** during the loop. To solve this problem be specific. Don't specify **var c: char** if it is actually **var c: array5** from the point of view of the external procedure.

Similar problems arise in FORTRAN programs that assume declaring a formal argument or common block to be an array of one element is the same as declaring it specifically.

```
common /x/ ary(1)
call matset(ary)
```

If a common declaration in another program unit specifies **ary(100)**, then the variable **ary** becomes 100 elements large when you link the program; but in this particular section, the optimizer behaves as if the variable had only one element. This problem can be solved as follows:

- Use consistent common declarations.
- Use an ANSI FORTRAN 77 declaration in the form of **Integer parm(*)** rather than the traditional trick of **Integer parm(1)** when the size of a formal parameter may vary.

## The Link Editor

This section describes the special features of the link editor that you should be aware of when porting a program. For information on the link editor and its libraries, refer to the **ld(1)** manual page.

### The -G option

The ESV Workstation compiler system sets up one register called **gp** to point to a 64 Kbyte block of global memory that can be addressed in half the number of instructions required for a normal global access.

It allocates by default to the **gp** area any global variable up to a maximum size of eight bytes. You can change the default size using the driver **-G** option.

There are three kinds of **gp**-related problems.

1) The **gp** area overflows because **gp**-relative data doesn't fit into its allocated 64 Kbytes of memory.

   If this problem occurs, the link editor prints a prediction of the best value to use as a maximum size in the **-G** option. The best value places as many global variables as possible into the 64 Kbyte area to improve performance, but excludes enough variables to prevent the area from overflowing.

   However, the best value is merely a prediction and may not produce successful results. To make sure that no **gp** area overflow occurs, and to produce an executable object immediately, note the best value provided by the link editor, and then recompile and relink your program using the **-G 0** option. You can then move that copy to a safe place and recompile and relink using the recommended best value.

   If your program does not fail, but you want to improve performance, then use the **-bestGnum** option. This option causes the link editor to predict a best value. Recompile and relink with the new value. However, you should first debug the program at the default setting, save a working copy, and then experiment with the best number prediction.

2) A variable larger than the maximum specified size is in the **gp** area.

   This problem can happen when two program modules disagree about the data type of an object. For example, one program sees the data as a small variable and addresses it within the **gp** area, and the other sees it as a large variable.

   The link editor retains the larger size of the variable, when possible, and places it into the **gp** area with a warning error message. This may cause the **gp** area to overflow. If the **gp** area overflows, then use the **-G 0** option or (preferably) reconcile the conflicting declarations so as to retain the advantages of using the **gp** area for other variables.

   Sometimes the link editor cannot put the large variable into the **gp** area because it is a synonym for some other object that cannot be addressed relative to the **gp** register. If this is the case, you must reconcile the conflicting declarations. For example, suppose one module defines an object as a function, which cannot be addressed relative to the **gp** register

```
int foo();
bar(foo);
```

   and another defines it as a small data item.

```
int foo, *ptr;
ptr = &foo;
```

   Most inconsistently sized declarations are caused by a violation of the ANSI standard with regard to FORTRAN common blocks. See the "FORTRAN Programming Language" section in this chapter for details.

3) The link editor believes that the **gp** register isn't initialized.

   This problem can occur when you use your own start-up code, rather than the runtime start-up code in **crt0.o** or **crt1.o** provided when a compiler driver (**cc, f77, pc, cobol**, or **pl1**) links your program.

   The runtime start-up code loads a link editor-defined symbol called **_gp** into the **gp** register. If you use your own start-up code instead, load **_gp** into some register (**$0** is acceptable) even if you load **gp** with some other value that you have calculated yourself; otherwise, the link editor issues an error message.

   Two details may help you in reconciling inconsistently sized declarations:

   - If a common variable is declared but not referenced in a module, then the compiler allocates it outside the **gp** area regardless of its size. This allocation reduces possible problems. Therefore, you should explicitly initialize unreferenced variables to zero, to ensure that they are placed within the **gp** area.

- In C, you can force a scalar variable to be referenced as if it lay outside the **gp** area by declaring it to be an array of unspecified size and referencing the first element (for example, **int j[]**; and **j[0]** rather than **int j** and **j**).

### Forcing Library Extractions

The ESV Workstation compiler system link editor opens and searches only one library at a time in the order you specify. This can cause problems as the following example shows. Suppose you try to link a program **p.o** with two libraries, **l1.a** and **l2.a**, as follows:

```
cc -o p p.o l1.a l2.a
```

The components that the program and libraries contain or need are:

| File/Library | Contain | Imports/Exports |
|---|---|---|
| **p.o** | | **imports l2proc** |
| **l1.a** | **l1.o** | **export l1proc, import l3proc** |
| **l2.a** | **l2.o** | **export l2proc, import l1proc** |
| **l2.a** | **16.o** | **exports l3proc** |

When the program is compiled the following steps occur:

1) The link editor sees that it needs to import **12proc** for **p.o**.
2) It searches **l1.a** for **l2proc**, and does not find it.
3) The link editor closes **l1a** and opens **l2a**.
4) It finds the **l2proc** but cannot find the **l1proc** because **l1.a** is closed.

If you specify **l1.a** and **l2.a** in the opposite order, the link editor fails to obtain **l3proc**.

The standard UNIX solution to this problem in which you assemble a file **kludge.s** containing

```
.globl l1proc
```

and link **kludge.o** prior to **l1.a** to import **l1proc** does not work on the ESV Workstation compiler system. The ESV compiler assembler notices that **kludge.s** does not really use **l1proc**, and as an optimization removes the request to import it. To solve this problem, edit **kludge.s** so that it defines **l1proc**.

```
.extern l1proc
.data
.word l1proc
```

Simpler solutions are to

1) Correct the problem on the command line by having the link editor search the **l1.a** library twice:

```
cc -o p p.o l1.a l2.a l1.a
```

2) Extract the object file and directly include it in the command line.

```
ar x l1.a l1.o
cc -o p p.o l1.o l1.a l2.a
```

**The Semantics of a Library Search**

Some programs assume that the link editor searches linearly within a library for symbols that it wishes to import. The ESV compiler link editor libraries use a hashed symbol table for faster linking, so the order in which **.o** files are added to a **.a** file is insignificant.

The link editor does not consider a common declaration to be a request to import every module that issues an identical common declaration. For example, a declaration of `int errno` in a C-coded main program does not cause the link editor to import every module that similarly declares `int errno`; those modules are imported only if they specifically export some symbol that your program specifically imports using a function definition or initialized data definition.

However, a "common" in the library can satisfy an import request without actually adding the library module to the program. For example, if your main program declares `extern int errno`, the occurrence of `int errno` in a module **foo.o** in the library would create a common `int errno` in the linked program, without necessarily adding **foo.o** to the linked program. This rather exotic behavior makes our link editor compatible with the one provided by the standard BSD UNIX distribution.

**Libraries Versus Object Files**

If you want to bundle together a group of infrequently-changed object files because it is more convenient to specify a single name when you link, it is faster to use **ld -r** to bundle them into a **.o** file than to use **ar -r** to add them to a **.a** file.

## Helpful Hints

### 1. Enumerated Types

ES/os is restrictive in the uses of **enumerated type** variables. They should not be used as indices.

### 2. types.h

Source code that needs the **#include** file **types.h** should specify **<sys/types.h>** for ES/os.

### 3. Internal Compiler Tables

The ES/os **cc** command uses several different internal tables. Their sizes seem to be too small for large applications. ES/os provides an undocumented method of expanding the table sizes by passing a special argument with the **cc** command. The following arguments may resolve the problem:

```
-Wf, -XNd50000
-Wf, -XNp50000
```

### 4. ESV Workstation Input

All input to the ESV Workstation is handled through the Evans & Sutherland X Input and the X Picking Extensions, which are documented in the *ESV Workstation Reference Manual.*

If you want to program low-level graphics, you can use the **Xlib** routines, which are supplied with the X Window System. High-level graphics can be programmed with **Xt** toolkit, available with the X Window System, or the Motif toolkit, available from the Open Software Foundation.

### 5. Separate and Map Graphics Model

For graphics applications, you should separate the graphics application code from the more general purpose application code, and map the existing graphics code in your application to the PHIGS model.

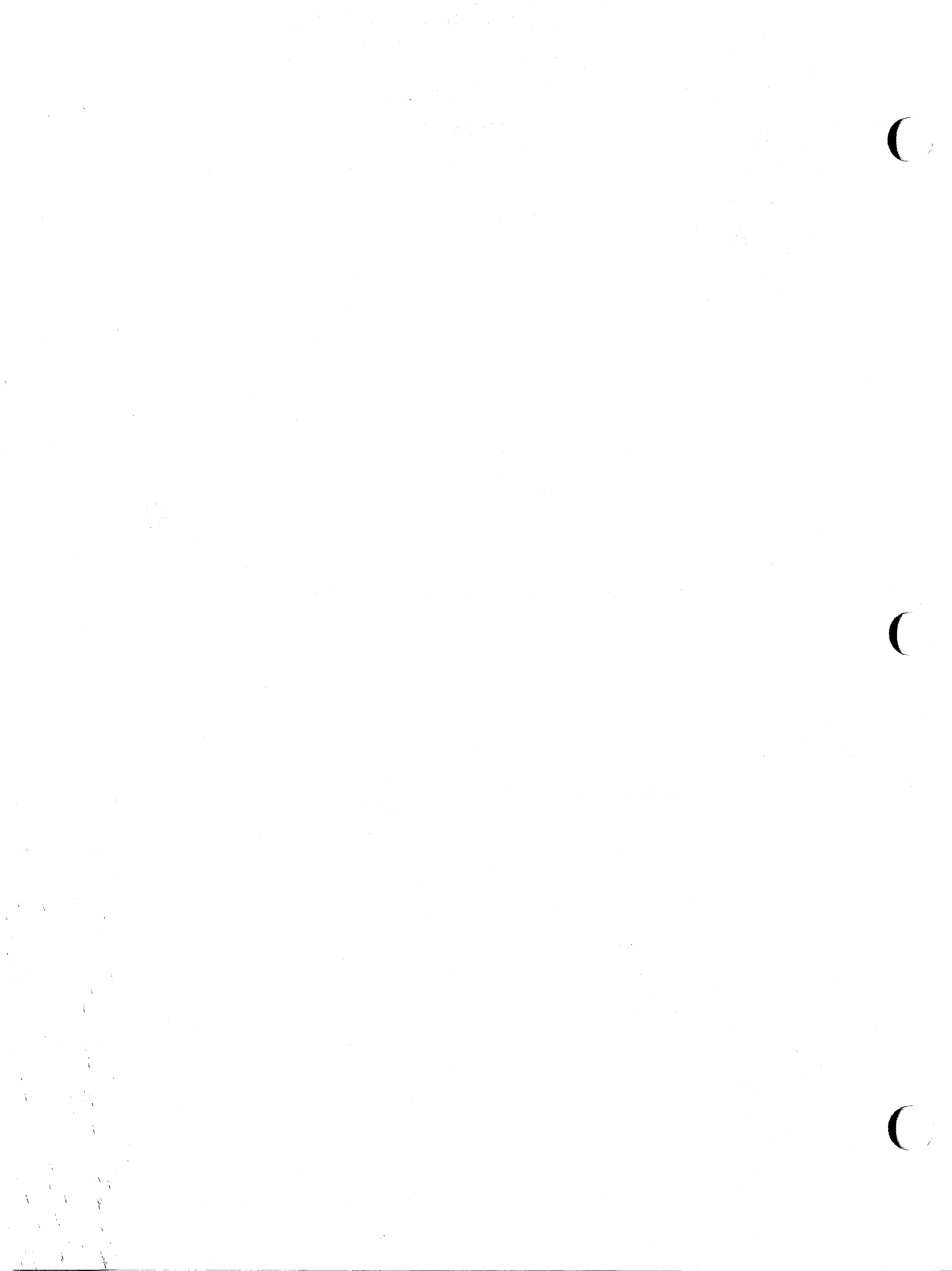
### 6. Compiling at -O2 Level

Compiling at the **-O2** level involves several sophisticated optimization algorithms. These algorithms must make certain assumptions with regard to your source code. Sometimes these assumptions are incorrect, and as a result, the compiler may generate incorrect code. Compile at this level or at **-O3** at your own risk.

## 7. BSD Enhancements to ES/os

The BSD enhancements added to ES/os fall into three main categories:

1) BSD user commands,
2) BSD or BSD-like system calls,
3) BSD or BSD-like kernel enhancements that are not reflected in either user commands or system calls. These include TCP/IP networking, the NFS remote file system, and the Berkeley FFS.



# Table of Contents

9. **Local Server** .......... **9-1**
Overview .......... 9-1
SCPU Card .......... 9-1
DPR Card .......... 9-1
Local Server Capabilities .......... 9-1
Power-on, Boot Sequence, and Shutdown .......... 9-2
Power-on Diagnostics .......... 9-2
Boot Sequence .......... 9-3
Server Console Switch .......... 9-4
Shutdown Procedure .......... 9-5
Effective System Administration .......... 9-7
Network Interface Addresses and Names .......... 9-8
ES/PSX .......... 9-8

# 9. Local Server

## Overview

The Local Server consists of the following two cards:

- SCPU card
- DPR card

## SCPU Card

The Local Server provides the ESV Workstation user with a second, fully-functional CPU card called the *Server CPU*, or *SCPU*. The SCPU supports all ESV Workstation hardware and software products supported by the standard CPU (called the *Graphics CPU*, or *GCPU*), with the exception of products that require the CPU to be directly connected to the ESV Workstation graphics hardware. The Local Server offers the ESV Workstation user a way to purchase an inexpensive network compute/file server.

## DPR Card

The Local Server also provides a shared memory module called the *dual-ported RAM card* (*DPR card*) that is accessible from the GCPU and the SCPU simultaneously. The DPR card is configured as a network device and is used as a high speed point-to-point network between the two CPUs. This high speed point-to-point network is called the *LCS network*.

The GCPU and SCPU can both be configured for full Ethernet network access. Access to the LCS network is similar to that of an Ethernet network. Any properly written X or PEX application that can be run over Ethernet can also be run using the LCS network.

Use of the LCS network requires no application source code modification. The LCS network supports all ES/os standard networking facilities, including all applications that use network protocols TCP/IP and UDP. Additionally, the LCS network supports diskless booting for the GCPU so that it may run diskless using the SCPU as its file server.

## Local Server Capabilities

Viewed from a network perspective, the SCPU can be thought of as a compute server, or a file server, or both. Its use is no different from any other compute or file server on the network, and its use does not detract from the performance of the GCPU.

The same view also applies to the user of the GCPU. However, the GCPU user has the option of using the LCS network, which is much faster than Ethernet. Consequently, the GCPU user may run applications over the network that might otherwise have been considered too slow.

The SCPU may be used to off-load compute- or I/O-intensive tasks that would otherwise cut down interactive performance on the user's CPU, or to improve GCPU display performance. In situations where the graphics server is consuming more than half of the GCPU time, it may be an advantage to run the client side of the application on the SCPU.

## Power-on, Boot Sequence, and Shutdown

This section describes the function/sequencing necessary to boot both CPUs (GCPU and SCPU) in a dual-CPU system. It also describes the proper method for shutting down such a system.

There are several issues that the PROMs must deal with, including:

- What happens after power-on,
- What to do with power on test failures,
- How to handle system administration problems, and
- How to get the system booted.

### Power-on Diagnostics

At this point the system is running the power-on (PON) confidence tests and must be made aware of the other CPU. The PON confidence tests run to completion on one board and then the other. First, the GCPU runs the tests. When successfully completed, the GCPU tells the SCPU to start its tests.

The output indicates exactly what happens for both CPUs in the PON sequence including any failure(s) that may have occurred. The text from the SCPU is communicated to the GCPU via the VMEbus.

If there is an error, the **FAILED** message is printed, and the auto boot sequence is disabled if it was previously enabled. This is the same behavior that currently happens when a PON confidence test fails.

## Boot Sequence

After the PON confidence tests are complete, the boot prompt is displayed. For a dual-CPU system, the prompt contains the CPU name as a prefix followed by the usual "**>>**" so the CPU executing the commands is always known.

The system is left (if not set to autoboot) with control going to the SCPU. Normally, you need to boot the SCPU first.

```
MIPS Monitor 4.10
...
SCPU>>
```

All the PROM commands plus one new command are available from the PROM prompt. This new command, **scpu** or **gcpu**, allows you to switch the command stream between the two CPUs. This is necessary since you must boot the SCPU first and then boot the GCPU.

For a normal boot (*i.e.*, not doing system administration tasks and not setting the system to autoboot), follow this procedure:

```
SCPU>> auto
loading sash
...
loading unix
...

The system is ready

GCPU>> auto
loading unix
...
The system is ready

Console login:
```

The above sequence consists of booting the SCPU and then the GCPU. If the GCPU is a diskless node, it boots from some server CPU, probably the SCPU. Otherwise, it could boot from its own disk. Booting the SCPU first is necessary since the SCPU cannot be accessed from the **login** prompt, except by using **rlogin**. To use **rlogin**, UNIX must be running. Also note that control automatically goes back to the GCPU after the SCPU is booted. For a system set to autoboot, all of this happens automatically after power-on.

For system administration tasks, it is necessary to load files from the tape. The tape is only accessible from the CPU to which it is attached by its SCSI cable (by default the SCPU). This may make it difficult to use a system with two drives, one for each CPU, since only one of the CPUs can access the tape drive. You must switch to the proper CPU to use the tape drive. The install

scripts work so control doesn't go back to the other CPU until UNIX is shut down. This allows you to do a complete scratch installation on the SCPU from the console.

Once UNIX is running on both CPUs, you log into the GCPU and start X. From here, you can **rlogin** to the SCPU and run X clients that use the GCPU as the display. Because of the DPR interface, X clients running on the SCPU should run faster than those displaying on the GCPU but connected via Ethernet. For more information, refer to the section "Effective System Administration".

The Local Server interface in the PROM code allows you to boot across the LCS network interface instead of Ethernet. This takes advantage of the much faster interface provided by the LCS network interface.

### Server Console Switch

If for any reason the SCPU is required to be booted to multi-user level, the console will switch to the GCPU upon completion of the boot. This may be undesirable under curtain circumstances. The end of the startup file **/etc/rc2** on the SCPU contains a statement that causes console control to be switched from the SCPU to the GCPU. The recommended procedure for disabling the switchover is to search for the following:

```
# This should cause CPUS to give control back to CPUG
if /etc/netstat -i | /usr/bin/fgrep lcs0 > /dev/null ;
then
 echo "Ready" > /dev/kbd
else
 echo "\c"
fi
```

and comment it out:

```
# This should cause CPUS to give control back to CPUG
#if /etc/netstat -i | /usr/bin/fgrep lcs0 > /dev/null ;
#then
# echo "Ready" > /dev/kbd
#else
# echo "\c"
#fi
```

From the boot prompt, the system is booted single-user. At the single user prompt, the **/usr** file system may be checked and mounted on **/usr**. The text editors **vi** and **emacs** can then be used to modify **/etc/rc2**. You should apply the changes to **/etc/rc2** and issue the **telinit** command **telinit 2** to bring the system up to multi-user.

Once the SCPU is up, the console is available to the SCPU, but graphics may not be started up from the SCPU console. When you are ready to boot the

GCPU (be sure to change **/etc/rc2** back to its original condition), you may issue the following command:

```
echo "Ready" > /dev/kbd
```

There is no way to go back to the SCPU console once you have issued this command.

## Shutdown Procedure

Once the GCPU is given control of the console, your only access to the SCPU is via the network. An alternate terminal may be connected to the SCPU and used as a console. The terminal should be connected to the SCPU debug port, which is set up for 9600 baud. The console mode of the PROMs must be set to **r** for the terminal to be used.

Any alternate terminal that is supported by the **termcap/terminfo** facility may be used for the SCPU console. In this case, both CPUs may be booted and shut down independently.

If your SCPU does not have an alternate terminal console (one is not provided with the ESV Workstation), there are several console dependencies that have to be honored.

- For a diskless GCPU, or situations where the GCPU has file systems mounted from the SCPU, it is important not to shut down the SCPU before the GCPU. The GCPU is dependent on the SCPU. If the SCPU is shut down before the GCPU, the GCPU hangs and you have to reset both CPUs in order to reboot.
- You are strongly advised not to mount GCPU disks on the SCPU. Console dependencies during shutdown and reboot may cause conflicts.
- The GCPU can be shutdown and rebooted independently of the SCPU. Since the SCPU depends on the GCPU console, the SCPU may not be shut down unless the GCPU is also shut down. This restriction does not apply to systems where the SCPU is attached to an alternate terminal console.

**Note:** You should not use the shell command **init 6** on the SCPU.

To shut down the SCPU of a Local Server, you first execute a delayed shutdown on the SCPU, log out of the SCPU, and immediately shut down the GCPU. Once the GCPU is down, you must direct the console to attach to the SCPU and wait for the SCPU prompt. The SCPU prompt is displayed when the SCPU completes its shutdown.

Following is an example of the shutdown sequence for an LS system:

Login to SCPU.

```
SCPU# cd /
SCPU# csh
SCPU# shutdown -y -i0 -g300 &
SCPU# exit
SCPU# exit

GCPU# shutdown -y -i0 -g0
```

Several GCPU shutdown messages are printed here.

`GCPU>> scpu` Switch to wait for SCPU to shut down.

```
...
SCPU>>
```

## Effective System Administration

The Local Server gives you a second fully-functional CPU within your ESV Workstation, and the LCS network improves the communication speed between the CPUs. Since there are two CPUs, system administration tasks are required for each CPU.

The LCS network is faster than Ethernet, and it should be used whenever possible. However, there may be some applications that will not make use of the LCS network interface, and in these cases the programs need to be modified or you have to use Ethernet.

Consider the following example:

- **frack** – GCPU Ethernet name
- **gcpu** – GCPU LCS network name
- **frick** – SCPU Ethernet name
- **scpu** – SCPU LCS network name

If you are logged into the GCPU (**frack**), there are two paths to the SCPU (**frick**). To login to the SCPU, you can either:

`frack> rlogin scpu` (through the LCS network)

or

`frack> rlogin frick` (through Ethernet)

If you are on another machine and the Local Server is configured with the defaults, the only usable system names are **frick** and **frack** because the LCS network is not being advertised on Ethernet.

If you have added a new disk to the SCPU, and you want the whole network to be able to mount the new disk over NFS, but you want the GCPU to mount the new disk through the LCS network, the GCPU **mount** command would be:

```
frack> mount scpu:/newdisk /newdisk
```

The **mount** command for other systems on the network would be:

```
hostname> mount frick:/newdisk /newdisk
```

If you want to start an X application on the SCPU and have it display on the GCPU, you can either set your ***DISPLAY*** environment variable to **gcpu** or use the **-display** option with your X application, as follows:

```
frick> setenv DISPLAY gcpu:0.0
frick> /usr/bin/X11/xterm
```

or

```
frick> /usr/bin/X11/xterm -display gcpu:0.0
```

To go over Ethernet, the **-display** option would be:

```
frick> /usr/bin/X11/xterm -display frack:0.0
```

### Network Interface Addresses and Names

The Local Server introduces a new network interface to the system. The SCPU and GCPU are configured to assign a unique network address to their LCS network interfaces. You should use the addresses and interface names (**gcpu** and **scpu**) provided in **/etc/hosts**. The only exception to this is if you want to use the Local Server to act as a gateway between two networks. In this case, addresses and interface names are assigned as they are for any other point-to-point network gateway.

By default, the system is configured so that the network router does not broadcast its routes onto the network. (There is no need to configure the router otherwise unless it is acting as a network gateway). Therefore, the LCS network is not visible to the other hosts on your network, but access is available via Ethernet.

**Note:** If your Local Server system is not being used as a network gateway, you should not allow the router process to broadcast routes unless you first assign a site-specific network address to your Local Server network interfaces.

The system comes configured so that you need not worry about this configuration issue, and you should leave it as it is. However, be aware that your network routers may become confused if duplicate network addresses are presented to them.

### ES/PSX

Users with the ES/PSX option should note that ES/PSX must be installed on the GCPU.