USER’S GUIDE

RASTERFLEX™ Raster Accelerators

Release 4.5

CONNECTWARE

USER'S GUIDE

RasterFlex™ Raster Accelerators

FCC INFORMATION

This device complies with Part 15 of the FCC Rules. Operation is subject to the following two conditions: (1) this device may not cause harmful interference, and (2) this device must accept any interference received, including interference that may cause undesired operation.

RADIO FREQUENCY INTERFERENCE STATEMENT

NOTE: This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15, Subpart B of the FCC Rules. This equipment generates, uses, and can radiate radio frequency energy. If not installed and used in accordance with the instructions, it may cause interference to radio communications.

The limits are designed to provide reasonable protection against such interference in a commercial environment. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause interference to radio or television reception, which can be determined by turning the equipment on and off, the user is encouraged to try to correct the interference by one of more of the following measures:

- Reorient or relocate the receiving antenna of the affected radio or television
- Increase the separation between the equipment and the affected receiver
- Connect the equipment and affected receiver to power outlets on separate circuits
- Consult the dealer or an experienced radio/TV technician for help

MODIFICATIONS

Changes or modifications not expressly approved by Connectware, Inc. could void the user's authority to operate this equipment.

SHIELDED CABLES

Shielded cables must be used with this equipment to maintain compliance with FCC Regulations.

CE MARK INFORMATION

The RasterFlex products are in conformity with the following standards or other normative documents:

- EN 55022, Class B, Limits and methods of measurement of radio interference characteristics of information technology equipment, 1987.
- EN 50082-1, Electromagnetic compatibility - Generic immunity standard -- Part 1: Residential, commercial, and light industry, January 1992.

following the provisions of The Electromagnetic Compatibility Directive, 89/336/EEC.

This product may cause radio interference in which case the user may be required to take adequate measures.

DISCLAIMER

TRADEMARKS

DOCUMENT NUMBER

16-DA3019-2

RELEASE HISTORY

Release 1.1	December 14, 1991
Release 1.2	February 14, 1992
Release 3.0	November 13, 1992
Release 3.1	March 15, 1993
Release 3.2	June 4, 1993
Release 3.3	July 31, 1993
Release 4.0	February 28, 1994
Release 4.0 Rev B	May 11, 1994
Release 4.0 Rev C	May 22, 1995
Release 4.5	May, 1996

COPYRIGHT AND PERMISSION NOTICES

This section lists the copyright and permission notices from Connectware's Licensors for the use of software and related documentation.

MIT

Since the X Window System is being distributed as part of this software release, Connectware includes the following copyright and permission notices. They apply only to the X Window System.

Sun Microsystems

OpenWindows™ V3 X11/NeWS Server

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF TABLES

1. INTRODUCTION

Congratulations on the purchase of a RASTERFLEX Raster Accelerator, designed and manufactured by Connectware, Inc. (Connectware), of Richardson, Texas.

This document provides the user with information on how to install the RASTERFLEX Raster Accelerator hardware and software into any SBus-capable Sun workstation. The RASTERFLEX software runs under SunOS 4.1.X or under Sun Solaris 2.1 and later. To use the RASTERFLEX OpenWindows software, Sun's OpenWindows 3 environment must be installed on your system prior to RASTERFLEX software installation.

This section describes the organization of this document and lists abbreviations, acronyms, mnemonics, and conventions used in this document.

1.1. WHO TO CALL FOR HELP

If you need help, you can call our Customer Assistance Center at (800) 654-7608 or at (214) 997-4188. Our electronic mail address is "support@connectware.com".

1.2. HOW THIS DOCUMENT IS ORGANIZED

This document is organized into 9 sections and two appendices. Each section is described briefly below:

SECTION 1. INTRODUCTION — provides our Customer Assistance Center number, describes the organization of this document and lists abbreviations, acronyms, mnemonics, and conventions used throughout this document.

SECTION 2. SYSTEM CONFIGURATION ISSUES — describes configuration issues to consider when keeping the current framebuffer (running two monitors) or when installing your RASTERFLEX card in certain SPARCstation models. It includes information on selecting the SBus slot and console device, limitations of specific workstations, changing the `sbus-probe-list`, and verifying the monitor and cable compatibility.

SECTION 3. INSTALLING YOUR RASTERFLEX CARD — provides general instructions about installing your RASTERFLEX card.

SECTION 4. INSTALLING YOUR RASTERFLEX SOFTWARE — provides an overview of the software environment for the RASTERFLEX framebuffers and how to install and configure the software.

SECTION 5. RASTERFLEX X11R5 WINDOWING ENVIRONMENT — describes the features of the X Window System™ server that supports MIT's X11R5 implementation of the X windowing system on your SPARCstation. This section provides an overview of the software components provided within the RASTERFLEX X11R5 Environment as well as specific information on how to use the RASTERFLEX X server (`Xrfx`).

SECTION 6. RASTERFLEX X11/NEWS ENVIRONMENT — describes the features of the RASTERFLEX X11/NeWS server for Solaris 1.X (SunOS 4.1.X) that is fully compatible with OpenWindows Version 3.0, 3.1, and 3.2 from SunSoft and provides additional support for the unique capabilities of the RASTERFLEX hardware. This section provides an overview of the software components provided within the RASTERFLEX OpenWindows Environment as well as specific information on how to use the RASTERFLEX X11/NeWS server (xnews-rfx).

SECTION 7. RASTERFLEX LOADABLE DDX ENVIRONMENT — describes the features of the RASTERFLEX Loadable DDX Module that is fully compatible with OpenWindows Version 3.3 and later from SunSoft and provides additional support for the unique capabilities of the RASTERFLEX hardware. This section provides an overview of the software components provided within the RASTERFLEX Loadable DDX Module Environment as well as specific information on how to use the RASTERFLEX module with the OpenWindows X server (Xsun).

SECTION 8. HARDWARE OVERVIEW — provides an overview of the RASTERFLEX accelerator cards, including a list of features and descriptions of the functionality of each of the RASTERFLEX cards.

SECTION 9. RASTERFLEX ADVANCED FEATURES — contains detailed descriptions of the advanced features of the RASTERFLEX cards and is provided for X programmers who want to develop their own applications. The section includes descriptions on the ability to use visual classes other than the server default, for example, using 24-bit windows when the server is brought up with 8-bit as the default; using overlays; and using shared memory for faster `XPutImage` and `XGetImage`. Each description includes an example.

APPENDIX A. SPECIFICATIONS — includes specifications for the RASTERFLEX-24, RASTERFLEX-32, and RASTERFLEX-HR raster accelerator cards, including physical characteristics and 8-bit and 24-bit performance specifications.

APPENDIX B. VIDEO FORMATS — includes horizontal and vertical timing for the video formats used by the RASTERFLEX cards.

1.3. ABBREVIATIONS, ACRONYMS, AND MNEMONICS

The following abbreviations, acronyms, and mnemonics are used in this document:

Table 1.1. Abbreviations, Acronyms, and Mnemonics

Term	Description
ASIC	Application-Specific Integrated Circuit
BitBlt	Bit Block Transfer
BW2	Sun Black-and-White Framebuffer
CG3	Sun 8-bit Pseudocolor Framebuffer
CG4	Sun 8-bit Pseudocolor Framebuffer
CG6	Sun GX 8-bit Accelerated Framebuffer

Table 1.1. Abbreviations, Acronyms, and Mnemonics (Continued)

Term	Description
DAC	Digital-to-Analog Converter
DDX	Device Dependent X
DVMA	Direct Virtual Memory Access
EEPROM	Electrically Erasable Programmable Read-Only Memory
GC	Graphics Context
LSB	Least Significant Bit
LUT	Look-Up Table
NeWS	Network Extensible Window System
RF-24	24-bit RASTERFLEX card
RF-32	32-bit RASTERFLEX card
RF-HR	High-resolution 32-bit RASTERFLEX card
RGB	Red, Green, and Blue planes
SCSI	Small Computer Systems Interface
TWM	Tab Window Manager
WTT	Window Tag Table
X11R5	X11 Release 5

1.4. CONVENTIONS

The following conventions are used in this document:

Table 1.2. Conventions

Convention	Description
`/usr/bin/X11`	File, directory, and program names are Courier regular
`halt`	Command names, instructions and operands are Courier bold
Title	Titles are Helvetica oblique
SECTION	Section names are Helvetica bold

Table 1.2. Conventions

Convention	Description
`dir>`	System prompts include a directory name and right angle bracket
`type`	Arguments and options are Courier italic
`    program listing`	Listings of programs are indented Courier regular text.

2. SYSTEM CONFIGURATION ISSUES

This section describes configuration issues to consider when keeping the current framebuffer (running two monitors) or when installing your RASTERFLEX card in certain SPARCstation models. It includes information on selecting the SBus slot and console device, limitations of specific workstations, changing the `sbus-probe-list`, and verifying the monitor and cable compatibility.

2.1. SELECTING THE SBUS SLOT

For most applications, installing the RASTERFLEX hardware and software is straightforward. If you are replacing a Sun framebuffer with a RASTERFLEX raster accelerator, simply remove the Sun framebuffer and install the RASTERFLEX hardware and software, following the instructions in this document.

If you are keeping the Sun framebuffer card in order to configure a two-monitor system, or installing your RASTERFLEX card in certain SPARCstation models, you may have to move this card or modify the values stored in EEPROM.

2.2. SELECTING THE CONSOLE DEVICE

The system selects the console display device at power-up when the boot software "probes" for the first SBus slot device it finds that is capable of this function. The order in which the slots are probed is determined by a parameter called `sbus-probe-list`, which is stored in the SPARCstation EEPROM.

RASTERFLEX cards are fully capable of acting as the console display device. However, you may prefer another framebuffer as the console display device. You can modify the `sbus-probe-list` to change the order in which SBus slots are probed. For details, see the paragraph on *CHANGING THE SBUS PROBE LIST* later in this section.

2.3. LIMITATIONS OF SPECIFIC WORKSTATIONS

The RASTERFLEX card(s) fits into any one of several Sun SPARCstation systems. Some of these systems have limitations that must be considered before beginning to install the RASTERFLEX card. These limitations are described in the following paragraphs.

2.3.1. SPARCSTATION 1/1+ SLOT LIMITATIONS

On SPARCstation 1 and SPARCstation 1+ machines, the RASTERFLEX cannot be installed in SBus slot 3. Your system can include other SBus add-in cards, particularly SCSI disk controllers, that require DVMA, and therefore also cannot be installed in SBus slot 3, limiting the configuration options to one DVMA-capable card.

Two-slot cards present other limitations. For example, assume that you want to have both a single-slot RASTERFLEX-32 and a double-slot Sun GX card — that you want to act as the console device — installed in a SPARCstation 1. Since the RASTERFLEX cannot be installed in slot 3, the only possibility is to install the GX card in slots 2 and 3, and the

RASTERFLEX-32 in slot 1. The default value for `sbus-probe-list` would choose the RASTERFLEX — in slot 1 — as the console device. To designate the Sun GX card as the console device, set `sbus-probe-list` to "*`0231`*". For details, see the paragraph on *CHANGING THE SBUS PROBE LIST* later in this section.

2.3.2. SPARCSTATION IPC FRAMEBUFFER

On SPARCstation IPC machines, the motherboard includes a black-and-white framebuffer that the system designates as logical SBus slot 3, even though there are only two physical SBus slots. In order to select this framebuffer as the console display device, you must change the `sbus-probe-list` to list slot 3 ahead of the RASTERFLEX slot.

For example, if the RASTERFLEX is installed in slot 1, you can designate the black-and-white framebuffer to act as the console display device by setting the `sbus-probe-list` to "*`0312`*". For details, see the paragraph on *CHANGING THE SBUS PROBE LIST* later in this section.

2.3.3. SPARCSTATION IPX GRAPHICS ACCELERATOR

On SPARCstation IPX machines, the motherboard includes a GX graphics accelerator. As with the IPC, the IPX designates the graphics accelerator as logical SBus slot 3, even though there are only two physical SBus slots. In order to select the GX as the console display device, you must change the `sbus-probe-list` to list slot 3 ahead of the RASTERFLEX slot.

For example, if the RASTERFLEX is installed in slot 1, you can designate the GX to act as the console display device by setting the `sbus-probe-list` to "*`0312`*". For details, see the paragraph on *CHANGING THE SBUS PROBE LIST* later in this section.

2.3.4. SPARCSTATION LX GRAPHICS ACCELERATOR

On SPARCstation LX machines, the motherboard includes a GX graphics accelerator. As with the IPC, the LX designates the graphics accelerator as logical SBus slot 3, even though there are only two physical SBus slots. In order to select the GX as the console display device, you must change the `sbus-probe-list` to list slot 3 ahead of the RASTERFLEX slot.

For example, if the RASTERFLEX is installed in slot 0, you can designate the GX to act as the console display device by setting the `sbus-probe-list` to "*`43012`*". For details, see the paragraph on *CHANGING THE SBUS PROBE LIST* later in this section.

2.3.5. SPARC CLASSIC

On SPARC Classic machines, the motherboard includes an 8-bit framebuffer that the system designates as logical SBus slot 3, even though there are only two physical SBus slots. In order to select this framebuffer as the console display device, you must change the `sbus-probe-list` to list slot 3 ahead of the RASTERFLEX slot.

For example, if the RASTERFLEX is installed in slot 0, you can designate the 8-bit framebuffer to act as the console display device by setting the `sbus-probe-list` to "*`43012`*". For details, see the paragraph on *CHANGING THE SBUS PROBE LIST* later in this section.

2.3.6. SPARCSTATION 10 AND 20 SBUS SLOTS

On SPARCstation 10 and SPARCstation 20 machines, four SBus slots are provided, with two each on two different levels. In order to install a RASTERFLEX card into a SPARCstation 10 or 20, the SBus handle and backplate adapter must first be removed. The holes where the SBus handle is normally installed are used by retaining hardware which is a part of the SPARCstation. Refer to the instructions in the section entitled **INSTALLING THE RASTERFLEX CARD** for instructions on removing the SBus handle and backplate adapter.

2.3.7. SPARCSYSTEM/SPARCSERVER 600MP SBUS SLOTS

On SPARCsystem 600MP and SPARCserver 600MP machines, the CPU card must be removed in order to install the RASTERFLEX card. In order to install the RASTERFLEX card onto the CPU card, the SBus handle and backplate adapter must first be removed. The holes where the SBus handle is normally installed are used by retaining hardware which is provided with the SPARCsystem. Refer to the instructions in the section entitled **INSTALLING THE RASTERFLEX CARD** for instructions on removing the SBus handle and backplate adapter.

2.4. CHANGING THE SBUS PROBE LIST

To change the value of **`sbus-probe-list`**, you must have root privileges (you must be the super-user). Nonvolatile RAM is accessed using the `eeprom` command. To determine the current value of **sbus-probe-list**, type:

```
system% /usr/etc/eeprom sbus-probe-list [SunOS 4.1.X]
system% /usr/sbin/eeprom sbus-probe-list [Solaris 2.X]
```

The system responds with the current value of the **`sbus-probe-list`**. If the system is in its default state, the response is:

```
sbus-probe-list=0123
sbus-probe-list=f0123 [SPARCstation 10, 600MP]
sbus-probe-list=40123 [SPARC Classic, SPARCstation LX]
```

For machines that include "4" or "f" as the first character of the sbus-probe-list string, be sure to include the same first character in your replacement string. If you do not, the machine will not boot. Should this happen inadvertently, you can restore the default values by cycling power while holding down the "L1" and "N" keys until the machine comes up.

To change the **sbus-probe-list** parameter value to "*0312*", type:

```
system% /bin/su
Password: your_superuser_password
system% /usr/etc/eeprom sbus-probe-list=0312
```

Verify that the value was changed properly by using the `eeprom` command again:

```
system% /usr/etc/eeprom sbus-probe-list
sbus-probe-list=0312
```

Once you have changed the value of **`sbus-probe-list`**, you will need to cycle power on your system in order for the changes to take effect.

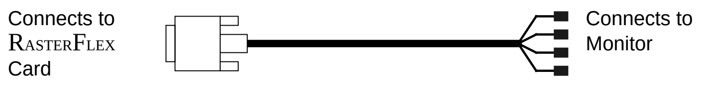
2.5. VERIFYING THE MONITOR AND CABLE COMPATIBILITY

RASTERFLEX cards use a 13W3 coaxial D-shell connector for video output. This connector is compatible with all Sun SPARCstation cabling. If you are replacing an existing Sun color framebuffer with a RASTERFLEX raster accelerator, then simply plug the Sun video cable into the RASTERFLEX card.

If you are installing a RASTERFLEX card into a SPARCstation that did not have a color framebuffer previously, or if you are adding the RASTERFLEX as a second framebuffer along with a second monitor, then you need to determine that the monitor and cabling both are compatible with the new RASTERFLEX card. Connectware offers a color monitor and cable for this purpose. If you would like to use another monitor, ensure that it is compatible with the video format generated by the RASTERFLEX card. This format is described in Appendix B.

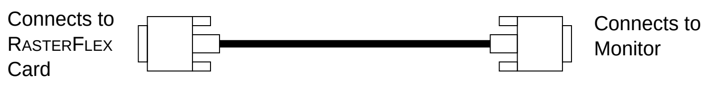
To connect your monitor to the RASTERFLEX, you need one of two types of video cables, depending on the connector type provided on your monitor. If the monitor has separate BNC connectors for Red, Green, Blue, and Sync, you need a 13W3-to-4BNC video cable.

Figure 2.1. 13W3-to-4BNC cable for BNC connector monitor.



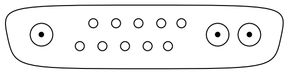
If the monitor has a 13W3 coaxial D-shell connector, you will need a 13W3-to-13W3 video cable. This is the same cable that Sun supplies with their color systems.

Figure 2.2. 13W3-to-13W3 cable for D-shell connector monitor.



The end view of the 13W3 connector on the RASTERFLEX card is illustrated in the following figure:

Figure 2.3. 13W3 connection on the RASTERFLEX card.



2.6. SELECTING MONITOR RESOLUTION

The RASTERFLEX cards are capable of automatically configuring the video resolution to adapt to the monitor they are connected to. All Sun monitors provide sense information which enables the frame buffer to automatically generate the appropriate video resolution. In most cases, this will work transparently, and no action is required by the user.

However, in special cases, you may wish to override the automatic selection mechanism. For instance, if you are using a non-Sun monitor (which does not provide the monitor sense lines) the RASTERFLEX will default to 1152x900 66 Hz resolution. If your monitor requires one of the other supported resolutions, you can select the desired resolution by means of jumpers on the card.

Figure 2.4. Resolution Selection Jumpers - RasterFlex-24 and -32

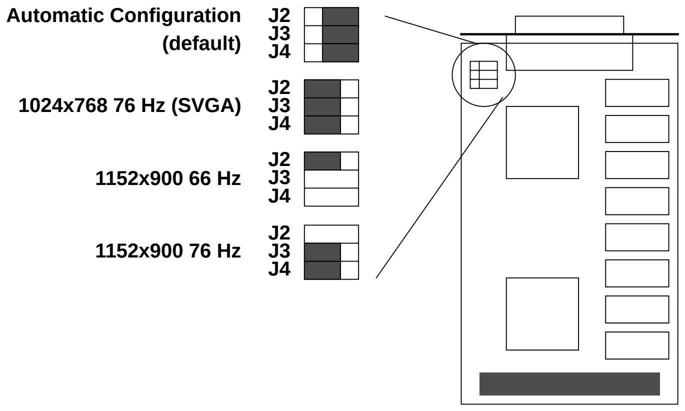
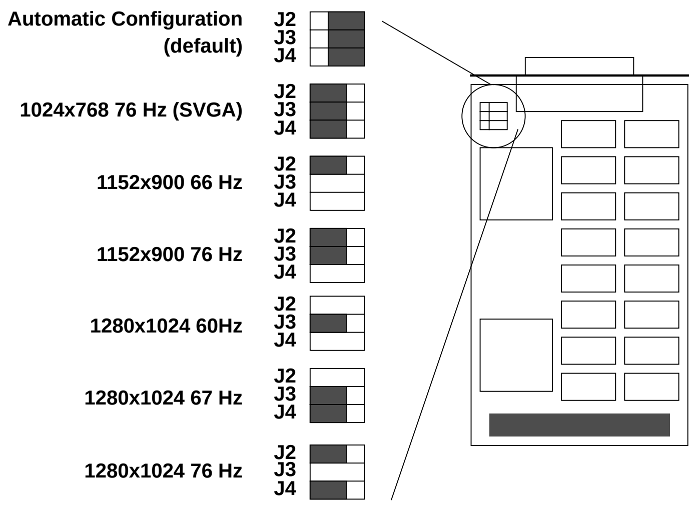


Figure 2.5. Resolution Selection Jumpers - RasterFlex-HR



To select a particular video resolution, install the jumpers at locations J2 through J4 as shown in the applicable figure above. Be sure to retain any spare jumpers in case you need to change the selection in the future. The new jumper settings will take effect the next time power is applied.

3. INSTALLING YOUR RASTERFLEX HARDWARE

This section provides general information to review before installing your RASTERFLEX card, including protecting your card from static, tools you will need, selecting the slot, checking the monitor cables, and general instructions on installing the card into any SBus-capable SPARCstation.

> **NOTE**
> The figures in this section show examples of particular SPARCstation types. Refer to the Installation Guide provided with your workstation for specific instructions for SBus card installation.

3.1. BEFORE YOU START

This section reviews important information you will need before beginning the installation. Please read this section before proceeding.

3.1.1. PROTECTION FROM STATIC

The RASTERFLEX card(s) is shipped in antistatic wrap. In addition, an antistatic wrist strap is provided for your use. Please use the wrist strap when handling the card to avoid electrostatic damage to the product.

3.1.2. TOOLS YOU WILL NEED

Refer to the Installation Guide provided with your workstation for a list of required tools. For most workstation types, you will need a medium and/or small Phillips screwdriver.

3.1.3. SELECTING THE SLOT

Please read the section on **SYSTEM CONFIGURATION ISSUES** before installing the hardware, if you are configuring either of the following systems:

- Installing the RASTERFLEX in a SPARCstation which does not currently have a color framebuffer installed or
- Adding the RASTERFLEX as a second framebuffer.

3.1.4. CHECKING THE MONITOR CABLES

If you are configuring either of the following monitor setups, be certain that you have the appropriate monitor cabling:

- Set up a dual-monitor configuration or
- Interface your RASTERFLEX card with a monitor other than the standard monitor that is provided with your SPARCstation.

Please refer to the section on **SYSTEM CONFIGURATION ISSUES** for more information on monitor cables.

3.2. SHUTTING DOWN THE WORKSTATION

To shut down the machine, you must have root privileges (you must be the superuser).

1. Log on as superuser.

   ```
   system% /bin/su
   Password:your_root_password
   ```

2. Halt the system by entering the UNIX **`halt`** command.

   ```
   system% /etc/halt
   ```

3. The system responds with information, such as the following:

   ```
   syncing file systems... done
   Halted

   Program terminated
   Type b (boot), c (continue), or n (new command mode)
   >
   ```

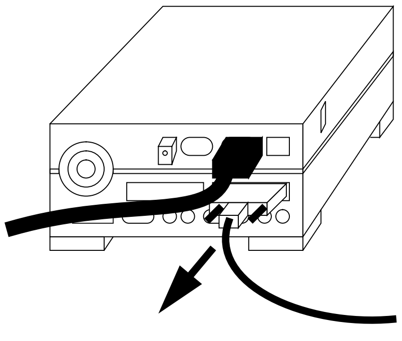
4. Wait for the system boot prompt (">" or "`ok`") to appear.
5. Turn off power to the system unit. Leave the power cord plugged into the wall outlet and workstation in order to provide electrostatic grounding. The AC power in the system unit is limited to the inside of the power supply.

3.3. DISCONNECTING THE MONITOR CABLES

If you are going to use the monitor that is currently set up with your system, and replace another framebuffer card with the RASTERFLEX card, then perform the following steps:

1. Turn off the monitor power switch.
2. Gain access to the rear of the unit.
3. Disconnect the video cable connection from the Sun monitor to the back of the workstation unit.

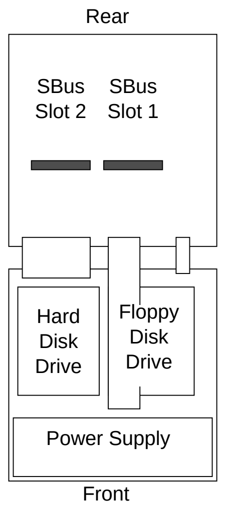
Figure 3.1. Disconnect the monitor video cable from the framebuffer.



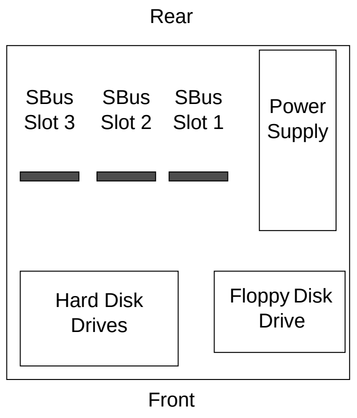
3.4. REMOVING THE WORKSTATION COVER

1. Remove the hardware retaining the SPARCstation cover. Refer to the Installation Guide provided with your workstation for specific instructions.
2. Locate the SBus slots in your workstation, as illustrated in the following footprints of example workstations. Refer to the Installation Guide provided with your workstation for specific information on slot numbering.

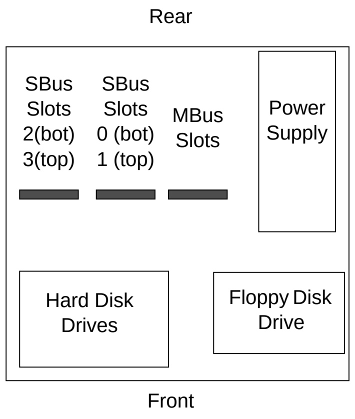
Figure 3.2. Top view of SPARCstations with cover removed.



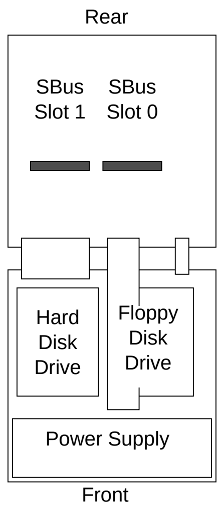
SPARCstation IPC



SPARCstation1/ 2/5



SPARCstation 10/20



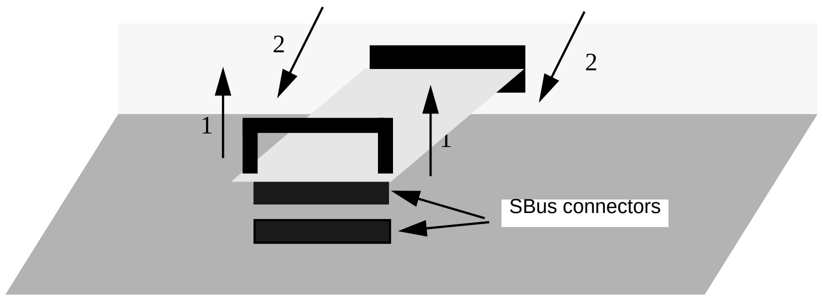
SPARCstation LX/SPARC Classic

3.5. REMOVING THE EXISTING FRAMEBUFFER CARD

If you are replacing the framebuffer card in your system with the RasterFlex card, then perform the following steps:

1. Remove any retaining hardware.
2. Remove the framebuffer card by grasping the handle and pulling straight up. If the card is two slots wide, pull up on both handles at the same time to avoid flexing the card. If the existing card does not have a handle, grasp it above the connector at both ends and pull straight up.

Figure 3.3. Remove existing framebuffer card.

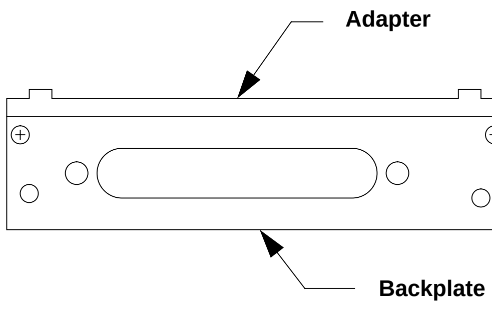


3.6. INSTALLING THE RASTERFLEX CARD

The RasterFlex card can be installed in any SBus slot except slot 3 of a SPARCstation 1/1+ machine. Refer to Section 2.3 for details on this restriction.

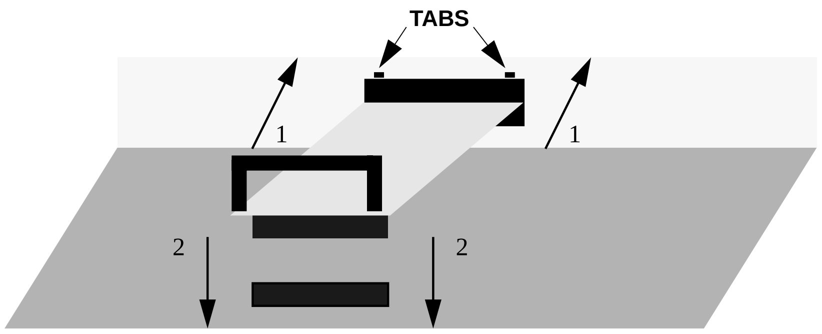
1. Locate the SBus slots that are available for the RasterFlex card and decide into which slot you will install your new RasterFlex card.
2. If you are installing the RasterFlex card in a previously unused slot, remove the dummy backplate and any retaining hardware.
3. If your SPARCstation does not utilize the backplate adapter, remove the adapter from your RasterFlex card using a small Phillips screwdriver.

Figure 3.4. Remove the Backplate Adapter, if necessary.



4. If your workstation does not allow space for SBus handles, remove the handle(s) from your RASTERFLEX card.
5. Install the backplate end of the RASTERFLEX card first, lining up the tabs in the slots of the chassis (1), as illustrated in the following figure. If you removed the backplate adapter in the previous step, simply insert the backplate into the chassis.

Figure 3.5. Install backplate end of the card first.



6. Rotate the card down. Use extra care when plugging in the SBus connectors (2), since the connector pins can be bent. Plug in the connectors.
7. Make sure the connector is firmly seated.
8. Install the dummy backplates into any unused slots to ensure proper airflow and ElectroMagnetic Interference (EMI) protection. If you are replacing a two-slot SBus card with the RASTERFLEX, you may need to obtain an additional dummy backplate for this purpose.
9. Install any needed retaining hardware.

3.7. REPLACING THE WORKSTATION COVER

Install the cover according to the instructions in your Installation Guide.

3.8. CONNECTING THE MONITOR

By this time, you have one of two possible configurations:

- Either you have removed an existing Sun color framebuffer from your system altogether, **replacing** it with the RASTERFLEX card, or
- You are **adding** the RASTERFLEX card to your workstation configuration.

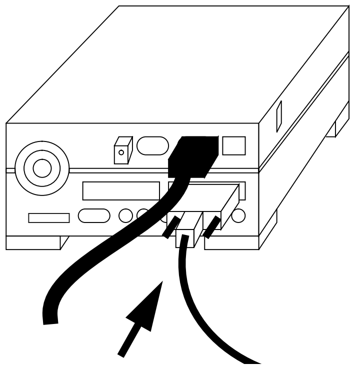
The following procedures describe how to set up your monitor for each of these possibilities.

3.8.1. WHEN REPLACING YOUR SUN FRAMEBUFFER

If you are replacing an existing Sun color framebuffer, the Sun video cable can be plugged directly into the RASTERFLEX card.

1. Connect the Sun video cable to the RASTERFLEX card connector.
2. Tighten the jackscrews finger-tight.

Figure 3.6. Connect the video cable to the RASTERFLEX.



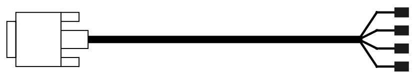
3.8.2. WHEN KEEPING THE SUN FRAMEBUFFER

If you are keeping the Sun framebuffer and adding the RASTERFLEX card with an additional monitor, be certain that the monitor and cable are compatible with the RASTERFLEX card. Refer to Appendix B for information on RASTERFLEX video formats. The RASTERFLEX video connector is a 13W3 coaxial D-shell connector compatible with Sun framebuffers and cabling.

3.8.2.1. MONITOR WITH BNC CONNECTORS

If the monitor has four BNC connectors, you need a 13W3-to-4BNC video cable, illustrated in the following figure. Connectware supplies this type of cable with the optional color monitor.

Figure 3.7. 13W3-to-4BNC cable for BNC connector monitor.

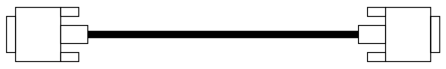


1. Connect the 13W3 connector to the RASTERFLEX and tighten the jackscrews finger-tight.
2. Connect the four BNCs to the Red, Green, Blue and Sync connectors on the monitor. The BNC connectors are color-coded (Red, Green, Blue) and a fourth for Sync.

3.8.2.2. MONITOR WITH 13W3 COAXIAL D-SHELL CONNECTOR

If the monitor has a 13W3 coaxial D-shell connector, you need a 13W3-to-13W3 video cable, as illustrated in the following figure. This is the same cable that Sun supplies with its color systems.

Figure 3.8. 13W3-to-13W3 cable for D-shell connector monitor.



1. Connect either end of the video cable to the RASTERFLEX card, and the other end to the monitor.
2. Tighten the jackscrews finger-tight at both ends.

3.9. TURN ON POWER

You may now turn on power to your SPARCstation.

3.10. PERFORM A RECONFIGURATION BOOT

For Solaris 2, it is necessary to perform a reconfiguration boot the first time you boot your system after installing a new hardware device. To specify that a reconfiguration boot should be performed, simply add the “-r” flag to the normal boot command.

For example, from the system boot prompt type:

```
> b -r
```

or, from the Forth monitor type:

```
ok% boot -r
```

4. INSTALLING YOUR RASTERFLEX SOFTWARE

This section provides a brief overview of the RASTERFLEX software environment, prerequisites for installing the software, and instructions on performing the software installation and configuration. For advanced users, additional details are provided in the sections describing the X11R5, X11/NeWS, and Loadable DDX windowing environments.

4.1. WHAT YOU WILL NEED

In order to install the RasterFLEX software release, you will need the following:

- Access to a standard Sun CD-ROM drive. This drive can be connected directly to your workstation, or to a remote machine on the network.
- The name of the machine that the drive is on (if it is on a remote machine) and the name of the drive (typically `sr0`).
- You will also need superuser privileges, that is, you need to be able to log in as "`root`".
- Also check to see that you have adequate free disk space before attempting the installation. Refer to the section on RELEASE SPACE DISK REQUIREMENTS in the release notes supplied with your software for information on the size of each software component.

4.2. SOFTWARE OVERVIEW

The RASTERFLEX software environment contains a device driver and three different windowing systems: **X11 Release 5, X11/NeWS,** and **Loadable DDX**.

- The device driver provides system-level support for accessing and controlling the RASTERFLEX framebuffers.
- The X11 Window System environment includes an X Window System™ server which uses the unique features of the RASTERFLEX framebuffers along with the set of standard clients, demos, and libraries which are distributed as part of X11 Release 5 from the X Consortium.
- The X11/NeWS environment includes an X11/NeWS server supporting the RASTERFLEX devices for use with OpenWindows 3,0, 3.1, or 3.2.
- The Loadable DDX environment includes a Loadable DDX object module supporting the RASTERFLEX devices for use with OpenWindows 3.3.

A single version of each software component (device driver, window system servers, etc.) supports all RASTERFLEX hardware types: RASTERFLEX-24, RASTERFLEX-32, and RASTERFLEX-HR. Separate versions are required for support of SunOS 4.1.X and Solaris 2.X.

The following table indicates which software items are supported under the various SunOS releases:

Table 4.1. RASTERFLEX Operating System Support

Item	Solaris 1	Solaris 2
Device Driver	All	All
X11R5 Environment	All	All
X11/NeWS Environment	All	None
Loadable DDX Environment	None	SunOS 5.3 SunOS 5.4 SunOS 5.5

4.2.1. SOFTWARE REQUIREMENTS

The RASTERFLEX software environment requires that SunOS 4.1.X or Solaris 2.X or later is installed on the host SPARCstation. In order to run the X11/NeWS Environment for Solaris 1.X (SunOS 4.1.X) on the RASTERFLEX framebuffers, the standard OpenWindows 3.0, 3.1, or 3.2 release from SunSoft must also be installed before the RASTERFLEX X11/NeWS software is installed. In order to run the Loadable DDX Environment, the standard OpenWindows 3.3 (or higher) release must be installed on the system.

4.2.2. THE DEVICE DRIVER

The RASTERFLEX software release contains a device driver which manages the RASTERFLEX device. This device driver assists in maintaining console integrity when the RASTERFLEX device is being used as the system console and also provides low-level support for the RASTERFLEX window system environments.

The RASTERFLEX device driver is loaded automatically at system boot time once it has been installed on the system. If installed correctly, the following message should appear on the system console during the boot sequence under SunOS 4.1.X:

```
    VITec, RasterFLEX0 at SBus slot N 0x0 pri 7
```

For Solaris 2, this message will appears as follows:

```
    VITec, RasterFLEX-320 at sbus0: SBus slot N 0x0 SBus level 5
            sparc ipl 9
```

In addition, device nodes with the appropriate major device number for the loaded driver are created automatically in the `/dev` directory for each RASTERFLEX device which is installed on the system. If installed, these device nodes are named `/dev/rfx0`, `/dev/rfx1`, and so on.

The interfaces to the RASTERFLEX device driver are not public. These devices should not be opened directly by any software other than the RASTERFLEX window system software supplied by Connectware.

4.2.3. THE X11R5 WINDOWING ENVIRONMENT

The RASTERFLEX software release contains a full X11 Release 5 windowing environment including a server, clients, demos, and libraries.

The clients, demos, and libraries provided on the release tape include the standard set of X software, as distributed within X11 Release 5 from the X Consortium. All official X Consortium patches for X11 Release 5 have been applied. This software is supplied “as is” for the convenience of RASTERFLEX users who do not have access to the X release from other sources and is not directly supported by Connectware.

The X Window System server (**`Xrfx`**) provided with the software release is a specially modified version of the X11 Release 5 Sample Server which has been enhanced to support the unique display capabilities of the RASTERFLEX hardware. Additionally, this server supports the standard Sun monochrome and 8-bit color framebuffers which are supported by the original X11 release in combination with the RASTERFLEX hardware in a multi-screen environment. The **`Xrfx`** server is a Connectware-supported software component. Refer to the section on the **RASTERFLEX X11R5 WINDOWING ENVIRONMENT** for more details.

4.2.4. THE X11/NeWS ENVIRONMENT

The RASTERFLEX software release contains an X11/NeWS server (**`xnews-rfx`**) for Solaris 1.X (SunOS 4.1.X) based upon Sun’s OpenWindows 3. This server is based upon the standard `xnews` server distributed by Sun Microsystems as part of OpenWindows 3.0, 3.1, and 3.2, but has been enhanced to support the unique capabilities of the RASTERFLEX hardware. The clients and libraries provided with OpenWindows 3 can be used in conjunction with the RASTERFLEX X11/NeWS server.

The RASTERFLEX X11/NeWS server does not run in SunView compatibility mode, that is, SunView applications cannot be run concurrently with the OpenWindows environment. Refer to the section on the **RASTERFLEX X11/NEWS ENVIRONMENT** for more details.

4.2.5. THE LOADABLE DDX ENVIRONMENT

The RASTERFLEX software release contains a loadable DDX module which is fully compatible with the OpenWindows 3.3 server (or higher). This module is automatically located and loaded by the standard **`Xsun`** server distributed by Sun Microsystems as part of OpenWindows 3.3 (or higher), and supports the unique capabilities of the RASTERFLEX hardware. The clients and libraries provided with OpenWindows 3.3 (or higher) can be used in conjunction with the RASTERFLEX Loadable DDX Environment. Refer to the section on the **RASTERFLEX LOADABLE DDX ENVIRONMENT** for more details.

4.3. INSTALLING THE SOFTWARE

The RASTERFLEX software release is provided on a single CD-ROM which contains all software elements for both Solaris 1 and Solaris 2. To install the software, the CD-ROM image is mounted as an ISO 9660 file system and a special script called rfxinstall is used to initiate installation. This script will determine which software may be loaded based upon the operating system being run and will prompt the user to select which portions of the release should loaded and the directories into which they should be placed.

To install the software, perform the following steps:

4.3.1. LOG IN AS ROOT.

To install the RASTERFLEX software, you must have superuser privileges on both the target machine and the machine containing the CD-ROM device if performing a remote installation. (you must be logged in as "root").

```
login: root
Password:your_root_password
```

4.3.2. INSERT THE RELEASE CD-ROM

Insert the RASTERFLEX CD-ROM (label side up) into the CD-ROM carrier and insert the carrier into the drive. You must know the machine name (if it is a remote machine) and the CD-ROM drive name (such as `sr0`) in order to begin the installation.

4.3.3. MOUNT THE CD-ROM FILESYSTEM

After inserting the RASTERFLEX CD-ROM, you must mount the CD-ROM contents as a read-only ISO 9660 filesystem on the machine containing the CD-ROM drive. The following mount commands will do this under Solaris 1 and 2:

Solaris 1 (SunOS 4.1.X):

```
mount -rt hsfs /dev/sr0 /cdrom
```

Solaris 2.0 and Solaris 2.1 (SunOS 5.0, 5.1):

```
mount -o ro -F hsfs /dev/sr0 /cdrom
```

If you are running Solaris 2.2 or later, the SunOS Volume Management software included in this release will automatically mount the CD upon insertion into the drive. For this version, the RASTERFLEX Software Release will be mounted under the directory `/cdrom/rasterflex_software_release_#_#` where "`#_#`" is the software release number. No further action is necessary to mount the CD-ROM.

4.3.4. EXPORT THE CD-ROM FILESYSTEM (REMOTE)

If you are performing a remote CD-ROM installation, the machine containing the CD-ROM drive must be set up to allow the CD-ROM filesystem to be exported to a remote system. The following sections describe how to do this under Solaris 1 and 2:

4.3.4.1. Solaris 1 (SunOS 4.1.X):

Under Solaris 1, you must ensure that an entry exists in the file `/etc/exports` to allow the CD-ROM file system to be mounted as a read-only filesystem. The entry in the `/etc/exports` file should look something like:

```
/cdrom -ro
```

Whenever you modify the `exports` file, you must also run **`exportfs`** to notify NFS that changes have been made:

```
exportfs -a
```

4.3.4.2. Solaris 2 (SunOS 5.X):

Under Solaris 2, you may use the **`share`** command to indicate that the CD-ROM file system may be exported as a read-only file system across the network:

```
share -o ro /cdrom
```

4.3.5. NFS MOUNT THE CD-ROM FILESYSTEM (REMOTE)

If you are performing a remote installation and you have configured the system containing the CD-ROM drive to export its filesystem, you may then mount the CD-ROM filesystem on the machine upon which the RASTERFLEX software release is to be installed. This can be done using the following **`mount`** command (where "*remote*" is the hostname of the remote system containing the CD-ROM drive).

```
mount -r remote:/cdrom /cdrom
```

4.3.6. EXECUTE THE RFXINSTALL UTILITY

The top-level directory of the RASTERFLEX CD-ROM filesystem contains a shell script named **`rfxinstall`** which will initiate the RASTERFLEX software installation. If you are running a version of Solaris earlier than Solaris 2.2:

```
cd /tmp
/cdrom/rfxinstall
```

If you are running Solaris 2.2 or later:

```
cd /tmp
/cdrom/cdrom0/rfxinstall
```

You will be asked to respond to questions regarding whether or not you wish to install certain portions of the software release, and where you wish to install the software in your file system.

You will probably want to use all the default answers if you have not previously installed a RASTERFLEX software release, and you have enough disk space to contain it. Refer to the next section for detailed information on the various modules which are provided in the RASTERFLEX software release.

If you are constrained for disk space and already have OpenWindows 3 installed on your system, selecting the OpenWindows configuration will provide a fully functional window system environment with the minimal amount of disk space.

Under Solaris 2, you can use the **`pkginfo`** command to list the RASTERFLEX software packages which have been successfully installed on your system. To do this, use the following command:

```
pkginfo | grep VIT
```

4.4. CONFIGURING THE SOFTWARE RELEASE

After completing the software installation, the installation utility will automatically provide you with the option of executing the Software Configuration Utility. Additionally, you may run the configuration utility at any time if you have installed additional software or wish to alter your software configuration. The Software Configuration Utility is a shell script named **`rfxconfig.sh`** which will reside in the directory into which the RASTERFLEX device driver has been installed (by default, `/etc/modules` for Solaris 1, `/opt/VITrflex` for Solaris 2). The utility may be run by any user (it does not require root privileges). To start the configuration utility, simply change to the appropriate directory and execute the script.

Solaris 1 (SunOS 4.1.X):

```
cd /etc/modules
./rfxconfig.sh
```

Solaris 2 (SunOS 5.X):

```
cd /opt/VITrflex
./rfxconfig.sh
```

The Software Configuration Utility will automatically generate the appropriate environmental information required to run the X11 Release 5, X11/NeWS, or Loadable DDX window systems on the RASTERFLEX accelerators. The information generated by the utility consists of an initialization file named `.cshrc.flex` which can be sourced from within an individual user's `.cshrc` file. The `.cshrc.flex` file will contain the appropriate setting of the **`PATH`**, **`LD_LIBRARY_PATH`**, **`MANPATH, SERVER`** and other environment variables based upon the locations into which the RASTERFLEX software release has been installed.

The Software Configuration utility will first scan the system to determine the locations where RASTERFLEX software modules have been loaded onto the system and the directories within which they reside. If the utility detects any problems with your software configuration, it will display an error message along with the corrective action you should take. The utility will interactively prompt you to determine how you would like to have your user environment configured. Once all questions have been answered, the `.cshrc.flex` file will be generated. The new configuration information can be incorporated into any login environment by adding the following line to the end of the `.cshrc` file in each RASTERFLEX user's home directory:

```
    source {directory-name}/.cshrc.flex
```

After modifying the appropriate `.cshrc` file, log off the system and log in again. You should now be able to start up the window system environment on the RASTERFLEX accelerator by simply typing **`xinit`** or **`openwin`**.

4.5. CONTENTS OF THE SOLARIS 1 (SUNOS 4.1.X) RELEASE

This subsection and following subsections are organized around the list of components provided in the Solaris 1 version of the RASTERFLEX Software Release providing a quick way to find information about a particular piece of software.

4.5.1. DEVICE DRIVER

The Device Driver is a **`required`** software component. The installation utility will automatically install the device driver during software installation if the RASTERFLEX hardware has already been installed within your system. These files must be placed in `/etc/modules`.

rfx.o	RASTERFLEX loadable device driver
rfx.sh	Device driver installation script
rfxconfig.sh	RASTERFLEX Software Configuration Utility

4.5.2. OPENWINDOWS 3.0 X11/NEWS SERVER

The RASTERFLEX X11/NeWS Server is a **required** software component if you intend to run the OpenWindows 3.0 X11/NeWS Window System on the RASTERFLEX hardware. The default location for these files is `/usr/openwin.`

./bin/xnews-rfx	RASTERFLEX OpenWindows 3.0 X11/NeWS server
./bin/vset	RASTERFLEX Visual Selection utility

4.5.3. OPENWINDOWS 3.0 MANUAL PAGES

The OpenWindows 3.0 manual pages consists of manual pages which describe additional unique capabilities of the RASTERFLEX OpenWindows software environment. This component is optional. The default location for this file is `/usr/openwin`.

./man/man1/xnews-rfx.1	Manual page for RASTERFLEX X11/NeWS server.
./man/man1/vset.1	Manual page for RASTERFLEX Visual Selection utility

4.5.4. X11R5 SERVER

The RASTERFLEX X Server is a **required** software component if you intend to run the X11 Release 5 Window System on the RASTERFLEX hardware. The default location for these files is `/usr/X11R5`.

./bin/Xrfx	RASTERFLEX X11R5 server

./bin/X	Symbolic link to Xrfx
./bin/constype	Program to display sun console type
./bin/kbd_mode	Set console keyboard mode
./bin/showrgb	Display contents of RGB database
./lib/rgb.dir	DBM file for RGB database
./lib/rgb.pag	DBM file for RGB database
./lib/rgb.txt	Textual version of RGB database

4.5.5. X11R5 FONTS

The Fonts component provides font utilities and font data files used by the RASTERFLEX X server. The RASTERFLEX X Server uses the Portable Compiled Format (PCF) font file format which was included in X11 Release 5. The RASTERFLEX X11R5 Server will also support the Server Natural Format (SNF) font file format used in X11 Release 4 and earlier releases. The RASTERFLEX X11R5 Server does not support the OpenWindows font format. You will be required to load this component unless you have already installed PCF fonts from the MIT X Release. The default location for these files is `/usr/X11R5`.

./bin/bdftopcf	BDF to PCF font format converter
./bin/fs	X Font Server
./bin/fsinfo	X Font Server information utility
./bin/fslsfonts	X Font Server font list displayer
./bin/fstobdf	BDF font generator
./bin/mkfontdir	Generates font directories
./bin/showfont	Font display utility
./lib/fonts/100dpi/*	100 dots-per-inch bitmap font files
./lib/fonts/75dpi/*	75 dots-per-inch bitmap font files
./lib/fonts/misc/*	Miscellaneous bitmap font files
./lib/fonts/Speedo/*	Contributed Speedo scalable font files
./lib/fonts/Type1/*	Contributed Type 1 scalable font files
./lib/fonts/PEX/*	PEX format font files

4.5.6. X11R5 CLIENTS

The Clients module consists of the set of clients and demonstrations which were included in the Core distribution for X11 Release 5. This component is optional. If preferred, you may use the X clients which are included in the standard OpenWindows 3.0 release from SunSoft (if it has been installed on your system). The default location for these files is `/usr/X11R5`.

./bin/*	X11 Release 5 Core clients/demos (xterm, xinit, etc)

./include/X11/bitmaps/*	Bitmap files used by X11R5 clients
./lib/X11/app-defaults/*	Application default files for Core clients/demos
./lib/X11/twm/*	Default twm window manager configuration files
./lib/X11/xman.help	xman help file
./lib/X11/images/rfx.ppm	PPM format image file for RASTERFLEX signature image

4.5.7. X11R5 DEVELOPMENT ENVIRONMENT

The Development Environment module consists of the set of include files and libraries which were included in the Core distribution for X11 Release 5. This component is optional; however, it is required if you have loaded the Clients module. If preferred, you may use the X development libraries which are included in the standard OpenWindows release from Sun (if it has been installed on your system). The default location for these files is `/usr/X11R5`.

./bin/imake	Imake utility
./include/X11/*	Xlib and Xt Intrinsics include files
./include/X11/PEX5/*	PEX include files
./include/X11/Xaw/*	Athena Widgets include files
./include/X11/Xmu/*	Miscellaneous utilities include files
./include/X11/bitmaps/*	X bitmap files
./include/X11/extensions/*	X11R5 extension include files
./include/X11/phigs	PHIGS include files
./lib/libPEX5.*	PEX 5.0 libraries
./lib/libX11.*	Xlib libraries
./lib/libXau.a	X Authorization library
./lib/libXaw.*	Athena Widgets libraries
./lib/libXdmcp.a	X Display Manager library
./lib/libXext.*	X Extensions libraries
./lib/libXi.*	X Input Extension libraries
./lib/libXmu.*	X Miscellaneous Utilities libraries
./lib/libXt.*	X Toolkit Intrinsics libraries
./lib/libXtst.*	X Test Extension libraries
./lib/libXv.a	X Video Extensions library
./lib/liboldX.*	X10 Compatibility libraries
./lib/libphigs.a	Phigs support library
./lib/X11/XErrorDB	X Error Database

4.5.8. X11R5 MANUAL PAGES

The Manual Pages module consists of the set of manual pages which were included in the Core distribution for X11 Release 5. This component is optional. The default location for these files is `/usr/X11R5`.

./man/man3/*	Manual pages for Xlib and Xt Intrinsics functions
./man/mann/*	Manual pages for Xrfx server, clients, and demos

4.6. CONTENTS OF THE SOLARIS 2 (SUNOS 5.X) RELEASE

This subsection and following subsections are organized around the list of components provide in the Solaris 2 version of the RASTERFLEX Software Release providing a quick way to find information about a particular piece of software.

4.6.1. VITrdrvr - DEVICE DRIVER

The Device Driver is a **required** software component under Solaris 2. The Software Installation Utility will automatically install the device driver during software installation if the RASTERFLEX hardware has already been installed within your system. The default location for these files is in `/opt/VITrflex`.

modules/rfx	RASTERFLEX loadable device driver for Solaris 2.[012]
modules/rfx_mapdev	RASTERFLEX loadable device driver for Solaris 2.3
modules/seg_mapdev	Segment driver required for DGA support
rfxconfig.sh	RASTERFLEX configuration utility

After installing the device driver module, this package runs a post-installation script which will automatically select the correct version of the RASTERFLEX device driver and configure it to be automatically loaded on your system.

4.6.2. VITropwin - X11/NEWS ENVIRONMENT

The RASTERFLEX X11/NeWS Server is a **required** software component if you intend to run the OpenWindows 3.1 X11/NeWS Window System on the RASTERFLEX hardware. The default location for these files is `/opt/VITrflex`. This package will only be available when the software installation is being performed on a system running versions of Solaris 2 prior to Solaris 2.3.

./bin/xnews-rfx	RASTERFLEX OpenWindows 3.1 X11/NeWS server
./bin/vset	RASTERFLEX Visual Selection utility
./man/man1/xnews-rfx.1	Manual page for RASTERFLEX X11/NeWS server.
./man/man1/vset.1	Manual page for RASTERFLEX Visual Selection utility

After installing the OpenWindows 3.1 environment, this package contains a post-installation script which will automatically create symbolic links from the SunSoft OpenWindows 3.1 environment (if it is installed) to the installed RASTERFLEX files.

NOTE: No files from the original OpenWindows release will be altered in any manner.

4.6.3. VITrfddx - OW LOADABLE DDX ENVIRONMENT

The RASTERFLEX OpenWindows Loadable DDX Environment is a **required** software component if you intend to run the OpenWindows 3.3 X Window System on the RASTERFLEX hardware. The default location for these files is `/opt/VITrflex.`

./modules/ddxVITrfx.so.1	RASTERFLEX Loadable DDX Object Module
./bin/vset	RASTERFLEX Visual Selection utility
./man/man1/vset.1	Manual page for RASTERFLEX Visual Selection utility

After installing the Loadable DDX environment, this package contains a post-installation script which will automatically create symbolic links from the SunSoft OpenWindows 3.3 environment (if it is installed) to the installed RASTERFLEX files. This package will only be available when the software installation is being performed on a system running Solaris 2.3 or later.

4.6.4. VITrxserv - X11R5 SERVER

The RASTERFLEX X Server is a **required** software component if you intend to run the X11 Release 5 Window System on the RASTERFLEX hardware. The default location for these files is `/opt/VITrflex`.

./bin/Xrfx	RASTERFLEX X11R5 server
./bin/X	Symbolic link to Xrfx
./bin/constype	Program to display sun console type
./bin/kbd_mode	Set console keyboard mode
./bin/showrgb	Display contents of RGB database
./lib/rgb.dir	DBM file for RGB database
./lib/rgb.pag	DBM file for RGB database
./lib/rgb.txt	Textual version of RGB database

4.6.5. VITrxfont - X11R5 FONTS

The Fonts component provides font utilities and font data files used by the RASTERFLEX X server. The RASTERFLEX X Server uses the Portable Compiled Format (PCF) font file format which was included in X11 Release 5. The RASTERFLEX X11R5 Server will also support the Server Natural Format (SNF) font file format used in X11 Release 4 and earlier releases. The RASTERFLEX X11R5 Server does not support the OpenWindows font format. You will be required to load this component unless you have already installed PCF fonts from the MIT X Release. The default location for these files is `/opt/VITrflex`.

./bin/bdftopcf	BDF to PCF font format converter
./bin/fs	X Font Server

./bin/fsinfo	X Font Server information utility
./bin/fslsfonts	X Font Server font list displayer
./bin/fstobdf	BDF font generator
./bin/mkfontdir	Generates font directories
./bin/showfont	Font display utility
./lib/fonts/100dpi/*	100 dots-per-inch bitmap font files
./lib/fonts/75dpi/*	75 dots-per-inch bitmap font files
./lib/fonts/misc/*	Miscellaneous bitmap font files
./lib/fonts/Speedo/*	Contributed Speedo scalable font files
./lib/fonts/Type1/*	Contributed Type 1 scalable font files
./lib/fonts/PEX/*	PEX format font files

4.6.6. VITrxsupt - X11R5 SUPPORT ENVIRONMENT

The support module consists of the set of clients, libraries, and demonstrations which were included in the Core distribution for X11 Release 5. This component is optional. If preferred, you may use the X clients which are included in the standard OpenWindows 3.X release from SunSoft (if it has been installed on your system). The default location for these files is `/opt/VITrflex`.

./bin/*	X11 Release 5 Core clients/demos (xterm, xinit, etc)
./include/X11/bitmaps/*	Bitmap files used by X11R5 clients
./lib/X11/app-defaults/*	Application default files for Core clients/demos
./lib/X11/twm/*	Default twm window manager configuration files
./lib/X11/xman.help	xman help file
./lib/X11/images/rfx.ppm	PPM format image file for RASTERFLEX signature image
./bin/imake	Imake utility
./include/X11/*	Xlib and Xt Intrinsics include files
./include/X11/PEX5/*	PEX include files
./include/X11/Xaw/*	Athena Widgets include files
./include/X11/Xmu/*	Miscellaneous utilities include files
./include/X11/bitmaps/*	X bitmap files
./include/X11/extensions/*	X11R5 extension include files
./include/X11/phigs	PHIGS include files
./lib/libPEX5.*	PEX 5.0 libraries
./lib/libX11.*	Xlib libraries
./lib/libXau.a	X Authorization library

./lib/libXaw.*	Athena Widgets libraries
./lib/libXdmcp.a	X Display Manager library
./lib/libXext.*	X Extensions libraries
./lib/libXi.*	X Input Extension libraries
./lib/libXmu.*	X Miscellaneous Utilities libraries
./lib/libXt.*	X Toolkit Intrinsics libraries
./lib/libXtst.*	X Test Extension libraries
./lib/libXv.a	X Video Extensions library
./lib/liboldX.*	X10 Compatibility libraries
./lib/libphigs.a	Phigs support library
./lib/X11/XErrorDB	X Error Database

4.6.7. VITrxman - X11R5 MANUAL PAGES

The Manual Pages package consists of the set of manual pages which were included in the Core distribution for X11 Release 5. This component is optional. The default location for these files is `/opt/VITrflex`.

./man/man3/*	Manual pages for Xlib and Xt Intrinsics functions
./man/mann/*	Manual pages for Xrfx server, clients, and demos

5. RASTERFLEX X11R5 WINDOWING ENVIRONMENT

The RASTERFLEX X11R5 Windowing Environment consists of an X Window System server (**`Xrfx`**) for the RASTERFLEX cards plus the standard set of clients, libraries, demonstrations, and manual pages which were included in the core distribution of X11 Release 5 from the X Consortium. This section provides an overview of the components provided within the RASTERFLEX X11R5 Windowing Environment as well as specific information on how to use the RASTERFLEX X11R5 server.

5.1. X11R5 SOFTWARE COMPONENTS

This section describes the various software components which are provided with the X11R5 Window System Environment for the RASTERFLEX framebuffers. Under SunOS 4.1.X, the default location for X11R5 software installation is `/usr/X11R5`. Under Solaris 2.X, the default location is `/opt/VITrflex`. Throughout this section, this default location will be referred to using an environment variable, `$X_INSTALL`. You may either set this environment variable to the location into which the X11R5 software has been installed and enter the commands as shown, or simply substitute the actual installation location for each instance of `$X_INSTALL` in this document.

5.1.1. SOFTWARE RELEASE BUTLER

The Software Release Butler is a utility which prompts you with questions regarding the installation, then performs the installation for you. The questions it asks involve such things as which portions of the software you wish to install and where you wish to install them in your file system. The Software Release Butler is invoked automatically when you use the **`rfxinstall`** utility provided on the RASTERFLEX CD-ROM.

5.1.2. SOFTWARE CONFIGURATION MECHANIC

The Software Configuration Mechanic is a utility which will examine the manner in which you have installed the RASTERFLEX software release on your system, ask you a few basic configuration questions, and then generate a file containing a set of environment variable assignments suitable for inclusion in a user `.cshrc` file. The Software Configuration Mechanic is a shell script named `rfxconfig.sh` which by default is stored in `/etc/modules` under SunOS 4.1.X, and in `/opt/VITrflex` under Solaris 2.X. Refer to the Software Release notes for more specific information on running the Software Configuration Mechanic.

5.1.3. DEVICE DRIVER

The device driver allows the operating system to communicate with the RASTERFLEX hardware. It is **required** in order to run the remainder of the RASTERFLEX software. Refer to the Software Release notes for more specific information on how to install the device driver.

5.1.4. SERVER & RGB DATABASE

The X11R5 server is **required** to run X on the RASTERFLEX card. Connectware has ported the original X11R5 Sun Sample Server to the RASTERFLEX hardware, making optimizations to support the special features of the RASTERFLEX device. If you already have an X11R5 release on your workstation, you may wish to install only this piece of the X release. The server binary file is loaded into `$X_INSTALL/bin` and the RGB database is loaded into `$X_INSTALL/lib`.

5.1.5. X FONTS

This is a library of fonts used by the X server. Installation of fonts is **required** for use of the Connectware X11R5 server unless you already have X11R5 installed on your workstation. You cannot use the fonts supplied with the OpenWindows release directly with the X11R5 server. The font files are written to `$X_INSTALL/lib/X11/fonts` and various font utilities are stored in `$X_INSTALL/bin/`.

5.1.6. X CLIENTS

This is a collection of executable X clients and demonstration programs, such as the X terminal emulator (**`xterm`**), clock (**`xclock`**), calculator (**`xcalc`**) and Tab Window Manager (**`twm`**). These are useful items which you probably want to install unless you already have an X11R5 release on your system, or you may prefer to use the OpenWindows clients. The client programs are stored in `$X_INSTALL/bin` and any required support files, such as application defaults, are placed in `$X_INSTALL/lib/X11`.

5.1.7. X DEVELOPMENT ENVIRONMENT

This is a collection of libraries, header files, etc., used by programmers to create and execute X application programs. Installation of these files is **required** if you plan to use the client and/or demo programs included in this release. The RASTERFLEX programming examples in this document are also among the files in this library. The library files are placed in `$X_INSTALL/lib` and include files reside in `$X_INSTALL/include`.

5.1.8. X MANUAL PAGES

These are the standard manual pages for X11R5, as well as a manual page specific to the RASTERFLEX X11R5 server. You probably want to install the manual pages even if you already have X11R5 installed in order to have access to the RASTERFLEX manual pages. The X manual pages are installed in `$X_INSTALL/man/man3` for X libraries and `$X_INSTALL/man/mann` for the X server and clients.

5.2. SETTING UP THE USER ENVIRONMENT

This section outlines the changes which may be required to the user environment after loading the RASTERFLEX software. It may be desirable to add the following commands to the system `.cshrc` file or other user start-up files so that these changes will apply to all users.

IMPORTANT!
Please read this section carefully, particularly if you have elected to load any portions of the RASTERFLEX software release in locations other than the ones recommended by the Release Butler.

Note that the RASTERFLEX Software Configuration Mechanic, provided as a part of the RASTERFLEX software release, performs basic environment setup automatically. This section is provided for advanced users who wish to customize their environment. The Configuration Mechanic is (by default) installed in `/etc/modules/rfxconfig.sh` under SunOS 4.1.X, and under `/opt/VITrflex` under Solaris 2.X.

5.2.1. LOCATING SERVER RESOURCES

The RASTERFLEX X11R5 server must be able to locate certain resources in order to execute properly. If the Server and Fonts components of the RASTERFLEX software release have been loaded into their standard locations, this happens automatically. If you have selected a location other than the default into which these files should be loaded, then you need to explicitly tell the server where they are located via command line options. The critical resources are the server font directories and the server RGB database.

The standard location for the font directories is in `$X_INSTALL/lib/X11/fonts/misc,$X_INSTALL/lib/X11/fonts/75dpi,$X_INSTALL/lib/X11/fonts/100dpi`. If the font directories are loaded at a different location on your system, you always need to use the font path (`-fp`) option when executing the RASTERFLEX server. For example, if you elected to load the RASTERFLEX software release under the single directory `/usr/rfx`, then the following command line option would be required:

```
... -fp /usr/rfx/lib/X11/fonts/misc,/usr/rfx/lib/X11/fonts/
75dpi,/usr/rfx/lib/X11/fonts/100dpi ...
```

The single parameter to `-fp` option is a comma-separated list of font directory names (with no spaces between directory names).

The standard location for the server RGB database is `$X_INSTALL/lib/X11/rgb`. This actually means that there should be two files in the `$X_INSTALL/lib/X11` directory: `rgb.pag` and `rgb.dir`. If these files are not at that location, you need to specify their location via the color database (`-co`) option when executing the RASTERFLEX server. For example, if you elected to load the RASTERFLEX software release under the single directory `/usr/rfx`, then the following command line option would be required:

```
... -co /usr/rfx/lib/X11/rgb ...
```

The single parameter to the `-co` option is the directory which contains the `rgb.dir` and `rgb.pag` files with the string `rgb` appended to it.

5.2.2. LOCATING SHARED LIBRARIES

It is necessary to set up a user's shared library link path to locate the shared libraries included in this release. This must be done to run any of the client and demo programs included in the release. To do this, the **`LD_LIBRARY_PATH`** environment variable should be set to point to the location where the libraries have been loaded. For example, if the RASTERFLEX software release was loaded under the single directory `/usr/X11R5`, the following command would be used:

```
setenv LD_LIBRARY_PATH /usr/X11R5/lib:$LD_LIBRARY_PATH
```

5.2.3. SETTING THE PATH VARIABLE

The location of the RASTERFLEX clients and demos should be added to the execution search path for all users who wish to use them. For a default installation, the following commands add the appropriate directory to the **`PATH`** environment variable:

```
setenv PATH $X_INSTALL/bin:$PATH
rehash
```

If the RASTERFLEX software release was loaded under the single directory `/usr/rfx`, the following commands could be used:

```
setenv PATH /usr/rfx/bin:$PATH
rehash
```

5.2.4. LOCATING MANUAL PAGES

It is necessary to add the location of these pages to the **`MANPATH`** environment variable in order for them to be located properly. For example, if the entire RASTERFLEX software release was loaded under the directory `/usr/X11R5`, the following command could be used:

```
setenv MANPATH /usr/X11R5/man:$MANPATH
```

After this, the **`man`** command is able to locate the RASTERFLEX manual pages.

5.2.5. SETTING UP APPLICATION DEFAULTS

Many of the clients and demos included in the RASTERFLEX software release have X Application Defaults files which specify the default settings for application resources. The standard locations for these files is in the directory `$X_INSTALL/lib/X11/app-defaults`. If you did not load the RASTERFLEX software release in the standard locations, you need to set the **`XAPPLRESDIR`** environment variable to point to the directory in which these files reside. For example, if you elected to load the entire software release under the directory `/usr/rfx`, the following command could be used for this purpose:

```
setenv XAPPLRESDIR /usr/rfx/lib/X11/app-defaults
```

5.3. THE RASTERFLEX X11R5 SERVER (Xrfx)

The following sections describe the features of the X Window System server (**`Xrfx`**) for the RASTERFLEX cards. The RASTERFLEX software release includes an X Window System™ server that supports the MIT X Consortium's X11R5 implementation of the windowing system on your SPARCstation. The **`Xrfx`** server has been enhanced to utilize the unique hardware acceleration and display capabilities of the RASTERFLEX hardware.

The **`Xrfx`** server uses the unique acceleration and display features of the RASTERFLEX hardware transparently. As a result, the application can take advantage of these features without having to be modified. An application can simply use standard Xlib graphics primitives, and these operations are automatically accelerated by the hardware, if applicable.

Other capabilities, such as the use of overlays and shared memory image and pixmap operations, require explicit selection by the application. Examples of using these capabilities within a simple X program are provided in the section on **RASTERFLEX ADVANCED FEATURES**.

The **`Xrfx`** server supports all RASTERFLEX framebuffers: the RASTERFLEX-24, the RASTERFLEX-32, and the RASTERFLEX-HR. Additionally, the **`Xrfx`** server supports the standard Sun framebuffers which were supported in the original X11R5 Sun Sample Server (the CG3, CG4, CG6/GX, and BW2).

5.3.1. INVOKING THE X11R5 SERVER

This section describes how to invoke the Connectware **`Xrfx`** server for the RASTERFLEX card. It includes certain requirements that must be satisfied, as well as several methods of invoking the X server on the host system, including

- using the X initializer (**`xinit`**),
- using the X display manager (**`xdm`**), and
- using the Sun `openwin` start-up script.

5.3.1.1. REQUIRED CONDITIONS

Before attempting to invoke the X11 Server, the following conditions must be satisfied:

- You must have a set of valid fonts loaded on the host system. By default, these fonts are loaded in the directories `$X_INSTALL/lib/X11/fonts/100dpi`, `/$X_INSTALL/lib/X11/fonts/75dpi`, and `$X_INSTALL/lib/X11/fonts/misc`. If they are installed in another location, you must use the `-fp` font path option when starting the server.
- You must have a valid RGB database loaded on the host system. The RGB database consists of the two files `rgb.dir` and `rgb.pag`, which allow the X server to map logical color names (such as "violet") to an associated Red/Green/Blue triplet. The default location for these files is `$X_INSTALL/lib/X11`. If they

are installed in another location, you must use the `-co` color database option when starting the server.

- You may not be running Sunview or any other windowing environment while running the **`Xrfx`** Server. Because the X server requires dedicated use of the host input devices, the X server should be invoked only when the system console is in raw console mode.
- You should have the location where the Connectware X11 binary files are stored (`$X_INSTALL/bin/` by default) in your search path.

5.3.1.2. USING XINIT - THE X SYSTEM INITIALIZER

The X Window System Initializer (**`xinit`**) allows you to customize your environment for invoking the X server and to bring up several initial X clients.

The **`xinit`** command program starts the X Window System server and a first client program (usually a terminal emulator or window manager). When the first client program exits, **`xinit`** kills the X server and then terminates.

5.3.1.3. COMMAND FORMAT

The basic format of the **`xinit`** command is:

```
    xinit [[client]options][--[server] [display] options]
```

5.3.1.4. EXAMPLE COMMAND LINE

A simple example of using **`xinit`** is illustrated in the following command line:

```
    xinit $X_INSTALL/bin/xterm -- $X_INSTALL/bin/Xrfx
```

This command line starts up the Connectware RASTERFLEX X11 server executable, then brings up the **`xterm`** terminal emulator client. Then the system user can bring up other client applications using the terminal emulator. When finished using the window system, the user can exit from the emulator window by logging out, and the server is shut down also, since **`xinit`** assumes that the user session has ended when control is returned from the client program or the `.xinitrc` script (see below).

If no specific client is specified on the command line, **`xinit`** looks for a file called `.xinitrc` in the user's home directory. This file usually is a shell script containing a series of command and client programs to execute at window system start-up.

5.3.1.5. EXAMPLE .xinitrc FILE

An example `.xinitrc` file follows:

```
  #! /bin/csh
  #! Sample client initialization script
  xclock &
  xterm &
  twm
```

This script starts up the X clock program, a terminal emulator window, and the Tab Window Manager, starting all commands — except the final one — in background mode. As a result, the shell script is not exited from until the **twm** program stops executing. When the user exits from the window manager (using an *Exit* option from a root menu), the window system shuts down automatically.

Always place the client from which you exit the window system — usually a terminal emulator window or window manager — as the final command in your **xinit** client initialization file and ensure that the process is not started as a background job.

If you do not specify a client program on the command line and a `.xinitrc` file does not exist, then **xinit** uses the default command:

```
xterm -geometry +1+1 -n login -display :0
```

If you do not specify a server program on the command line, then **xinit** looks for a file, called `.xserverrc`, in the user's home directory. It runs this file as a shell script to start up the server.

5.3.1.6. EXAMPLE SERVER INITIALIZATION FILE

An example server initialization file for the Connectware X11 server follows:

```
#! /bin/csh
#! Server initialization shell script
$X_INSTALL/bin/Xrfx
```

This simple script starts the Connectware RASTERFLEX X11 server.

If you do not specify a server command on the command line and you do not specify that a `.xserverrc` file does not exist, then **xinit** uses the following command to start the X server:

```
X :0
```

The X Window System initializer, **xinit**, provides a simple, yet flexible, means for invoking the window system server and a set of client applications.

5.3.1.7. USING OPENWIN - THE OPENWINDOWS START-UP SCRIPT

The **openwin** script which is provided as part of the OpenWindows software release from SunSoft can also be used to invoke the RASTERFLEX X server. This is done by setting the **SERVER** environment variable to a string which should be used to initiate the **Xrfx** server. For example, if the server and libraries were stored in the default locations, the following sequence could be used:

```
setenv SERVER “$X_INSTALL/bin/Xrfx”
openwin
```

If any additional command line options are passed to the **Xrfx** server, they could be included (inside the double quotes) within the definition of the SERVER variable.

When starting the server using the **openwin** script, the following message may appear:

```
    svenv: can‘t get SunView environment information
```

This is due to the fact that the **Xrfx** server does not support running SunView applications concurrently with the X Window System environment. This message can be ignored with no adverse effects.

5.3.1.8. USING XDM - THE X DISPLAY MANAGER

The X Display Manager (**xdm**) program manages a collection of X displays. It is designed to provide services similar to those provided by **init**, **getty**, and **login** on character terminals:

- Prompting for login/password,
- Authenticating the user, and
- Running a ‘session’.

The X Display Manager starts up the X server which is under its control, and displays a login / password prompt. No other client application can connect to the server when the login prompt is displayed.

After you successfully login, the server initiates a session, which may be a terminal emulator, a window manager, or some application-specific program that the user is running. Once the session is over, and the user exits from the terminal emulator, window manager, or program, the X display manager resets the X server and redisplays the login / password prompt.

For complete information and options on configuring the X Display Manager, refer to the **xdm** manual page.

5.3.2. X11R5 (Xrfx) SERVER OPTIONS

The following is a summary of all command line options available when invoking the `Xrfx` Server:

`-a` *`n`* — Set pointer acceleration.

The pointer acceleration value allows the pointer (mouse) movements to be accelerated (multiplied) by some factor if they are larger than some threshold value. The threshold parameter can be set by the `-t` option. The default acceleration value is *`4`*.

`-ar1` *`milliseconds`* — Set auto-repeat initiation time.

The auto-repeat initiation time specifies the number of milliseconds which a key must remain down before auto-repeat operations are initiated. The default value is *`200`* milliseconds. This option currently is supported only for Sun hosts.

`-ar2` *`milliseconds`* — Set auto-repeat separation time.

The auto-repeat separation time specifies the number of seconds between key event generation once auto-repeat is initiated. The default value is *`50`* milliseconds. This option currently is supported only for Sun hosts.

`-auth` *`authorization-file`* Set authorization file.
The authorization file is used to enable per-user access to the X server. The file contains a collection of records used to authenticate access. The default behavior is to use host-based authorization.

`bc` Enable bug compatibility.
The bug compatibility mode allows applications which generate questionable protocol streams, such as setting undefined bits in a mask value, to operate without generating a protocol error. Many older X toolkits and applications require this mode to operate properly. This mode is set by default.

`-broadcast` Broadcast for XDMCP.
Refer to the XDM manual pages for more information.

`-bs` Disable backing store.
Explicitly disables backing store operations on all screens.

`-c` Disable key click.
Disables key click on the keyboard device. Key click is enabled by default.

`c` *`volume`* Set key click volume.
Sets the key click volume, if key click is enabled. Valid values are in the range 0-100. The default value is *`0`*.

`-cc` *`class`* Select default visual class.
Set the default visual class for the server (SELECTING DEFAULT COLOR CLASS on page 5.14))

`-class` *`display-class`* Set the display class to send in manage.
Refer to the XDM Manual pages for more information.

`-co` *`filename`* Set color database file.
Set path name of the RGB data base which is used to translate color names to numeric RGB values. The default data base is `$X_INSTALL/lib/X11/rgb`. **This option is required if the RGB database has not been installed at this location.**

`-dd` *`n`* Set server default depth.
(See SELECTING DEFAULT WINDOW DEPTH on page 5.14)

`-dev` *`filename`*`[:filename]` Set framebuffer device name.
Open the specified file name as the framebuffer. This option can occur multiple times on the command line, in which case each framebuffer is registered as a separate *screen* controlled by the single *display* created by running `Xrfx`. The following types of framebuffers are supported:

RASTERFLEX-24, RASTERFLEX-32, RASTERFLEX-HR, Sun CG3, Sun CG4, Sun CG6 (GX), Sun BW2.

`-displayID` *`display-id`*	Set manufacturer display ID for request.

Refer to the XDM manual pages for more information.

`-dpi` *`n`*	Set screen dots per inch.

Set the dots-per-inch value that the server reports to applications.

`-fc` *`string`*	Set cursor font.

Set the name of the font to use for "font cursors"

`-fn` *`string`*	Set default font name.

Set the name of the font to use when applications do not specify one.

`-fp` *`pathname`*	Set default font path.

A comma separated list of directories for font directory files named `fonts.dir`. These directory files map logical font names to actual PCF font files. The default value is: `$X_INSTALL/lib/X11/fonts/misc,$X_INSTALL/lib/X11/fonts/100dpi,$X_INSTALL/lib/X11/fonts/75dpi.` **This option is required if the fonts have not been installed at this location.**

`-f` *`n`*	Set the bell volume.

Sets bell volume (allowable values 0-7).

`-help`	Print usage summary.

Prints a summary of all command line options.

`-I`	Ignore remaining options.

Ignore all command line options following `-I`.

`-indirect` *`host-name`*	Select host for indirect XDMCP.

Refer to the XDM manual pages for more information.

`-ld` *`n`*	Limit server data space to *n* Kilobytes.
`-logo`	Enable X Logo for Screen Saver.

The `logo` option turns on the X Window System logo display in the screen saver. This is the default value.

`nologo`	Disable X Logo for Screen Saver.

The `nologo` option turns off the X Window System logo display in the screen saver.

`-mincmaps`	Advertise minimal installed colormaps.

The default is to advertise 255 installed colormaps. The server will advertise 255 installed colormaps because the RasterFlex hardware can support an unlimited number of 24-bit TrueColor and 8-bit StaticGray colormaps. If this option is specified, the server will report the minimal number of installed colormaps.

`-once`	Terminate server after one session.
`-overlay4`	Set overlays to 4-bit depth (RFX-32/HR)

(See 4-BIT OVERLAY MODE on page 5.17) This is the default setting.

`-overlay8` Set overlays to 8-bit depth (RFX-32/HR)
(See 8-BIT OVERLAY MODE on page 5.18)

`-p` *`minutes`* Set screen saver cycle time.
The pattern time option allows the screen-saver pattern cycle time to be explicitly set. The default value is *`10`* minutes.

`-port` *`port-num`* Set UDP port number to send messages to.

`-query` *`host-name`* Select named host for XDMCP.
See the XDM manual pages for more information.

`-r` Disable auto repeat.
This option disables auto repeat for the keyboard device.

`r` Enable auto repeat.
This option enables auto repeat for the keyboard device. This is the default setting.

`-s` *`minutes`* Set screen saver time-out.
The screen saver time-out option sets the screen saver time-out value in minutes. If the server input devices are not used for the time-out value, the screen saver function is initiated, unless it has been explicitly disabled. A value of *`0`* disables the screen saver. The default value is *`10`* minutes.

`-screen` *`number`* Set screen-specific options
This options causes any subsequent screen-specific options (-cc, -dd, etc) to apply only to the screen specified by *number* where *number* will vary from 1 to the number of available screens.

`-su` Disable save under support.
This option disables support for **`save under`** operations on all screens.

`-sunsupport` Support standard Sun framebuffers.
This option specifies that the server should also open and use any of the standard Sun framebuffers which it is capable of supporting.

`-t` *`n`* Set pointer acceleration threshold.
The pointer threshold option sets the number of pixels which the pointer must move in order for pointer acceleration to be enabled. The default value is *`4`* pixels.

`-to` *`seconds`* Set connection time-out value.
The connection time-out option sets the maximum number of seconds which the server will wait to establish a connection with a client program. The default value is *`60`* seconds.

`-v` Sets video-on screen saver preference.
The video-on screen saver option ensures that the screen video is not blanked when screen saver operations are initiated.

`v`	Sets video-off screen saver preference.

The video-off screen saver option causes the screen video to be blanked when screen saver operations are initiated.

`-wm`	Enable backing store on all mapped windows.

The `when-mapped` backing store option causes backing store to be enabled for all mapped windows. Use of this option is not recommended, as it can result in excessive allocation of host memory resources.

5.3.3. USING MULTIPLE SCREENS

The RASTERFLEX server permits the use of multiple screens/framebuffers controlled by a single invocation of the server. The server automatically locates each RASTERFLEX device which is installed and initializes it as a unique screen. Additionally, the RASTERFLEX server locates and initializes many standard Sun framebuffers when the `-sunsupport` command line option is specified.

The following types of devices are supported by the RASTERFLEX:

- RASTERFLEX-24 (`/dev/rfx`*n*)
- RASTERFLEX-32 (`/dev/rfx`*n*)
- RASTERFLEX-HR (`/dev/rfx`*n*)
- Sun CG3 (`/dev/cgthree`*n*)
- Sun CG4 (`/dev/cgfour`*n*)
- Sun CG6 (`/dev/cgsix`*n*)
- Sun BW2 (`/dev/bwtwo`*n*).

To cause the **`Xrfx`** server to use a single device, even though multiple devices are present, the `-dev` option can be used to select a specific device to run on.

In order to start a client on a specific screen, the **`DISPLAY`** environment can be set using the notation `[node]:[server].[screen]`. For example, to start a client on screen 1 of a locally running X server, the **`DISPLAY`** variable could be set to "`unix:0.1`".

5.3.4. RASTERFLEX COMPATIBILITY ISSUES

X Window System applications should always be able to work with any compliant X Window System Server. However, many of the unique display capabilities of the RASTERFLEX-24, RASTERFLEX-32, and RASTERFLEX-HR devices is not anticipated by many existing applications. This section provides some tips on how to circumvent some common problems with such applications.

5.3.4.1. VISUAL SELECTION

The X Window System allows a server to advertise the full range of display capabilities which it is capable of supporting by advertising a set of visuals.

An X visual is a combination of display depth and color class which define specifically how an application should view the pixel data which is being manipulated.

For example, an 8-bit `PseudoColor` visual specifies that there are 8 bits of significant pixel data and that this data is arbitrarily mapped through a modifiable color Look-Up Table which maps the 8 bits of data to the displayed red, green, and blue values.

A 24-bit `TrueColor` visual specifies that there are 24 bits of significant pixel data, and that the data is further subdivided into three separate channels, one each for red, green, and blue, and that each channel is mapped through a fixed color lookup operation before display.

The RASTERFLEX-32 and RASTERFLEX-HR devices are unique in that they advertise a wide variety of visual classes and depths; whereas most common framebuffers support just a single depth, but usually multiple classes. The RASTERFLEX-24 device will also support multiple depths; however, for this device the display depth is selected at server start-up and only windows of a single depth can be created.

All X Window System servers also define a default visual, which is the visual type from which the server root window is derived. The compatibility problem which arises is that many applications simply utilize the default visual without examining the full set of visuals which are supported by the server to determine the one which is most appropriate for that application’s purposes. A more sophisticated application might examine the full set of available visuals and decide to create its windows using the one which best meets its needs.

The RASTERFLEX-32/HR framebuffers are fully capable of displaying windows of different visual types on the screen simultaneously; however, the less sophisticated application will often only utilize the server default. For example, an image display program may only display data in 8-bit `PseudoColor` mode (if that is the default) even though 24-bit `TrueColor` display capabilities are also available.

To address this issue, the RASTERFLEX server supports command line options in order to configure the **default** behavior of the server so as to match the **assumed** behavior made by an application program.

This approach, unfortunately, has the following drawbacks:

- The default behavior for one application can be completely inappropriate for another application — in fact it can cause the other program to not operate at all. In the latter case, the user must stop and restart the server in order to use the second application.
- The default behavior can entail large inefficiencies for other applications, lowering their performance or visual appeal. An example of this would be running an application which uses a small number of distinct colors in 24-bit mode.
- The user must explore internal details of each application and experiment with server configurations until it works.

For the RASTERFLEX-32 and RASTERFLEX-HR cards, the server defaults to a common, 8-bit `PseudoColor` configuration. Most recent color applications work in this mode. For the RASTERFLEX-24 card, the server defaults to a 24-bit `TrueColor` configuration.

5.3.4.2. SELECTING DEFAULT WINDOW DEPTH

The X11R5 server supports *both* 8- and 24-bit depth windows on the same display for the RASTERFLEX-32 and RASTERFLEX-HR devices. For the RASTERFLEX-24 device, it will allow selection of either 8-bit or 24-bit depth for all windows at server start-up.

You may set the *default* depth for the server (the depth of the background pattern and the depth of windows created for applications that do not explicitly select a depth) with the `-dd` option as follows:

`Xrfx -dd 8` [Default for RASTERFLEX-32/HR]

or

`Xrfx -dd 24` [Default for RASTERFLEX-24]

5.3.4.3. SELECTING DEFAULT COLOR CLASS

Each window supports one of a number of different color schemes or classes. These classes are the standard X Window color classes. The RASTERFLEX-32/HR supports color classes on a window-by-window basis. This provides the following advantages:

- Applications that require a specific color class will work with the RASTERFLEX-32/HR card.
- Fewer server resources must be shared between applications, which helps reduce the colormap flashing that sometimes occurs when switching between applications.
- Applications can choose window depth and color class on a per-window basis rather than being forced to stick with the server default at all times.

Once again, you can choose the server default with a command line option of `-cc`:

`-cc PseudoColor` [The default value for -dd 8]
PseudoColor windows use a color Look-Up Table to convert pixel values in memory to color intensities on the screen. The RASTERFLEX-32 card supports PseudoColor for 8- and 4-bit depth color maps.

`-cc StaticColor`
Similar to PseudoColor above, but the color Look-Up Table cannot be modified. Instead, a selection of 256 colors spanning the RGB space is pre-loaded into the Look-Up Table.

`-cc StaticGray`
The pixel value is interpreted directly as an intensity without going through a color Look-Up Table. The intensity is applied equally to the Red, Green and Blue components, creating shades of gray. StaticGray is supported for 8-bit color maps only.

`-cc GrayScale`

This is similar to PseudoColor, except there is only one intensity (applied equally to Red, Green and Blue as in StaticGray) per entry in the color Look-Up Table. 8-bit color maps only.

`-cc DirectColor`

This method is supported for 24-bit windows only. It is like the PseudoColor class for 8-bit windows, but each of the Red, Green and Blue components is separately indexed from a part of the 24-bit pixel.

`-cc TrueColor` [The default value for -dd 24]

In this method, like StaticGray above, there is no Look-Up Table. But unlike StaticGray, a different intensity is used for each component. The value is taken from three different parts of the 24-bit pixel.

`Xrfx -cc` *n*

The number *n* is used to specify the color class. The relationship between the number and the color class is defined in the standard X11 header file `X.h`.

5.3.5. USING THE VISUAL SELECTION EXTENSION

The RASTERFLEX X11R5 Server supports a special Connectware-developed extension called the Visual Selection extension which will allow clients which normally utilize the default visual of the server to be run using any of the available server visual types. The Visual Selection Extension allows a user to specify that one of the available visuals be advertised to a client as if the selected visual were actually the server default. For example, a user could start the server with the default visual type set to 8-bit `PseudoColor` so that all standard clients such as terminal windows and other utilities run in this manner, then use the Visual Selection extension to cause a raster display application to be run as if the server default was set to 24-bit `TrueColor`.

The Visual Selection Extension is accessed via a special client called **`vset`** which allows the user to specify the visual which will be advertised to the next client program (after **`vset`**) which connects to the RASTERFLEX server. The parameters to **`vset`** are:

```
vset <visual id>
        or
vset <visual class> [visual depth]
```

where <visualid> refers to the specific id of the visual to be advertised as the server default or <visual class> is one of the valid X visual classes (`PseudoColor`, `TrueColor`, `StaticGray`, etc.) and <visual depth> is a numeric value representing the desired visual depth. For example, to start an application named **`appl`** so that it believes the server default visual type is 24-bit `TrueColor,` the following sequence would be used:

```
vset TrueColor 24
appl
```

Immediately after the **appl** client is started, the default visual type advertised by the server will revert to the true server default (as specified at start-up via the command line options described above).

The Visual Selection Extension does not modify the server default visual, nor does it change any of the attributes of the root window (which always will be of the true server default visual class/depth). For this reason, clients which attempt to perform operations directly upon the root window with an altered notion of the default visual will not work properly. Applications which currently fall into this category include window managers (**`twm, mwm, olwm`**) and screen dump utilities (**`xmag`**, **`xwd`**).

> **NOTE**
> The Visual Selection Extension is only a workaround for existing applications which do not properly select the desired visual type or allow explicit visual selection by the end-user. Some applications (as noted above) may not operate properly when using an altered notion of the default visual via the Visual Selection extension.

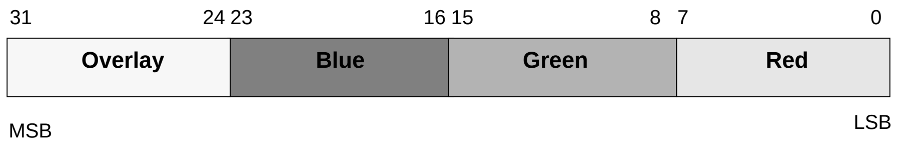
5.3.6. OVERLAY MODE SELECTION

The RASTERFLEX X server is also different from most common framebuffers in that it supports two distinct layers of display hierarchy — a set of overlay planes and a set of underlay planes.

> **NOTE**
> For most existing applications, the existence of overlay planes is of no concern and the application will run without any problems. However, the manner in which the overlay planes are configured (see below) can have an impact on the manner in which colormaps and other resources are utilized, so users may benefit from an understanding of these issues. For application developers who do want to take advantage of the overlay capabilities of the RASTERFLEX hardware, a programmatic example is provided in a subsequent section.

The framebuffer within the RASTERFLEX-32 and RASTERFLEX-HR devices actually contains 32 bits of data for each displayed pixel. The most significant 8 bits of the 32 bits of pixel data constitute the overlay plane while the remaining 24 bits are used as the underlay plane to display either 8 or 24-bit data. Windows can be created in either the underlay planes or the overlay planes, both never both. The decision about which planes a window resides in is strictly a function of the visual type with which the window has been created (more on this later).

Figure 5.1. Hardware pixel format.



Overlay windows are regular windows with the added feature of transparency on a pixel-by-pixel basis. The term “overlay” is used since generally one wishes the transparent window to be on top of some other window, referred to as the “underlay”. Drawing into the overlay window does not destroy data in the underlay.

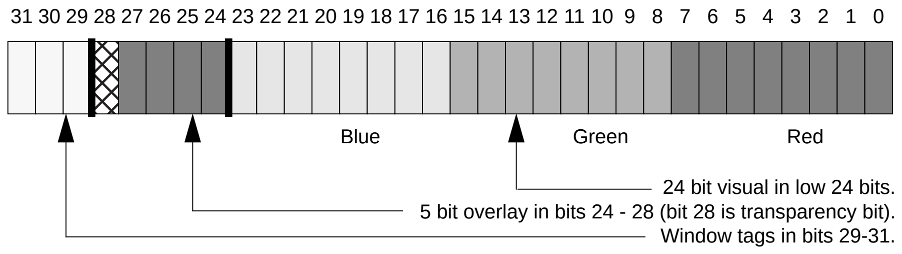
The 8-bit overlay within the RASTERFLEX-32/HR can be configured in one of two manners: 4-bit Overlay Mode or 8-bit Overlay Mode. The mode used is selected at server start-up via the **`-overlay4`** and -**`overlay8`** command line options to **`Xrfx`**. If neither option is specified, the default behavior is to advertise 4-bit overlay capabilities.

5.3.6.1. 4-BIT OVERLAY MODE

In 4-bit Overlay Mode, the 8-bit overlay pixel is actually broken into three separate components:

- a 4-bit color value (hence the name),
- a 1-bit transparency control bit, and
- 3 bits of control information.

Figure 5.2. Hardware pixel format — 4-bit overlay model.


31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Blue
Green
Red
24 bit visual in low 24 bits.
5 bit overlay in bits 24 - 28 (bit 28 is transparency bit).
Window tags in bits 29-31.

The four bits of color value in the 4 least significant bits of the overlay allow 16 distinct overlay colors to be available. The transparency control bit allows the visibility of the overlay pixel to be controlled on a pixel-by-pixel basis.

If this bit is set, the overlay pixel is transparent, and the contents of the underlay planes beneath the window become visible. If the bit is clear (initial state), the pixel is considered opaque, and the displayed color is selected, based upon the mapping of the 4 bits of color data through the currently installed Look-Up Table for the overlay planes (see Multiple Look-Up Table Usage below).

> **NOTE**
> The transparency of a given pixel can be changed without making any changes to the four bits of color data by simply toggling the value of the transparency bit. This enables overlay data to be non-destructively turned on and off.

In 4-bit Overlay Mode, the data within the underlay planes can be either 8-bit or 24-bit. The remaining three control bits are used to determine the format of the underlay data based upon its visual type. The values within the control bits are maintained entirely by the **`Xrfx`** server and are not accessible to application programs.

5.3.6.2. 8-BIT OVERLAY MODE

In 8-bit Overlay Mode, all 8 bits of the overlay pixel are used in determining the displayed value for the overlay planes. Transparency can still be selected on a pixel-by-pixel basis in this mode by storing a value of zero (0) within the overlay planes. Two overlay colors also are reserved for cursor display, leaving a total of 253 colors available for application usage.

When in 8-bit Overlay Mode, only 24-bit data can be stored within the underlay planes. This is necessary due to the fact that no control bits are left over for selection of multiple underlay display formats. Additionally, 8-bit Overlay Mode has implications on the color Look-Up Table allocation (see below).

5.3.7. MULTIPLE LOOK-UP TABLE MANAGEMENT

The RASTERFLEX-32 and RASTERFLEX-HR framebuffers are different from most common framebuffers in that they also provide multiple hardware color Look-Up Tables; most framebuffers only have one.

The RASTERFLEX framebuffers actually support two full 256-entry color Look-Up Tables plus a 16-entry Look-Up Table. Additionally, they have the capability to display 24-bit `TrueColor` or 8-bit `StaticGray` data without using any Look-Up Table space.

The following paragraphs describe the manner in which these Look-Up Tables are managed by the **`Xrfx`** server.

> **NOTE**
> The management of hardware color Look-Up Tables places no limitations on the number of X Colormaps which can be created. Its only impact is the manner in which X Colormaps are downloaded to the hardware as a result of colormap installation requests from a client application or window manager.

The manner in which the overlay planes have been configured at server start-up has a major implication on the manner in which colormaps are managed.

If the server is started in 8-bit Overlay Mode, the colormap handling is relatively fixed. One 256-entry color palette is reserved for the 8-bit overlay planes, and the other 256-entry palette is used for the 24-bit underlay planes. The 16-entry Look-Up Table is not utilized.

In 8-bit Overlay Mode, the hardware color Look-Up Tables always contain the most recently installed X colormap which is associated with a visual of the proper type. One Look-Up Table contains the most recently installed overlay visual colormap, and the other always contains the most recently installed underlay visual colormap.

Hardware Look-Up Table management is more complex when using 4-bit overlay mode, and also significantly more flexible.

In 4-bit Overlay Mode, one of the 256-entry color Look-Up Tables is reserved for the default colormap which is advertised to all clients. This colormap always remains installed, meaning that applications which use it are never subjected to colormap flashing. The intention is that applications which use only a few colors, such as the standard X clients, are always displayed with their proper colors.

The other 256-entry colormap is available for use by applications which create and install (or request installation via a window manager) their own colormaps. This is common for many raster display programs which require a large number of colors to operate. This hardware color Look-Up Table always contains the most recently installed underlay visual colormap (other than the default). The 16-entry Look-Up Table is used for the most recently installed colormap associated with an overlay visual.

As an additional bonus, X colormaps which are associated with either the 24-bit `TrueColor` or 8-bit `StaticGray` visual types are displayed using a special "pass-through" mode which actually uses no space within the hardware Look-Up Tables. For these visual types, the actual data within the framebuffer is passed around the color Look-Up Tables and directly drives the displayed red, green, and blue values.

This effectively means that windows using the default colormap, an application-defined underlay colormap, an application-defined overlay colormap, and any using 24-bit `TrueColor` or 8-bit `StaticGray` can all be displayed simultaneously in 4-bit overlay mode with no visual anomalies due to colormap flashing.

6. RASTERFLEX X11/NEWS ENVIRONMENT

The RASTERFLEX X11/NeWS Windowing Environment consists of an X11/NeWS server (**`xnews-rfx)`** for Solaris 1.X (SunOS 4.1.X) for the RASTERFLEX framebuffers. This server is fully compatible with OpenWindows 3 from SunSoft and provides additional support for the unique capabilities of the RASTERFLEX hardware. This section provides an overview of the software components provided within the RASTERFLEX OpenWindows Environment as well as specific information on how to use the RASTERFLEX X11/NeWS server.

You must have the OpenWindows 3.0, 3.1, or 3.2 software release from SunSoft already loaded on your system to use the RASTERFLEX OpenWindows environment. The **`xnews-rfx`** server depends upon font files, postscript initialization information, and other support items loaded as part of the standard OpenWindows software installation.

Note
This section is intended to be an addendum to the standard OpenWindows 3 documentation from Sun. Its purpose is to describe only those features which are unique to the RASTERFLEX OpenWindows environment. Refer to original documentation and manual pages for more in-depth information on the OpenWindows environment and the X11/NeWS server.

6.1. OPENWINDOWS SOFTWARE COMPONENTS

This section describes the various software components that are provided with the OpenWindows Environment for the RASTERFLEX framebuffers. The RASTERFLEX OpenWindows software release was intended to be integrated seamlessly with the standard OpenWindows 3 release from SunSoft and provides only those items which are different or not supplied in the original release.

Installing the RASTERFLEX software into the standard OpenWindows release tree does not remove or destroy any portions of the original release, nor does it prevent running the original X11/NeWS server on hardware other than the RASTERFLEX framebuffers.

6.1.1. SOFTWARE RELEASE BUTLER

The Software Release Butler is a utility which prompts you with questions regarding the installation, then performs the installation for you. The questions it asks involve such things as which portions of the software you wish to install and where you wish to install them in your file system. The Software Release Butler is invoked automatically when you use the **`rfxinstall`** utility provided on the RASTERFLEX software CD-ROM.

6.1.2. SOFTWARE CONFIGURATION MECHANIC

The Software Configuration Mechanic is a utility which will examine the manner in which you have installed the RASTERFLEX software release on your system, ask you a few basic configuration questions, and then generate a file containing a set of environment variable assignments suitable for inclusion in a user `.cshrc` file. The Software Configuration Mechanic is a shell script named `rfxconfig.sh` which by default is stored in `/etc/modules` under SunOS 4.1.X. Refer to the Software Release notes for more specific information on running the Software Configuration Mechanic.

6.1.3. DEVICE DRIVER

The device driver allows the operating system to communicate with the RASTERFLEX hardware. It is **required** in order to run the remainder of the RASTERFLEX software. Refer to the Software Release notes for more specific information on how to install the device driver.

6.1.4. X11/NEWS SERVER (xnews-rfx)

The **`xnews-rfx`** server is **required** to run the X11/NeWS window environment on the RASTERFLEX card. Connectware has ported the standard OpenWindows 3 Window System Server to the RASTERFLEX hardware, making optimizations to support the special features of the RASTERFLEX. By default, the server software is loaded into `/usr/openwin/bin`.

6.1.5. OPENWINDOWS MANUAL PAGES

This is the manual page for the RASTERFLEX X11/NeWS server. By default, the manual page is installed in `/usr/openwin/man`.

6.2. SETTING UP THE USER ENVIRONMENT

This section outlines the changes which may be required to the user environment after loading the RASTERFLEX software. It is preferable to add the following commands to the system `.cshrc` file or other user start-up files so that these changes apply to all users.

Note that the RASTERFLEX Software Configuration Mechanic, provided as a part of the RASTERFLEX software release, performs basic environment setup automatically. This section is provided for advanced users who wish to customize their environment. The Configuration Mechanic is (by default) installed in `/etc/modules/rfxconfig.sh` under SunOS 4.1.X.

6.2.1. LOCATING SERVER RESOURCES

The RASTERFLEX X11/NeWS server must be able to locate certain resources in order to execute properly. All resources required by the server are located relative to the top of the standard OpenWindows 3 release tree, which is identified by the **`OPENWINHOME`** environment variable. For example, if you have loaded OpenWindows 3 under the directory `/usr/openwin` [the default], then you would use the following C Shell command to set the **`OPENWINHOME`** variable:

```
    setenv OPENWINHOME /usr/openwin
```

6.2.2. LOCATING SHARED LIBRARIES

When running the X11/NeWS server or OpenWindows clients, it is necessary to add the X11, toolkit and server support shared libraries, which are supplied with the standard OpenWindows release, to the library search path. To do this, set the **`LD_LIBRARY_PATH`** environment variable to point to the location where the libraries have been loaded. For example, the following command could be used:

```
setenv LD_LIBRARY_PATH $OPENWINHOME/lib:$LD_LIBRARY_PATH
```

6.2.3. SETTING THE PATH VARIABLE

The location of the X11 and NeWS clients and utilities should be added to the execution search path for all users who wish to use them. For a default installation, use the following commands to add the appropriate directory to the **`PATH`** environment variable:

```
setenv PATH $OPENWINHOME/bin:$PATH
rehash
```

6.2.4. LOCATING MANUAL PAGES

If the OpenWindows manual pages have been loaded in a location other than `/usr/man`, it is necessary to add the location of these pages to the **`MANPATH`** environment variable in order for them to be located properly. The following command could be used for this purpose:

```
setenv MANPATH $OPENWINHOME/man:$MANPATH
```

After this, the **`man`** command is able to locate the OpenWindows manual pages.

6.3. THE RASTERFLEX X11/NEWS SERVER

The following paragraphs describe the features of the X11/NeWS Window System server (**`xnews-rfx`**) for Solaris 1.X (SunOS 4.1.X) for the RASTERFLEX cards. The RASTERFLEX OpenWindows software release includes a server that supports the OpenWindows 3 implementation of the X11/NeWS windowing system on your SPARCstation. The **`xnews-rfx`** server has been enhanced to utilize the unique hardware acceleration and display capabilities of the RASTERFLEX hardware.

The **`xnews-rfx`** server uses the unique acceleration and display features of the RASTERFLEX hardware transparently. As a result, your application can take advantage of these features without having to be changed. An application can simply use standard Xlib or NeWS graphics primitives, and these operations, if applicable, are accelerated automatically by the hardware.

Other capabilities, such as the use of overlays and shared memory image and pixmap operations, require the application to select them explicitly. Examples of using these capabilities within a simple X program are provided in a subsequent section.

The **xnews-rfx** server supports both RASTERFLEX framebuffers: the RASTERFLEX-24, the RASTERFLEX-32 and the RASTERFLEX-HR. Additionally, the **xnews-rfx** server supports many of the standard Sun framebuffers which were supported in the original OpenWindows 3 release (the CG3, CG4, CG6, CG8, and BW2).

6.4. INVOKING THE X11/NEWS SERVER

This section describes how to invoke the Connectware **xnews-rfx** server for the RASTERFLEX card. It includes certain requirements that must be satisfied, as well as several methods of invoking the X11/NeWS server on the host system, including

- using the Sun `openwin` start-up script,
- using the X initializer (**xinit**), and
- using the X display manager (**xdm**).

6.4.1. REQUIRED CONDITIONS

Before attempting to invoke the RASTERFLEX X11/NeWS Server for Solaris 1.X (SunOS 4.1.X), the following conditions must be satisfied:

- You must already have installed the OpenWindows 3 software release from SunSoft onto your system. Additionally, you should set the **OPENWINHOME** environment variable to point to the top of the OpenWindows 3 release tree.
- You cannot be running Sunview or any other windowing environment while running the **xnews-rfx** Server. Because the server requires dedicated use of the host input devices, the X11/NeWS server should be invoked only when the system console is in raw console mode.
- You should have the location where the OpenWindows executables are stored (`$OPENWINHOME/bin` by default) in your search path.
- You should have the location where the OpenWindows shared libraries are stored (`$OPENWINHOME/lib` by default) in your library load path.

6.4.1.1. USING OPENWIN - THE OPENWINDOWS START-UP SCRIPT

The **openwin** script, which is provided as part of the OpenWindows software release from Sun, can be used to invoke the RASTERFLEX X11/NeWS server. This is done by setting the **SERVER** environment variable to a string that is used to initiate the **xnews-rfx** server. For example, if the server and libraries were stored in the default locations, the following sequence could be used:

```
setenv SERVER "/usr/openwin/bin/xnews-rfx"
openwin
```

If any additional command line options are passed to the **xnews-rfx** server, they could be included (inside the double quotes) within the definition of the *SERVER* variable.

When you use the **openwin** script to start the server, the following message may appear:

```
svenv: can't get SunView environment information
```

This is due to the fact that the **`xnews-rfx`** server does not support running SunView applications concurrently with the X Window System environment. This message can be ignored with no adverse effects.

6.4.1.2. USING XINIT - THE X SYSTEM INITIALIZER

The X Window System Initializer (**`xinit`**) allows you to customize your environment for invoking the X11/NeWS server and to bring up several initial X clients.

The **`xinit`** command program starts the X11/NeWS Window System server and a first client program (usually a terminal emulator or window manager). When the first client program exits, **`xinit`** kills the server and then terminates.

6.4.1.3. COMMAND FORMAT

The basic format of the **`xinit`** command is:

```
    xinit [[client]options][--[server] [display] options]
```

6.4.1.4. EXAMPLE COMMAND LINE

A simple example of using **`xinit`** is illustrated in the following command line:

```
    xinit xterm -- /usr/openwin/bin/xnews-rfx
```

This command line starts up the Connectware RASTERFLEX X11/NeWS server executable, then brings up the **`xterm`** terminal emulator client. Then you can bring up other client applications using the terminal emulator. When finished using the window system, you can exit from the emulator window by logging out, and the server is shut down also, since **`xinit`** assumes that the user session has ended when control is returned from the client program or the `.xinitrc` script (see below).

If no specific client is specified on the command line, **`xinit`** looks for a file, called `.xinitrc`, in the user's home directory. This file usually is a shell script containing a series of command and client programs to execute at window system start-up.

6.4.1.5. EXAMPLE .xinitrc FILE

The following is an example `.xinitrc` file:

```
  #! /bin/csh
  #! Sample client initialization script
  xclock &
  xterm &
  olwm
```

This script starts up the X clock program, a terminal emulator window, and the OpenLook Window Manager, starting all commands — except the final one — in background mode. As a result, the shell script is not exited from until the **`olwm`** program stops executing. When you exit from the window manager (using an *`Exit`* option from a root menu), the window system shuts down automatically.

Always place the client from which you exit the window system — usually a terminal emulator window or window manager — as the final command in your **`xinit`** client initialization file and ensure that it is not started as a background job.

If you do not specify a client program on the command line and a `.xinitrc` file does not exist, then **`xinit`** uses the default command:

```
xterm -geometry +1+1 -n login -display :0
```

If you do not specify a server program on the command line, then **`xinit`** looks for a file, called `.xserverrc`, in the user's home directory. It runs this file as a shell script to start up the server.

6.4.1.6. EXAMPLE SERVER INITIALIZATION FILE

An example server initialization file for the Connectware X11/NeWS server follows:

```
#! /bin/csh
#! Server initialization shell script
/usr/openwin/bin/xnews-rfx
```

This simple script starts the Connectware RASTERFLEX X11/NeWS server.

If you do not specify a server command on the command line and you do not specify that a `.xserverrc` file does not exist, then **`xinit`** uses the following command to start the X server:

```
xnews :0
```

The X Window System initializer, **`xinit`**, provides a simple, yet flexible, means for invoking the window system server and a set of client applications.

6.4.1.7. USING XDM - THE X DISPLAY MANAGER

The X Display Manager (**`xdm`**) program manages a collection of X displays. It is designed to provide services similar to those provided by **`init`**, **`getty`**, and **`login`** on character terminals:

- Prompting for login/password,
- Authenticating the user, and
- Running a 'session'.

The X Display Manager starts up the X server, which is under its control, and displays a login / password prompt. No other client application can connect to the server when the login prompt is displayed.

After you successfully login, the server initiates a session, which can be a terminal emulator, a window manager, or some application-specific program that the user is running. Once the session is over, and the user exits from the terminal emulator, window manager, or program, the X display manager resets the X server and redisplays the login / password prompt.

For complete information and options on configuring the X Display Manager, refer to the **`xdm`** manual pages that are supplied with the Connectware software release.

6.4.2. X11/NEWS (xnews-rfx) SERVER OPTIONS

The basic command syntax when invoking the **`xnews-rfx`** server is:

```
xnews-rfx [serveroptions] [[-dev device] [deviceoptions]] ...
```

The **`xnews-rfx`** server supports two basic types of options: *`serveroptions`* and *`deviceoptions`*. The *`serveroptions`* are global options that affect the server for all display screens. The *`deviceoptions`* are options that apply only to a specific framebuffer device which the server is controlling.

6.4.2.1. SERVER OPTIONS

`:display` Set display number

This options specifies the display number upon which the server listens for X11 and NeWS connections. The default values is *`:0`*.

`-auth` *`authorization-file`* Set authorization file.

The authorization file is used to enable per-user access to the server. The file contains a collection of records used to authenticate access. The default behavior is to use host-based authorization.

`-cubesize small | large` Set static color cube size.

Sets the color cube size for `StaticColor` colormaps which are defined by the server.

`-escape` Allow “hot-key” exit.

Allows the user to forcibly exit the window system server by holding down the following sequence of keys: L1-Alt-Delete.

`-defeateventsecurity` Disable synthetic event security.

Tells the server to disable the security feature which detects synthetically created events. All events distributed by the server, even if created by a client program, will have a synthetic field of false. Its use is not recommended, unless running the journaling demo or other applications which need to generate synthetic events.

`-fp` *`pathname`* Set default font path.

A comma separated list of directories for fonts. These directories must contain files which map logical font names to actual font files. Directories that do not contain font databases created by **`bldfamily(1)`** will be ignored. The default font path is `$OPENWINHOME/lib/fonts`.

`-init ‘POSTSCRIPT-code‘` Execute Postscript initialization code.

This option allows the user to specify Postscript code which is to be used to initialize the server. If unspecified, the default value is:

```
(NeWS/init.ps) (r) file cvx exec
```

`-nobanner` Disable banner display.

Disables display of the OpenWindows banner screen at start-up, slightly decreasing the amount of time needed to start the server.

`-banner` Enable banner display.

Enables display of the OpenWindows banner screen at start-up.This is the default.

`-nominexp` Disable "minimized exposure" handling.

This option is used to disable "minimized exposure", which is only used by multi-planegroup devices such as gt, cg12, cg8, and rfx. "Minimized Exposure" means that the server does not send expose events to windows in one planegroup that are exposed by windows in another planegroup.

`-iobuffersize size` Set input/output buffer size.

Requests that the size of the network input/output buffers be set to a specific size. The arguments should be specified in kilobytes. There is no guarantee that the server can configure the network buffers to the requested size.

`-overlay4` Set overlays to 4-bit depth.

Selects the manner in which the overlay planes are configured for the RASTERFLEX-32 and RASTERFLEX-HR framebuffers (See 4-BIT OVERLAY MODE on page 14). This is the default setting.

`-overlay8` Set overlays to 8-bit depth.

Selects the manner in which the overlay planes are configured for the RASTERFLEX-32 and RASTERFLEX-HR framebuffers (See 8-BIT OVERLAY MODE on page 15)

`-nosunview` Disable SunView support.

Disables SunView binary compatibility mode.

`-sunview` Enable SunView support.

Enables SunView binary compatibility mode. The benefits of using this mode is that SunView libraries and kernel driver are no longer required to start OpenWindows and the **`xnews-rfx`** server will run with less overhead. Note: This option will only affect the manner in which **`xnews-rfx`** input device events are received. Even in SunView support mode, it is not possible to run SunView applications on the RASTERFLEX device while running the X11/NeWS server. SunView support is not available under Solaris 2.X.

6.4.2.2. DEVICE OPTIONS

`-dev framebuffer` Set framebuffer display device.

This specifies the framebuffer device which the server should use for display. If the option is not set, the default is `/dev/rfx0`. Subsequent uses of this option indicate multiple display devices (screens) on the same server. After each `-dev` option, any of the following modifiers may be used to change the behavior of the named device.

`grayvis` Select gray default visual.

Tells the **`xnews-rfx`** server to default to using a **`GrayScale`** or **`StaticGray`** visual as the default visual, depending upon the state of the *`staticvis`* device modifier. The default behavior is to use color visual classes on non-monochrome devices.

`staticvis` Select static default visual.

Tells the **`xnews-rfx`** server to default to using a `StaticColor`, `StaticGray`, or `TrueColor` visual as the default visual depending upon the state of the *`grayvis`* and *`defdepth`* modifiers. The default behavior is to use dynamic visual classes.

`defdepth n` Select default visual depth.

Tells the **`xnews-rfx`** server to default to using a visual which is of the given depth as the default visual. The color class of the visual is dependent upon the values of the *`grayvis`* and *`staticvis`* modifiers. The default value (if unspecified) is *8* for the RASTERFLEX-32 and RASTERFLEX-HR framebuffers, and 24 for RASTERFLEX-24 framebuffers.

`left|right|top|bottom` Set screen position.

Indicates where the given screen resides in relation to the previous one given on the command line. The default is to place each subsequent screen to the right of the previous one.

6.4.3. USING MULTIPLE SCREENS

The RASTERFLEX server permits the use of multiple screens/framebuffers controlled by a single invocation of the server. The server automatically locates each RASTERFLEX device which is installed and initializes it as a unique screen. Additionally, the RASTERFLEX server locates and initializes many standard Sun framebuffers. The screens to use are specified by providing multiple `-dev` options to the **`xnews-rfx`** server. For example, to start a dual-screen server using a RASTERFLEX-32 and a Sun BW2 framebuffer, you can use the following command line:

```
xnews-rfx -dev /dev/rfx0 -dev /dev/bwtwo0
```

The following types of devices are supported by the RASTERFLEX X11/NeWS Server:

- RASTERFLEX-24 (`/dev/rfx`*n*)
- RASTERFLEX-32 (`/dev/rfx`*n*)
- RASTERFLEX-HR (`/dev/rfx`*n*)
- Sun CG3 (`/dev/cgthree`*n*)
- Sun CG4 (`/dev/cgfour`*n*)
- Sun GX (`/dev/cgsix`*n*)
- Sun CG8 (`/dev/cgeight`*n*)
- Sun BW2 (`/dev/bwtwo`*n*)

To cause the **`xnews-rfx`** server to use just a single device, even though multiple devices are present, the `-dev` option can be used to select a specific device to run on.

In order to start a client on a specific screen, the **`DISPLAY`** environment can be set using the notation `[node]:[server].[screen]`. For example, to start a client on screen 1 of a locally running X11/NeWS server, set the **`DISPLAY`** variable to "`unix:0.1`".

6.4.4. RASTERFLEX COMPATIBILITY ISSUES

X Window System applications should always be able to work with any compliant X Window System. However, many of the unique display capabilities of the RASTERFLEX-32 and RASTERFLEX-HR cannot be anticipated by many existing applications. This section provides some tips on how to circumvent some common problems with such applications.

6.4.4.1. SELECTING A SET OF VISUALS

The X Window System allows a server to advertise the full range of display capabilities which it is capable of supporting by advertising a set of visuals.

An X visual is a combination of display depth and color class which defines specifically how an application should view the pixel data that is being manipulated.

For example, an 8-bit `PseudoColor` visual specifies that there are 8 bits of significant pixel data and that this data is arbitrarily mapped through a modifiable color Look-Up Table which maps the 8 bits of data to the displayed red, green, and blue values.

A 24-bit `TrueColor` visual specifies that there are 24 bits of significant pixel data, and that the data is further subdivided into three separate channels, one each for red, green, and blue, and that each channel is mapped through a fixed color look-up operation before display.

The RASTERFLEX-32 and RASTERFLEX-HR devices are unique in that they advertise a wide variety of visual classes and depths; whereas, most common framebuffers support just a single depth, but frequently multiple classes. The RASTERFLEX-24 device will also support multiple depths; however, for this device the display depth is selected at server start-up and only windows of a single depth can be created.

All X Window System servers also define a default visual, which is the visual type from which the server root window is derived. The compatibility problem which arises is that many applications simply utilize the default visual without examining the full set of visuals which are supported by the server to determine the one which is most appropriate for that application's purposes. A more sophisticated application might examine the full set of available visuals and decide to create its windows using the one which best meets its needs.

The RASTERFLEX-32/HR framebuffers are fully capable of displaying windows of different visual types on the screen simultaneously; however, the less sophisticated application will only utilize the server default. For example, an image display program may only display data in 8-bit `PseudoColor` mode (if that is the default) even though 24-bit `TrueColor` display capabilities are also available.

To address this issue, the RASTERFLEX server supports command line options in order to configure the **default** behavior of the server so as to match the **assumed** behavior made by an application program.

This approach, unfortunately, has the following drawbacks:

- The default behavior for one application can be completely inappropriate for another application — in fact, it can cause the other program to not operate at all. In the latter case, the user must stop and restart the server in order to use the second application.
- The default behavior can entail large inefficiencies for other applications, lowering their performance or visual appeal. An example of this would be running an application which uses a small number of distinct colors in 24-bit mode.
- The user must explore internal details of each application and experiment with server configurations until it works.

The server defaults to a common, 8-bit `PseudoColor` configuration for the RASTERFLEX-32 and RASTERFLEX-HR. Most recent color applications work in this mode. The default is 24-bit `DirectColor` for the RASTERFLEX-24.

6.4.4.2. SELECTING DEFAULT WINDOW DEPTH

The RASTERFLEX X11/NeWS server supports *both* 8- and 24-bit depth windows on the RASTERFLEX-32 and RASTERFLEX-HR devices. For the RASTERFLEX-24 device, it will allow selection of either 8-bit or 24-bit depth for all windows at server start-up.

You may set the *default* depth for the server (the depth of the background pattern and the depth of windows created for applications that do not explicitly select a depth) with the *`defdepth`* device modifier as follows:

```
xnews-rfx -dev /dev/rfx0 defdepth 8
```

or

```
xnews-rfx -dev /dev/rfx0 defdepth 24
```

NOTE
By specifying the default depth, you have not precluded the use of windows using other depths. You have only set the depth that a window will have if the application *does not* specify it. Many simple applications use the server default depth.

6.4.4.3. SELECTING DEFAULT COLOR CLASS

Each window supports one of a number of different color schemes or classes. These classes are the standard X Window color classes. The RASTERFLEX environment supports color classes on a window-by-window basis. This provides the following advantages:

- Applications that require a specific color class will work with the RASTERFLEX hardware.
- Fewer server resources must be shared between applications, which helps reduce the colormap flashing that sometimes occurs when switching between applications.
- Applications can choose window depth and color class on a per-window basis rather than being forced to stick with the server default at all times.

Once again, you can choose the server default color class through combinations of the *`defdepth`*, *`staticvis`*, and *`grayvis`* device modifiers. The following table shows how the default visual is selected based upon the value for these parameters:

Table 6.1. Default Color Classes

defdepth	staticvis	grayvis	Selected Visual
5 or 8	No	No	5/8-bit PseudoColor
5 or 8	No	Yes	5/8-bit GrayScale
5 or 8	Yes	No	5/8-bit StaticColor
5 or 8	Yes	Yes	5/8-bit StaticGray
24	No	Ignored	24-bit DirectColor
24	Yes	Ignored	24-bit TrueColor

A value of "*`No`*" for *`staticvis`* and *`grayvis`* modifiers means that the option is not specified on the command line. A value of "*`Ignored`*" indicates that the modifier has no effect upon visual selection. The *`defdepth`* parameter defaults to *`8`* if unspecified.

6.4.5. USING THE VISUAL SELECTION EXTENSION

The RASTERFLEX X11/NeWS Server supports a special Connectware-developed extension called the Visual Selection extension which will allow clients which normally utilize the default visual of the server to be run using any of the available server visual types. The Visual Selection Extension allows a user to specify that one of the available visuals be advertised to a client as if the selected visual were actually the server default. For example, a user could start the server with the default visual type set to 8-bit `PseudoColor` so that all standard clients such as terminal windows and other utilities run in this manner, then use the Visual Selection extension to cause a raster display application to be run as if the server default was set to 24-bit `TrueColor`.

The Visual Selection Extension is accessed via a special client called **`vset`** which allows the user to specify the visual which will be advertised to the next client program (after **`vset`**) which connects to the RASTERFLEX server. The parameters to **`vset`** are:

```
vset <visual id>
        or
vset <visual class> [visual depth]
```

where <visualid> refers to the specific id of the visual to be advertised as the server default or <visual class> is one of the valid X visual classes (`PseudoColor`, `TrueColor`, `StaticGray`, etc.) and <visual depth> is a numeric value representing the desired visual depth. For example, to start an application named **`appl`** so that it believes the server default visual type is 24-bit `TrueColor,` the following sequence would be used:

```
vset TrueColor 24
appl
```

Immediately after the **appl** client is started, the default visual type advertised by the server will revert to the true server default (as specified at start-up via the command line options described above).

The Visual Selection Extension does not modify the server default visual, nor does it change any of the attributes of the root window (which always will be of the true server default visual class/depth). For this reason, clients which attempt to perform operations directly upon the root window with an altered notion of the default visual will not work properly. Applications which currently fall into this category include window managers (**`twm, mwm, olwm`**) and screen dump utilities (**`xmag`**, **`xwd`**).

NOTE
The Visual Selection Extension is only a workaround for existing applications which do not properly select the desired visual type or allow explicit visual selection by the end-user. Some applications (as noted above) may not operate properly when using an altered notion of the default visual via the Visual Selection extension.

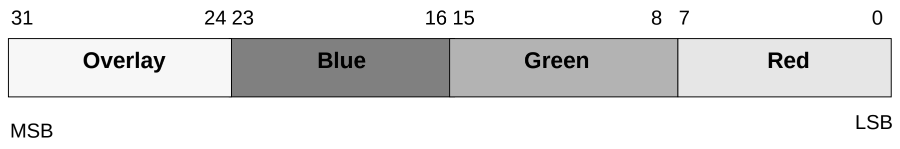
6.4.6. OVERLAY MODE SELECTION

The RASTERFLEX X11/NeWS server is also different from most common framebuffers in that it supports two distinct layers of display hierarchy — a set of overlay planes and a set of underlay planes.

NOTE
For most existing applications, the existence of overlay planes is of no concern and the application will run without any problems. However, the manner in which the overlay planes are configured (see below) can have an impact on the manner in which colormaps and other resources are utilized, so users may benefit from an understanding of these issues. For application developers who do want to take advantage of the overlay capabilities of the RASTERFLEX hardware, a programmatic example is provided in a subsequent section.

The framebuffer within the RASTERFLEX-32 and RASTERFLEX-HR devices actually contains 32 bits of data for each displayed pixel. The most significant 8 bits of the 32 bits of pixel data constitute the overlay plane, while the remaining 24 bits are used as the underlay plane to display either 8- or 24-bit data. Windows can be created in either the underlay planes or the overlay planes, but never both. The decision about which planes a window resides in is strictly a function of the visual type with which the window has been created (more on this later).

Figure 6.1. Hardware pixel format.



Overlay windows are regular windows with the added feature of transparency on a pixel-by-pixel basis. The term "overlay" is used since generally one wishes the transparent window to be on top of some other window, referred to as the "underlay". Drawing into the overlay window does not destroy data in the underlay.

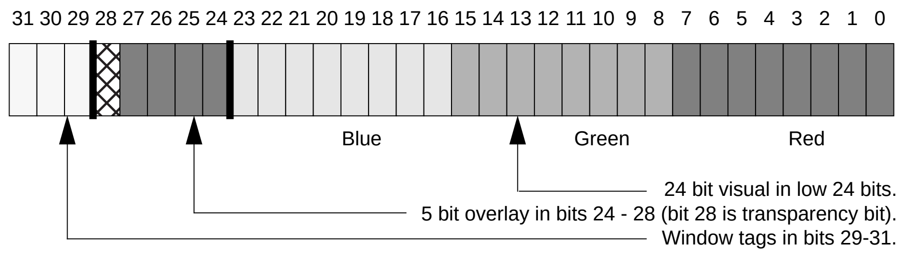
The 8-bit overlay within the RASTERFLEX-32/HR can be configured in one of two manners: 4-bit Overlay Mode or 8-bit Overlay Mode. The mode used is selected at server start-up via the `-overlay4` and `-overlay8` command line options to **`xnews-rfx`**. If neither option is specified, the default behavior is to advertise 4-bit overlay capabilities.

6.4.6.1. 4-BIT OVERLAY MODE

In 4-bit Overlay Mode, the 8-bit overlay pixel is actually broken into three separate components:

- a 4-bit color value (hence the name),
- a 1-bit transparency control bit, and
- 3 bits of control information.

Figure 6.2. Hardware pixel format — 4-bit overlay model.


31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Blue
Green
Red
24 bit visual in low 24 bits.
5 bit overlay in bits 24 - 28 (bit 28 is transparency bit).
Window tags in bits 29-31.

The four bits of color value in the 4 least significant bits of the overlay allow 16 distinct overlay colors to be available. The transparency control bit allows the visibility of the overlay pixel to be controlled on a pixel-by-pixel basis.

If this bit is set, the overlay pixel is transparent, and the contents of the underlay planes beneath the window become visible. If the bit is clear (initial state), the pixel is considered opaque, and the displayed color is selected, based upon the mapping of the 4 bits of color data through the currently installed Look-Up Table for the overlay planes (see **USING MULTIPLE LOOK-UP TABLES** below).

> **NOTE**
> The transparency of a given pixel can be changed without making any changes to the four bits of color data by simply toggling the value of the transparency bit. This enables overlay data to be non-destructively turned on and off.

In 4-bit Overlay Mode, the data within the underlay planes can be either 8-bit or 24-bit. The remaining three control bits are used to determine the format of the underlay data based upon its visual type. The values within the control bits are maintained entirely by the **`xnews-rfx`** server and are not accessible to application programs.

6.4.6.2. 8-BIT OVERLAY MODE

In 8-bit Overlay Mode, all 8 bits of the overlay pixel are used in determining the displayed value for the overlay planes. Transparency can still be selected on a pixel-by-pixel basis in this mode by storing a value of zero (0) within the overlay planes. Two overlay colors also are reserved for cursor display, leaving a total of 253 colors available for application use.

When in 8-bit Overlay Mode, only 24-bit data can be stored within the underlay planes. This is necessary due to the fact that no control bits are left over for selection of multiple underlay display formats. Additionally, 8-bit Overlay Mode has implications on the color Look-Up Table allocation (see below).

6.4.7. MULTIPLE LOOK-UP TABLE MANAGEMENT

The RASTERFLEX-32 and RASTERFLEX-HR framebuffers are different from most common framebuffers in that they also provide multiple hardware color Look-Up Tables; most framebuffers only have one.

The RASTERFLEX framebuffers actually support two full 256-entry color Look-Up Tables plus a 16-entry Look-Up Table. Additionally, they have the capability to display `TrueColor` or `StaticGray` data without using any Look-Up Table space.

The following paragraphs describe the manner in which these Look-Up Tables are managed by the **`xnews-rfx`** server.

NOTE
The management of hardware color Look-Up Tables places no limitations on the number of X Colormaps which can be created. Its only impact is the manner in which X Colormaps are downloaded to the hardware as a result of colormap installation requests from a client application or window manager.

The manner in which the overlay planes have been configured at server start-up has a major implication on the manner in which colormaps are managed.

If the server is started in 8-bit Overlay Mode, the colormap handling is relatively fixed. One 256-entry color palette is reserved for the 8-bit overlay planes, and the other 256-entry palette is used for the 24-bit underlay planes. The 16-entry Look-Up Table is not utilized.

In 8-bit Overlay Mode, the hardware color Look-Up Tables always contain the most recently installed X colormap which is associated with a visual of the proper type. One Look-Up Table contains the most recently installed overlay visual colormap, and the other always contains the most recently installed underlay visual colormap.

Hardware Look-Up Table management is more complex when using 4-bit overlay mode, and also significantly more flexible.

In 4-bit Overlay Mode, one of the 256-entry color Look-Up Tables is reserved for the default colormap which is advertised to all clients. This colormap always remains installed, meaning that applications which use it are never subjected to colormap flashing. The intention is that applications which use only a few colors, such as the standard X clients, are always displayed with their proper colors.

The other 256-entry colormap is available for use by applications which create and install (or request installation via a window manager) their own colormaps. This is common for many raster display programs which require a large number of colors to operate. This hardware color Look-Up Table always contains the most recently installed underlay visual colormap (other than the default). The 16-entry Look-Up Table is used for the most recently installed colormap associated with an overlay visual.

As an additional bonus, X colormaps which are associated with either the 24-bit `TrueColor` or 8-bit `StaticGray` visual types are displayed using a special "pass-through" mode which actually uses no space within the hardware Look-Up Tables. For these visual types, the actual data within the framebuffer is passed around the color Look-Up Tables and directly drives the displayed red, green, and blue values.

This effectively means that windows using the default colormap, an application-defined underlay colormap, an application-defined overlay colormap, and any using 24-bit `TrueColor` or 8-bit `StaticGray`, can all be displayed simultaneously in 4-bit overlay mode with no visual anomalies due to colormap flashing.

7. RASTERFLEX LOADABLE DDX ENVIRONMENT

The RASTERFLEX Loadable DDX (Device-Dependent X) Environment provides a dynamically loadable object module which supports the OpenWindows Window System on the RASTERFLEX Accelerators under the Solaris 2.3 and subsequent releases. Starting with the OpenWindows 3.3 environment found under Solaris 2.3, SunSoft created an interface which allows the device-dependent software required to support the X Window System on a display device to be packaged within a dynamically loadable object module which is loaded into the standard OpenWindows X Window System Server (**`Xsun`**) at run-time. The RASTERFLEX Loadable DDX Environment uses this capability to seamlessly integrate the RASTERFLEX-24, RASTERFLEX-32, and RASTERFLEX-HR hardware into the SunSoft OpenWindows environment. This section provides an overview of the components provided within the RASTERFLEX Loadable DDX Environment as well as specific information on how to use the unique features of the RASTERFLEX hardware within the OpenWindows environment.

7.1. LOADABLE DDX SOFTWARE COMPONENTS

This section describes the various software components which are provided with the Loadable DDX Environment for the RASTERFLEX framebuffers. Under Solaris 2, the default location for RASTERFLEX software is `/opt/VITrflex`. Unless otherwise specified during software installation, all software components will reside beneath this directory.

7.1.1. SOFTWARE RELEASE BUTLER

The Software Release Butler is a utility which prompts you with questions regarding the installation, then performs the installation for you. The questions it asks involve such things as which portions of the software you wish to install and where you wish to install them in your file system. Refer to the Software Release Notes provided with your RASTERFLEX software for information on how to run the Software Release Butler.

7.1.2. SOFTWARE CONFIGURATION MECHANIC

The Software Configuration Mechanic is a utility which will examine the manner in which you have installed the RASTERFLEX software release on your system, ask you a few basic configuration questions, and then generate a file containing a set of environment variable assignments suitable for inclusion in a user `.cshrc` file. The Software Configuration Mechanic is a shell script named `rfxconfig.sh` which by default is stored in `/opt/VITrflex` under Solaris 2.X. Refer to the Software Release notes for more specific information on running the Software Configuration Mechanic.

7.1.3. DEVICE DRIVER

The device driver allows the operating system to communicate with the RASTERFLEX hardware. It is **required** in order to run the remainder of the RASTERFLEX software. Refer to the Software Release notes for more specific information on how to install the device driver.

7.1.4. LOADABLE DDX MODULE

The Loadable DDX Module is **required** to run X on the RASTERFLEX hardware under OpenWindows 3.3 and subsequent releases. Connectware has used the hooks for loadable DDX support provided by SunSoft to integrate the unique display and acceleration capabilities of the RASTERFLEX Accelerators into the OpenWindows environment. This module contains only the object code required to support the X Window System on the RASTERFLEX devices. Support for other standard Sun devices is included as part of the standard OpenWindows release or other supplemental packages.

7.2. SETTING UP THE USER ENVIRONMENT

This section outlines the changes which may be required to the user environment after loading the RASTERFLEX software. It is preferable to add the following commands to the system `.cshrc` file or other user start-up files so that these changes apply to all users.

Note that the RASTERFLEX Software Configuration Mechanic, provided as a part of the RASTERFLEX software release, performs basic environment setup automatically. This section is provided for advanced users who wish to customize their environment. The Configuration Mechanic is (by default) installed in `/opt/VITrflex/rfxconfig.sh` under Solaris 2.X.

7.2.1. LOCATING SERVER RESOURCES

The Sun OpenWindows server must be able to locate certain resources in order to execute properly. All resources required by the server are located relative to the top of the standard OpenWindows release tree, which is identified by the **`OPENWINHOME`** environment variable. For example, if you have loaded OpenWindows under the directory `/usr/openwin` [the default], then you would use the following C Shell command to set the **`OPENWINHOME`** variable:

```
    setenv OPENWINHOME /usr/openwin
```

7.2.2. LOCATING SHARED LIBRARIES

When running the OpenWindows server or clients, it is necessary to add the X11, toolkit and server support shared libraries, which are supplied with the standard OpenWindows release, to the library search path. To do this, set the **`LD_LIBRARY_PATH`** environment variable to point to the location where the libraries have been loaded. For example, the following command could be used:

```
  setenv LD_LIBRARY_PATH $OPENWINHOME/lib:$LD_LIBRARY_PATH
```

7.2.3. SETTING THE PATH VARIABLE

The location of the X Windows System clients and utilities should be added to the execution search path for all users who wish to use them. For a default installation, use the following commands to add the appropriate directory to the **`PATH`** environment variable:

```
setenv PATH $OPENWINHOME/bin:$PATH
rehash
```

7.2.4. LOCATING MANUAL PAGES

If the OpenWindows manual pages have been loaded in a location other than `/usr/man`, it is necessary to add the location of these pages to the **`MANPATH`** environment variable in order for them to be located properly. The following command could be used for this purpose:

```
setenv MANPATH $OPENWINHOME/man:$MANPATH
```

After this, the **`man`** command is able to locate the OpenWindows manual pages.

7.3. THE RASTERFLEX LOADABLE DDX MODULE

The following sections describe the features of the OpenWindows Loadable DDX Module for the RASTERFLEX hardware. The RASTERFLEX software release includes an object module that supports the SunSoft's OpenWindows implementation of the X Window System on your SPARCstation. This object module provides full support for the unique hardware acceleration and display capabilities of the RASTERFLEX frame buffers.

The RASTERFLEX Loadable DDX Module uses the unique acceleration and display features of the RASTERFLEX hardware transparently. As a result, the application can take advantage of these features without having to be modified. An application can simply use standard Xlib graphics primitives, and these operations are automatically accelerated by the hardware, if applicable.

Other capabilities, such as the use of overlays and shared memory image and pixmap operations, require explicit selection by the application. Examples of using these capabilities within a simple X program are provided in the section on **RASTERFLEX ADVANCED FEATURES**.

The RASTERFLEX Loadable DDX Module supports all three types of RASTERFLEX frame buffers: the RASTERFLEX-24, the RASTERFLEX-32, and the RASTERFLEX-HR.

7.4. INVOKING THE OPENWINDOWS X SERVER

This section describes how to invoke the OpenWindows **`Xsun`** server on the RASTERFLEX hardware. It includes certain requirements that must be satisfied, as well as several methods of invoking the **`Xsun`** server on the host system, including

- using the Sun `openwin` start-up script,
- using the X initializer (**`xinit`**), and

- using the X display manager (**`xdm`**).

7.4.1. REQUIRED CONDITIONS

Before attempting to invoke the OpenWindows X Window System Server, the following conditions must be satisfied:

- You must already have installed the OpenWindows 3.3 or later software release from SunSoft onto your system. Additionally, you should set the **`OPENWINHOME`** environment variable to point to the top of the OpenWindows release tree.
- You should have the location where the OpenWindows executables are stored (`$OPENWINHOME/bin` by default) in your search path.
- You should have the location where the OpenWindows shared libraries are stored (`$OPENWINHOME/lib` by default) in your library load path.

7.4.1.1. USING OPENWIN - THE OPENWINDOWS START-UP SCRIPT

The **`openwin`** script, which is provided as part of the OpenWindows software release from Sun, can be used to invoke the Xsun server on the RASTERFLEX device. This is done by providing the *`-dev`* command line option to indicate that the X Window System should be run on the RASTERFLEX frame buffer device. For example, the following sequence could be used:

```
openwin -dev /dev/rfx0
```

If any additional command line options are to be passed to the **`Xsun`** server, they could be included before or after the *`-dev`* option.

7.4.1.2. USING XINIT - THE X SYSTEM INITIALIZER

The X Window System Initializer (**`xinit`**) allows you to customize your environment for invoking the **`Xsun`** server and to bring up several initial X clients.

The **`xinit`** command program starts the X Window System server and a first client program (usually a terminal emulator or window manager). When the first client program exits, **`xinit`** kills the server and then terminates.

7.4.1.3. COMMAND FORMAT

The basic format of the **`xinit`** command is:

```
xinit [[client]options][--[server] [display] options]
```

7.4.1.4. EXAMPLE COMMAND LINE

A simple example of using **`xinit`** is illustrated in the following command line:

```
xinit xterm -- /usr/openwin/bin/Xsun -dev /dev/rfx0
```

This command line starts up the OpenWindows **`Xsun`** server executable using the RASTERFLEX device (`/dev/rfx0`), then brings up the **`xterm`** terminal emulator client. Then you can bring up other client applications using the terminal emulator. When finished using the window system, you can exit from the emulator window by logging out, and the server is shut down also, since **`xinit`** assumes that the user session has ended when control is returned from the client program or the `.xinitrc` script (see below).

If no specific client is specified on the command line, **`xinit`** looks for a file, called `.xinitrc`, in the user's home directory. This file usually is a shell script containing a series of command and client programs to execute at window system start-up.

7.4.1.5. EXAMPLE .xinitrc FILE

The following is an example `.xinitrc` file:

```
#! /bin/csh
#! Sample client initialization script
xclock &
xterm &
olwm
```

This script starts up the X clock program, a terminal emulator window, and the OpenLook Window Manager, starting all commands — except the final one — in background mode. As a result, the shell script is not exited from until the **`olwm`** program stops executing. When you exit from the window manager (using an *`Exit`* option from a root menu), the window system shuts down automatically.

Always place the client from which you exit the window system — usually a terminal emulator window or window manager — as the final command in your **`xinit`** client initialization file and ensure that it is not started as a background job.

If you do not specify a client program on the command line and a `.xinitrc` file does not exist, then **`xinit`** uses the default command:

```
xterm -geometry +1+1 -n login -display :0
```

If you do not specify a server program on the command line, then **`xinit`** looks for a file, called `.xserverrc`, in the user's home directory. It runs this file as a shell script to start up the server.

7.4.1.6. EXAMPLE SERVER INITIALIZATION FILE

An example server initialization file for the **`Xsun`** server follows:

```
#! /bin/csh
#! Server initialization shell script
/usr/openwin/bin/Xsun -dev /dev/rfx0
```

This simple script starts the Sun OpenWindows X11R5 server.

If you do not specify a server command on the command line and an `.xserverrc` file does not exist, then **`xinit`** uses the following command to start the X server:

```
X :0 -dev /dev/fb
```

The X Window System initializer, **`xinit`**, provides a simple, yet flexible, means for invoking the window system server and a set of client applications.

7.4.1.7. USING XDM - THE X DISPLAY MANAGER

The X Display Manager (**`xdm`**) program manages a collection of X displays. It is designed to provide services similar to those provided by **`init`**, **`getty`**, and **`login`** on character terminals:

- Prompting for login/password,
- Authenticating the user, and
- Running a 'session'.

The X Display Manager starts up the X server, which is under its control, and displays a login / password prompt. No other client application can connect to the server when the login prompt is displayed.

After you successfully login, the server initiates a session, which can be a terminal emulator, a window manager, or some application-specific program that the user is running. Once the session is over, and the user exits from the terminal emulator, window manager, or program, the X display manager resets the X server and redisplays the login / password prompt.

For complete information and options on configuring the X Display Manager, refer to the **`xdm`** manual pages that are supplied with the OpenWindows software release.

7.4.2. USING MULTIPLE SCREENS

The OpenWindows **`Xsun`** server permits the use of multiple screens/frame buffers controlled by a single invocation of the server. The *`-dev`* command line switch is used to specify the display devices to use. Multiple *`-dev`* options may be used to enable a multi-screen environment. For example, if you wanted to run the X Window System on both a Sun GX (/dev/cgsix0) device and a RASTERFLEX device, the following options could be used:

```
-dev /dev/rfx0 -dev /dev/cgsix0
```

In order to start a client on a specific screen, the **`DISPLAY`** environment can be set using the notation `[node]:[server].[screen]`. For example, to start a client on screen 1 of a locally running X server, the **`DISPLAY`** variable could be set to "`unix:0.1`".

7.4.3. RASTERFLEX COMPATIBILITY ISSUES

X Window System applications should always be able to work with any compliant X Window System Server. However, many of the unique display capabilities of the RASTERFLEX-24, RASTERFLEX-32, and RASTERFLEX-HR hardware cannot be anticipated by many existing applications. This section provides some tips on how to circumvent some common problems with such applications.

7.4.3.1. VISUAL SELECTION

The X Window System allows a server to advertise the full range of display capabilities which it is capable of supporting by advertising a set of visuals.

An X visual is a combination of display depth and color class which define specifically how an application should view the pixel data which is being manipulated.

For example, an 8-bit `PseudoColor` visual specifies that there are 8 bits of significant pixel data and that this data is arbitrarily mapped through a modifiable color Look-Up Table which maps the 8 bits of data to the displayed red, green, and blue values.

A 24-bit `TrueColor` visual specifies that there are 24 bits of significant pixel data, and that the data is further subdivided into three separate channels, one each for red, green, and blue, and that each channel is mapped through a fixed color lookup operation before display.

The RASTERFLEX-32 and RASTERFLEX-HR devices are unique in that they advertise a wide variety of visual classes and depths; whereas most common framebuffers support just a single depth, but usually multiple classes. The RASTERFLEX-24 device will also support multiple depths; however, for this device the display depth is selected at server start-up and only windows of a single depth can be created.

All X Window System servers also define a default visual, which is the visual type from which the server root window is derived. The compatibility problem which arises is that many applications simply utilize the default visual without examining the full set of visuals which are supported by the server to determine the one which is most appropriate for that application's purposes. A more sophisticated application might examine the full set of available visuals and decide to create its windows using the one which best meets its needs.

The RASTERFLEX-32/HR frame buffers are fully capable of displaying windows of different visual types on the screen simultaneously; however, the less sophisticated application will often only utilize the server default. For example, an image display program may only display data in 8-bit `PseudoColor` mode (if that is the default) even though 24-bit `TrueColor` display capabilities are also available.

To address this issue, the RASTERFLEX server supports command line options in order to configure the **default** behavior of the server so as to match the **assumed** behavior made by an application program.

This approach, unfortunately, has the following drawbacks:

- The default behavior for one application can be completely inappropriate for another application — in fact it can cause the other program to not operate at all. In the latter case, the user must stop and restart the server in order to use the second application.
- The default behavior can entail large inefficiencies for other applications, lowering their performance or visual appeal. An example of this would be

running an application which uses a small number of distinct colors in 24-bit mode.

- The user must explore internal details of each application and experiment with server configurations until it works.

The server defaults to a common, 8-bit `PseudoColor` configuration for RASTERFLEX-32 and RASTERFLEX-HR framebuffers. Most recent color applications work in this mode. The default is 24-bit `TrueColor` for the RASTERFLEX-24.

7.4.3.2. SELECTING DEFAULT WINDOW DEPTH

The RASTERFLEX Loadable DDX Module supports ***both*** 8-bit and 24-bit depth windows on the same display for the RASTERFLEX-32 and RASTERFLEX-HR devices. For the RASTERFLEX-24 device, it will allow selection of either 8-bit or 24-bit depth for all windows at server start-up.

You may set the *default* depth for the server (the depth of the background pattern and the depth of windows created for applications that do not explicitly select a depth) with the `defdepth` option as follows:

```
-dev /dev/rfx0 defdepth 8
```
[Default for RASTERFLEX-32/HR]

or

```
-dev /dev/rfx0 defdepth 24
```
[Default for RASTERFLEX-24]

NOTE
The defdepth option is a device modifier option which may only be specified after a -dev option selecting a particular device. The selection of the default depth will apply only to this particular device.

7.4.3.3. SELECTING DEFAULT COLOR CLASS

Each window supports one of a number of different color schemes or classes. These classes are the standard X Window color classes. The RASTERFLEX software supports color classes on a window-by-window basis. This provides the following advantages:

- Applications that require a specific color class will work with the RASTERFLEX hardware.
- Fewer server resources must be shared between applications, which helps reduce the colormap flashing that sometimes occurs when switching between applications.
- Applications can choose window depth and color class on a per-window basis rather than being forced to stick with the server default at all times.

You may set the *default* class for the server (the class of the root window and the class of windows created for applications that do not explicitly select a class) with the `defclass` option as follows:

```
-dev /dev/rfx0 defclass PseudoColor
```

NOTE
The defclass option is a device modifier option which may only be specified after a -dev option selecting a particular device. The selection of the default class will apply only to this particular device.

The following are the valid color class selections, along with a brief description of each class:

`defclass PseudoColor` [Default for defdepth 8]
PseudoColor windows use a color Look-Up Table to convert pixel values in memory to color intensities on the screen. The RASTERFLEX-32 card supports PseudoColor for 8- and 4-bit depth color maps.

`defclass StaticColor`
Similar to PseudoColor above, but the color Look-Up Table cannot be modified. Instead, a selection of 256 colors spanning the RGB space is pre-loaded into the Look-Up Table.

`defclass StaticGray`
The pixel value is interpreted directly as an intensity without going through a color Look-Up Table. The intensity is applied equally to the Red, Green and Blue components, creating shades of gray. StaticGray is supported for 8-bit color maps only.

`defclass GrayScale`
This is similar to PseudoColor, except there is only one intensity (applied equally to Red, Green and Blue as in StaticGray) per entry in the color Look-Up Table. 8-bit color maps only.

`defclass DirectColor`
This method is supported for 24-bit windows only. It is like the PseudoColor class for 8-bit windows, but each of the Red, Green and Blue components is separately indexed from a part of the 24-bit pixel.

`defdepth TrueColor` [Default value for defdepth 24]
In this method, like StaticGray above, there is no Look-Up Table. But unlike StaticGray, a different intensity is used for each component. The value is taken from three different parts of the 24-bit pixel.

7.4.4. USING THE VISUAL SELECTION EXTENSION

The RASTERFLEX Loadable DDX Module supports a special Connectware extension called the Visual Selection Extension which will allow clients which normally utilize the default visual of the server to be run using any of the available server visual types. The Visual Selection Extension allows a user to specify that one of the available visuals be advertised to a client as if the selected visual were actually the server default. For example, a user could start the server with the default visual type set to 8-bit `PseudoColor` so that all standard clients such as terminal windows and other utilities run in this manner, then use the Visual Selection extension to cause a raster display application to be run as if the server default was set to 24-bit `TrueColor`.

The Visual Selection Extension is accessed via a special client called **`vset`** which allows the user to specify the visual which will be advertised to the next client program (after **`vset`**) which connects to the RASTERFLEX server. The parameters to **`vset`** are:

```
vset <visual id>
        or
vset <visual class> [visual depth]
```

where <visualid> refers to the specific id of the visual to be advertised as the server default or <visual class> is one of the valid X visual classes (`PseudoColor`, `TrueColor`, `StaticGray`, etc.) and <visual depth> is a numeric value representing the desired visual depth. For example, to start an application named **`appl`** so that it believes the server default visual type is 24-bit `TrueColor,` the following sequence would be used:

```
vset TrueColor 24
appl
```

Immediately after the **appl** client is started, the default visual type advertised by the server will revert to the true server default (as specified at start-up via the command line options described above).

The Visual Selection Extension does not modify the server default visual, nor does it change any of the attributes of the root window (which always will be of the true server default visual class/depth). For this reason, clients which attempt to perform operations directly upon the root window with an altered notion of the default visual will not work properly. Applications which currently fall into this category include window managers (**`twm, mwm, olwm`**) and screen dump utilities (**`xmag`**, **`xwd`**).

NOTE
The Visual Selection Extension is only a workaround for existing applications which do not properly select the desired visual type or allow explicit visual selection by the end-user. Some applications (as noted above) may not operate properly when using an altered notion of the default visual via the Visual Selection extension.

7.4.5. OVERLAY MODE SELECTION

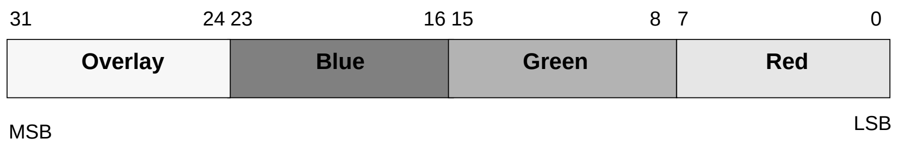
The RASTERFLEX-32 and RASTERFLEX-HR devices are also different from most common frame buffers in that they supports two distinct layers of display hierarchy — a set of overlay planes and a set of underlay planes.

NOTE
For most existing applications, the existence of overlay planes is of no concern and the application will run without any problems. However, the manner in which the overlay planes are configured (see below) can have an impact on the manner in which colormaps and other resources are utilized, so users may benefit from an understanding of these issues. For application developers who do want to take advantage of the overlay

capabilities of the RASTERFLEX hardware, a programmatic example is provided in a subsequent section.

The frame buffer within the RASTERFLEX-32 and RASTERFLEX-HR devices actually contains 32 bits of data for each displayed pixel. The most significant 8 bits of the 32 bits of pixel data constitute the overlay plane while the remaining 24 bits are used as the underlay plane to display either 8 or 24-bit data. Windows can be created in either the underlay planes or the overlay planes, both never both. The decision about which planes a window resides in is strictly a function of the visual type with which the window has been created (more on this later).

Figure 7.1. Hardware pixel format.



Overlay windows are regular windows with the added feature of transparency on a pixel-by-pixel basis. The term “overlay” is used since generally one wishes the transparent window to be on top of some other window, referred to as the “underlay”. Drawing into the overlay window does not destroy data in the underlay.

The 8-bit overlay within the RASTERFLEX-32/HR can be configured in one of two manners: 4-bit Overlay Mode or 8-bit Overlay Mode. The mode used is selected by setting the RFX_OVERLAY_MODE environment variable before running **Xsun**. If the environment variable is not set, the default behavior is to advertise 4-bit overlay capabilities. For example, to select 8-bit overlay mode, the following command could be used:

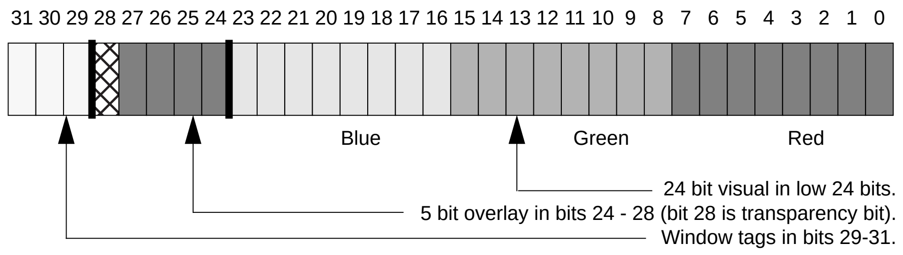
```
    setenv RFX_OVERLAY_MODE 8
```

7.4.5.1. 4-BIT OVERLAY MODE

In 4-bit Overlay Mode, the 8-bit overlay pixel is actually broken into three separate components:

- a 4-bit color value (hence the name),
- a 1-bit transparency control bit, and
- 3 bits of control information.

Figure 7.2. Hardware pixel format — 4-bit overlay model.


31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Blue
Green
Red
24 bit visual in low 24 bits.
5 bit overlay in bits 24 - 28 (bit 28 is transparency bit).
Window tags in bits 29-31.

The four bits of color value in the 4 least significant bits of the overlay allow 16 distinct overlay colors to be available. The transparency control bit allows the visibility of the overlay pixel to be controlled on a pixel-by-pixel basis.

If this bit is set, the overlay pixel is transparent, and the contents of the underlay planes beneath the window become visible. If the bit is clear (initial state), the pixel is considered opaque, and the displayed color is selected, based upon the mapping of the 4 bits of color data through the currently installed Look-Up Table for the overlay planes (see Multiple Look-Up Table Usage below).

NOTE
The transparency of a given pixel can be changed without making any changes to the four bits of color data by simply toggling the value of the transparency bit. This enables overlay data to be non-destructively turned on and off.

In 4-bit Overlay Mode, the data within the underlay planes can be either 8-bit or 24-bit. The remaining three control bits are used to determine the format of the underlay data based upon its visual type. The values within the control bits are maintained entirely by the RASTERFLEX Loadable DDX Module and are not accessible to application programs.

7.4.5.2. 8-BIT OVERLAY MODE

In 8-bit Overlay Mode, all 8 bits of the overlay pixel are used in determining the displayed value for the overlay planes. Transparency can still be selected on a pixel-by-pixel basis in this mode by storing a value of zero (0) within the overlay planes. Two overlay colors also are reserved for cursor display, leaving a total of 253 colors available for application usage.

When in 8-bit Overlay Mode, only 24-bit data can be stored within the underlay planes. This is necessary due to the fact that no control bits are left over for selection of multiple underlay display formats. Additionally, 8-bit Overlay Mode has implications on the color Look-Up Table allocation (see below).

7.4.6. MULTIPLE LOOK-UP TABLE MANAGEMENT

The RASTERFLEX-32 and RASTERFLEX-HR framebuffers are different from most common framebuffers in that they also provide multiple hardware color Look-Up Tables; most framebuffers only have one.

These RASTERFLEX frame buffers actually support two full 256-entry color Look-Up Tables plus a 16-entry Look-Up Table. Additionally, they have the capability to display 24-bit `TrueColor` or 8-bit `StaticGray` data without using any Look-Up Table space.

The following paragraphs describe the manner in which these Look-Up Tables are managed by the RASTERFLEX devices.

NOTE
The management of hardware color Look-Up Tables places no limitations on the number of X Colormaps which can be created. Its only impact is the manner in which X Colormaps are downloaded to the hardware as a result of colormap installation requests from a client application or window manager.

The manner in which the overlay planes have been configured at server start-up has a major implication on the manner in which colormaps are managed.

If the server is started in 8-bit Overlay Mode, the colormap handling is relatively fixed. One 256-entry color palette is reserved for the 8-bit overlay planes, and the other 256-entry palette is used for the 24-bit underlay planes. The 16-entry Look-Up Table is not utilized.

In 8-bit Overlay Mode, the hardware color Look-Up Tables always contain the most recently installed X colormap which is associated with a visual of the proper type. One Look-Up Table contains the most recently installed overlay visual colormap, and the other always contains the most recently installed underlay visual colormap.

Hardware Look-Up Table management is more complex when using 4-bit overlay mode, and also significantly more flexible.

In 4-bit Overlay Mode, one of the 256-entry color Look-Up Tables is reserved for the default colormap which is advertised to all clients. This colormap always remains installed, meaning that applications which use it are never subjected to colormap flashing. The intention is that applications which use only a few colors, such as the standard X clients, are always displayed with their proper colors.

The other 256-entry colormap is available for use by applications which create and install (or request installation via a window manager) their own colormaps. This is common for many raster display programs which require a large number of colors to operate. This hardware color Look-Up Table always contains the most recently installed underlay visual colormap (other than the default). The 16-entry Look-Up Table is used for the most recently installed colormap associated with an overlay visual.

As an additional bonus, X colormaps which are associated with either the 24-bit `TrueColor` or 8-bit `StaticGray` visual types are displayed using a special “pass-through" mode which actually uses no space within the hardware Look-Up Tables. For these visual types, the actual data within the framebuffer is passed around the color Look-Up Tables and directly drives the displayed red, green, and blue values.

This effectively means that windows using the default colormap, an application-defined underlay colormap, an application-defined overlay colormap, and any using 24-bit `TrueColor` or 8-bit `StaticGray` can all be displayed simultaneously in 4-bit overlay mode with no visual anomalies due to colormap flashing.

8. HARDWARE OVERVIEW

This section provides an overview of the RASTERFLEX accelerator cards, including a list of features and descriptions of the functionality of each of the RASTERFLEX cards.

8.1. RASTERFLEX CAPABILITIES

The RASTERFLEX series accelerators are designed for systems that conform to the SBus specification for SBus add-in cards. This includes all SBus-based Sun SPARCstations, as well as many SPARCstation-compatible machines.

RASTERFLEX cards and software support applications running under the X Window System or OpenWindows. RASTERFLEX cards accelerate general windowing operations as well as some raster primitives relative to dumb framebuffer performance levels. All RASTERFLEX cards are based on a chip developed by Connectware specifically for these products. This chip provides the capability for accelerating raster primitives such as rectangle fills, window copies, and framebuffer input.

The RASTERFLEX-32 and RASTERFLEX-HR enable display and manipulation of true color (24-bit per pixel) images, as well as graphic overlays and multiple windows of different pixel depths. For example, you can run a 24-bit desktop publishing application in one window and an 8-bit spreadsheet in another window on the same screen. Both 24-bit and 8-bit operations are accelerated by the RASTERFLEX hardware. The RASTERFLEX-32 and RASTERFLEX-HR each occupy a single SBus slot, leaving room for other SBus peripherals.

The RASTERFLEX-24 enables display and manipulation of true color (24-bit per pixel) images at an attractive price. It can also be initialized to run in an 8-bit mode. Both 24-bit and 8-bit operations are accelerated by the RASTERFLEX hardware. The RASTERFLEX-24 fits in a single SBus slot, leaving room for other SBus peripherals.

8.1.1. RASTERFLEX FEATURES

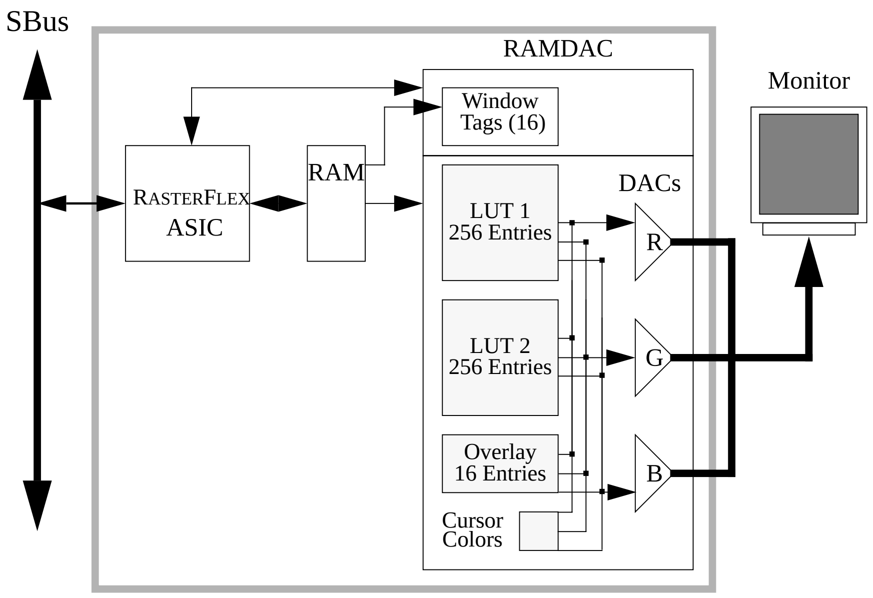
- Accelerated rectangle fill
- Accelerated Bit Block Transfer (BitBlt) (8 and 24 bit)
- Optimized host framebuffer access (for unaccelerated primitives)
- Two independent Look-Up Tables (LUTs) plus overlay and bypass (RASTERFLEX-32/HR)
- Single SBus slot
- 1024x768 76 Hz non-interlaced video resolution (RASTERFLEX-24/32/HR)
- 1152x900 66 Hz non-interlaced video resolution (RASTERFLEX-24/32/HR)
- 1152x900 76 Hz non-interlaced video resolution (RASTERFLEX-24/32/HR)
- 1280x1024 60 Hz non-interlaced video resolution (RASTERFLEX-HR)
- 1280x1024 67 Hz non-interlaced video resolution (RASTERFLEX-HR)
- 1280x1024 76 Hz non-interlaced video resolution (RASTERFLEX-HR)

- Multiple simultaneous pixel depths (RASTERFLEX-32/HR)

8.1.2. RASTERFLEX-32/HR ARCHITECTURE

The following figure depicts the architecture of the RASTERFLEX-32 and RASTERFLEX-HR cards.

Figure 8.1. RASTERFLEX-32 and RASTERFLEX-HR Block Diagram



The RASTERFLEX-32 and RASTERFLEX-HR consist of the RASTERFLEX ASIC, a RAMDAC, and framebuffer memory, plus a few support chips.

The RASTERFLEX ASIC performs most of the support functions required to interface to the SBus, generate video, and control the framebuffer memory. The architecture of the ASIC is optimized to provide efficient access from the host to the framebuffer for both 24-bit and 8-bit operations.

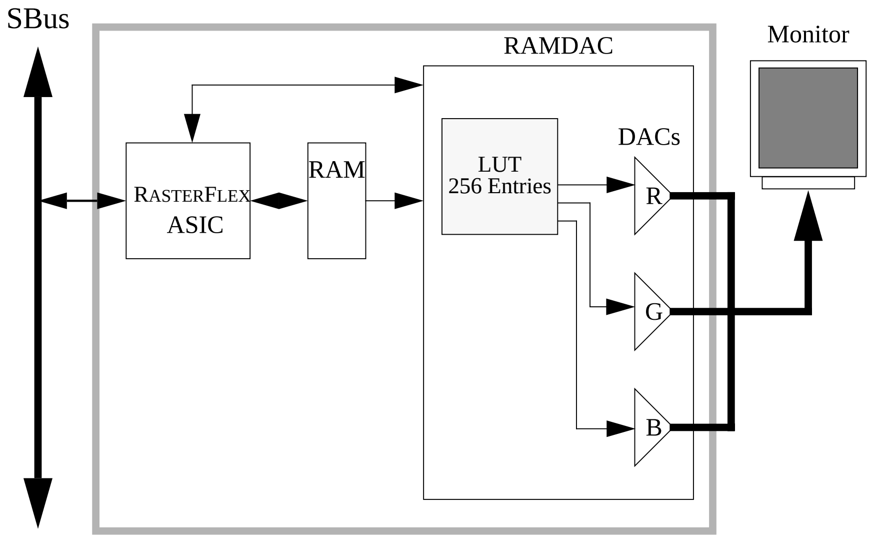
The RAMDAC performs lookup table mapping, pixel formatting for display, and generation of the video signals to the monitor. It provides two full 256-entry lookup tables, plus a 16-entry lookup table for overlay colormaps. These features allow multiple windows to maintain their own independent colormaps without the color flashing associated with single-lookup table framebuffers.

The framebuffer memory is made up of video dynamic RAMs, which are designed specifically for framebuffer applications. The RASTERFLEX-32 incorporates 4 Megabytes of RAM, while the RASTERFLEX-HR incorporates 8 Megabytes. The additional RAM on the RASTERFLEX-HR enables higher resolution display (1280x1024 vs. 1152x900).

8.1.3. RASTERFLEX-24 ARCHITECTURE

The following figure depicts the architecture of the RASTERFLEX-24 card.

Figure 8.2. RASTERFLEX-24 Block Diagram



The RASTERFLEX-24 consists of the RASTERFLEX ASIC, a RAMDAC, and framebuffer memory, plus a few support chips.

The RASTERFLEX ASIC performs most of the support functions required to interface to the SBus, generate video, and control the framebuffer memory. The architecture of the ASIC is optimized to provide efficient access from the host to the framebuffer for both 24-bit and 8-bit operations.

The RAMDAC performs lookup table mapping, pixel formatting for display, and generation of the video signals to the monitor. It provides a 256-entry lookup tables.

The framebuffer memory is made up of video dynamic RAMs, which are designed specifically for framebuffer applications. The RASTERFLEX-24 incorporates 3 Megabytes of RAM.

9. RASTERFLEX ADVANCED FEATURES

This section contains detailed descriptions of the advanced features of the RASTERFLEX hardware and is provided for X programmers who want to develop their own applications. The section includes descriptions on the following topics:

- How to use visual classes other than the server default. The RASTERFLEX-32 and RASTERFLEX-HR devices support multiple depths, an application may wish to create windows of a depth different from the server default.
- How to locate and use overlay visuals. The RASTERFLEX-32 and RASTERFLEX-HR devices provide overlay planes which are accessible to applications.
- How to use the Shared Memory Extension. The Shared Memory Extension is an efficient mechanism for transferring images and other raster data between the client and server when both are running on the system. The Shared Memory Extension is available on the RASTERFLEX-24, RASTERFLEX-32 and RASTERFLEX-HR devices.

Each description includes a simple example.

NOTE
This section only provides information on using the unique functionality associated with the RASTERFLEX accelerators. The concepts described in this section require some knowlege of X programming. If you are new to X programming, a more detailed reference may be needed to understand this section. If you want your programs to be portable, then you should carefully check for availability of these features before using them within your programs.

9.1. USING NON-DEFAULT VISUAL CLASSES

The following example creates an 8-bit and a 24-bit window, each with private modifiable colormaps. The information on which pixel depths are supported by the server is reported in a structure called an `XVisual.`

Use `XCreateWindow` instead of `XCreateSimpleWindow,` because the later always creates a window using the default visual.

The `Colormap` is not optional; the `Colormap` attribute for a window must match the visual class of the window.

You must also set the border pixel, since not setting it causes the window to default to having a `BorderPixmap` of `CopyFromParent`, in which case, it will be of the wrong depth. This example uses only standard Xlib functions; no server extensions are used.

```
#include <stdio.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>

Display             *display;
int                 screen;

XVisualInfo getVisual(desiredDepth)
```

```
{
    XVisualInfo visual;

    if (!XMatchVisualInfo(display, screen, desiredDepth, PseudoColor, &visual))
        if (!XMatchVisualInfo(display, screen, desiredDepth, DirectColor,
                              &visual))
            fprintf(stderr, “Unable to find visual, depth=%d”,
                    desiredDepth), exit(1);

    return visual;
}


main()
{
    int                 i;
    XVisualInfo         visual[2];
    Colormap            colorMap[2];
    Window              window[2];
    XSetWindowAttributes attr;
    XEvent              event;
    int                 mask;
    XColor              exactColor, actualColor;

    display = XOpenDisplay(““);
    screen = DefaultScreen(display);

    visual[0] = getVisual(8);
    visual[1] = getVisual(24);
    for (i = 0; i < 2; i++)
    {
        colorMap[i] = XCreateColormap(display, RootWindow(display, screen),
                                      visual[i].visual, AllocNone);
        if (colorMap[i] == NULL)
            fprintf(stderr, “Can’t create %d bit color map\n”,
                    visual[i].depth), exit(1);

        /* These window attributes must be set when creating a window
         * of a different depth than its parent.
         */
        mask                = 0;

        attr.colormap       = colorMap[i];
        mask               |= CWColormap;

        XAllocNamedColor(display, colorMap[i], White”,
                         &actualColor,&exactColor);
        attr.border_pixel   = actualColor.pixel;
        mask               |= CWBorderPixel;

        window[i] = XCreateWindow(display,RootWindow(display,screen),
                                  10+110*i,10,100,100, 1, visual[i].depth,
                                  InputOutput,visual[i].visual, mask, &attr);

        XMapRaised(display, window[i]);
    }

    while (XNextEvent(display, &event))
        /* Add event processing here */;
}
```

9.2. USING OVERLAYS

Overlays are implemented with a *property atom* defined on the root window of the RASTERFLEX screen by the server. The atom is called `SERVER_OVERLAY_VISUALS`.

This method for supporting overlays has become more or less agreed upon by several members of the X Consortium, but is still subject to change. If the atom is defined, then the server supports this style of overlays.

After you have found the atom, perform an `XGetWindowProperty()` call to get a list of `OverlayInfo` structures as defined below:

```
typedef enum {NotTransparent, TransparentPixel, TransparentMask} OverlayType;

typedef struct OverlayInfo {
    long       vid;
    OverlayTypetype;
    long       value;
    long       layer;
} OverlayInfo;
```

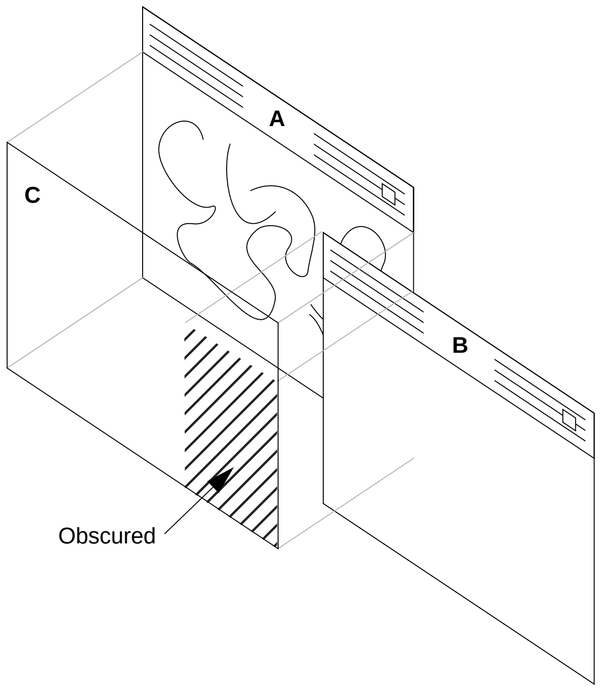
Overlay windows are created using a special visual class that the server defines with the following semantics:

- Pixels from underlay windows which normally would be obscured by an overlay window are visible where the overlay pixel value matches the type and value fields from the `OverlayInfo` structure above. If the pixel does not match, then the pixels from the overlay window itself are visible.
- Overlapping overlay windows do not combine their transparencies; whether an underlay window is visible is determined solely by the pixel values of the topmost overlay window.
- Overlay windows are treated identically to underlay windows in all other respects.

NOTE

The last bullet above deserves some additional commentary, because the notion of transparency does not match well with the X11 definition of visibility. Refer to the following figure for this discussion:

Figure 9.1. Transparency and visibility.



The two windows "*A*" and "*B*" are both underlay windows. "*C*" is a child of A and is an overlay window.

The pixels of *A* show through only where the pixels of *C* match the value defined in the `OverlayInfo` structure (more on this later). Since the overlay window *C* is treated as a regular window by the server, then *A* actually is obscured by *C* and no drawing operations that are directed at *A* will appear.

The solution is to set `GCSubwindowMode` to `IncludeInferiors` in the GC when drawing to *A*. This draws into all children of *A*, but only in the planes that *A* has in common with them, which does not include *C*.

Another consideration is that *A* receives no `Expose` events because it is always obscured by *C*. However, *C* can be obscured by other underlay windows such as *B*, so *C will* receive `Expose` events. In this case, you should go ahead and redraw both *C and A*, remembering to set `IncludeInferiors` when drawing *A*.

The pixel type and value returned in the `OverlayInfo` structure indicate how transparency is implemented for the overlay. If *`type`* is `TransparentMask`, then any pixel value in the overlay which has at least the same bits as *`value`* allows the underlay to show through, for example, `(pixel & value) == value`). If *`type`* is `TransparentPixel`, then any pixel value in the overlay *exactly* matching *`value`* allows the underlay to show through. A type of `NotTransparent` is useless and should never happen. The overlay field is used to distinguish between multiple layers of overlay data if support by the display hardware. The RASTERFLEX products only support a single layer so this field will always be set to one (1).

The 5-bit overlay implemented in the RASTERFLEX-32/HR uses the `TransparentMask` style with *`value`* equal to *`0x10`*.

> **NOTE**
> The value *`0x10`* is outside of the legal range for the colormap which allows only 16 entries. This is actually a feature, since it allows the overlay to display a full 16 opaque colors plus transparent. The 8-bit overlay uses the `TransparentPixel` style and is limited to displaying 255 opaque colors.

As mentioned earlier, the `XGetWindowProperty()` call can return multiple `OverlayInfo` structures. Each unique `vid` value represents a different overlay class which is simultaneously available. The same `vid` can appear multiple times, in which case the *`type`* and *`value`* fields for that visual are merged.

For instance, you can have both a transparent pixel value, as well as transparent mask bits. The RASTERFLEX device does not do any of this, however; it just returns one `OverlayInfo` structure.

Here is a complete program example using overlays.

```
/* ovdemo.c
 * This simple demonstration program will create a 24-bit underlay window
 * and fill it with a solid color (red). The window will have a child
 * which is an overlay window (either 5-bit or 8-bit depending on the
 * server overlay mode). Clicking any mouse button within the window
 * will cause a blue rectangular grid to be turned on/off within the
 * overlay window. Pressing any keyboard key will exit from the
 * demonstration.
 *
 * To compile under OpenWindows 3.0:
 *
 *     cc -o ovdemo ovdemo.c -I/usr/openwin/include -L/usr/openwin/lib -lX11
 *
 * To compile under X11R5:
 *
 *     cc -o ovdemo ovdemo.c -I/usr/X11R5/include -L/usr/X11R5/lib -lX11
 *
 * This demonstration will operate in either 4-bit or 8-bit overlay mode.
 */
#include <stdio.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>

typedef enum {NotTransparent, TransparentPixel, TransparantMask} OverlayType;

typedef struct OverlayInfo
{
    long vid;
    OverlayType type;
    long value;
    long layer;
} OverlayInfo;

/* Window width and height */
#define WWIDTH          500
#define WHEIGHT         500
```

```
/* Spacing of overlay grid */
#define GRID_SIZE        50

main()
{
    Display *dpy;
    int screen;
    Atom overlayAtom;
    int ignore;
    int overlayCount;
    OverlayInfo *pOverlayData;
    XVisualInfo *pOverlayVisual, *pUnderlayVisual;
    XVisualInfo matchInfo;
    Colormap overlayCmap, underlayCmap;
    XSetWindowAttributes winAttr;
    Window overlayWin, underlayWin;
    int exitFlag;
    XEvent event;
    XColor overlayColor, underlayColor, exactColor;
    GC overlayGC, underlayGC;
    XGCValues gcAttr;
    int gridOn = 0;
    int x, y;
    int winWidth = WWIDTH;
    int winHeight = WHEIGHT;
    XWindowChanges winChange;

    /*
     * Open the display and get the default screen
     */
    dpy = XOpenDisplay(“”);
    screen = DefaultScreen(dpy);

    /*
     * Get the overlay information from the server property.
     */
    overlayAtom = XInternAtom(dpy, “SERVER_OVERLAY_VISUALS”, True);
    if(!overlayAtom)
    {
        fprintf(stderr,”ovdemo: Cannot find SERVER_OVERLAY_VISUALS
property\n”);
        exit(1);
   }

    XGetWindowProperty(dpy, RootWindow(dpy, screen), overlayAtom, 0,
                       sizeof(OverlayInfo), False, AnyPropertyType,
                       &ignore, &ignore, &overlayCount,
                       &ignore, &pOverlayData);

    /*
     * Get the visual information associated with the first overlay
     * visual type.
     */
    matchInfo.visualid = pOverlayData->vid;
    pOverlayVisual = XGetVisualInfo(dpy, VisualIDMask, &matchInfo, &ignore);
    if(!pOverlayVisual)
    {
         fprintf(stderr,”ovdemo: Could not locate overlay visual info.\n”);
         exit(1);
    }
    if(pOverlayVisual->depth == 8)
        printf(“ovdemo: RasterFLEX is running in 8-bit overlay mode.\n”);
    else
        printf(“ovdemo: RasterFLEX is running in 4-bit overlay mode.\n”);

    /*
     * Get the visual information associated with the 24-bit TrueColor
     * underlay visual type.
     */
    matchInfo.depth = 24;
    matchInfo.class = TrueColor;
    pUnderlayVisual = XGetVisualInfo(dpy, VisualDepthMask|VisualClassMask,
                                     &matchInfo, &ignore);
    if(!pUnderlayVisual)
    {
         fprintf(stderr,”ovdemo: Could not locate underlay visual info.\n”);
         exit(1);
```

```
}

/*
 * Create an underlay and overlay colormaps of the appropriate visual
 * type if the visual is not the default visual.
 * Allocate a single pixel value in each colormap (red for
 * the underlay, blue for the overlay.
 */
if(pUnderlayVisual->visual == DefaultVisual(dpy, screen))
    underlayCmap = DefaultColormap(dpy, screen);
else
    underlayCmap = XCreateColormap(dpy, RootWindow(dpy, screen),
                                   pUnderlayVisual->visual, AllocNone);
if(!XAllocNamedColor(dpy, underlayCmap, “red”,
                     &underlayColor, &exactColor))
{
    fprintf(“ovdemo: Could not allocate underlay color\n”);
    exit(1);
}

if(pOverlayVisual->visual == DefaultVisual(dpy, screen))
    overlayCmap = DefaultColormap(dpy, screen);
else
    overlayCmap = XCreateColormap(dpy, RootWindow(dpy, screen),
                                  pOverlayVisual->visual, AllocNone);
if(!XAllocNamedColor(dpy, overlayCmap, “blue”,
                     &overlayColor, &exactColor))
{
    fprintf(“ovdemo: Could not allocate overlay color\n”);
    exit(1);
}

/*
 * Create the underlay window as a child of the root window. Set
 * the border_pixel and colormap attributes to prevent BadMatch
 * errors for non-default visuals.
 */
winAttr.border_pixel = 0;
winAttr.colormap = underlayCmap;
winAttr.event_mask = StructureNotifyMask;
underlayWin = XCreateWindow(dpy, RootWindow(dpy, screen), 0, 0,
                            winWidth, winHeight, 0, pUnderlayVisual->depth,
                            InputOutput, pUnderlayVisual->visual,
                            CWBorderPixel|CWColormap|CWEventMask,
                            &winAttr);

/*
 * Create the overlay window which is a child of the underlay window.
 * Set the event mask to receive exposures and user input for the
 * overlay window.
 * NOTE: the window background pixel is set to the transparent value.
 */
winAttr.background_pixel = pOverlayData->value;
winAttr.border_pixel = 0;
winAttr.colormap = overlayCmap;
winAttr.event_mask = KeyPressMask|ButtonPressMask|ExposureMask;
overlayWin = XCreateWindow(dpy, underlayWin, 0, 0,
                    winWidth, winHeight, 0, pOverlayVisual->depth,
                    InputOutput, pOverlayVisual->visual,
                    CWBackPixel|CWBorderPixel|CWColormap|CWEventMask,
                    &winAttr);

/*
 * Set the WM_COLORMAP_WINDOWS property so the window manager will
 * know to properly install both underlay and overlay colormaps when
 * the demo gets colormap focus. This is done on the underlay window
 * since it is the top-level (i.e. child of root) window.
 */
XSetWMColormapWindows(dpy, underlayWin, &overlayWin, 1);

/*
 * Create a graphics context for use in rendering to each window,
 * setting the foreground color in each to the allocated pixels.
 * NOTE: The underlay GC must have the IncludeInferiors sub_window
 *       mode set to prevent window clipping by the (overlay)
 *       child.
 */
```

```
gcAttr.subwindow_mode = IncludeInferiors;
gcAttr.foreground = underlayColor.pixel;
underlayGC = XCreateGC(dpy, underlayWin,
                       GCForeground|GCSubwindowMode, &gcAttr);

gcAttr.foreground = overlayColor.pixel;
overlayGC = XCreateGC(dpy, overlayWin, GCForeground, &gcAttr);

/*
 * Map the windows.
 */
XMapWindow(dpy, overlayWin);
XMapWindow(dpy, underlayWin);

/*
 * Now wait for events and process them.
 */
exitFlag = 0;
while(!exitFlag)
{
    XNextEvent(dpy, &event); switch(event.type)
    {
    case Expose:

        /*
         * On exposure, fill the underlay planes with the fill color.
         */
        XFillRectangle(dpy, underlayWin, underlayGC,
                       event.xexpose.x, event.xexpose.y,
                       event.xexpose.width, event.xexpose.height);

        /*
         * On exposure, fill the overlay planes with the grid (if
         * it is enabled. For simplicity, we redraw the entire
         * grid. For complex overlay, only the exposed area should
         * be refreshed.
         */
        if(gridOn)
        {
            for(x = 0; x < winWidth; x += GRID_SIZE)
                XDrawLine(dpy, overlayWin, overlayGC,
                          x, 0, x, winHeight);
            for(y = 0; y < winHeight; y += GRID_SIZE)
                XDrawLine(dpy, overlayWin, overlayGC,
                          0, y, winWidth, y);
        }
        break;

    case ButtonPress:

        /*
         * Toggle the state of the overlay grid flag. Draw the
         * overlay grid if it is turned on.
         */
        gridOn = !gridOn;

        if(gridOn)
        {
            for(x = 0; x < winWidth; x += GRID_SIZE)
                XDrawLine(dpy, overlayWin, overlayGC,
                          x, 0, x, winHeight);
            for(y = 0; y < winHeight; y += GRID_SIZE)
                XDrawLine(dpy, overlayWin, overlayGC,
                          0, y, winWidth, y);
        }
        else
            XClearArea(dpy, overlayWin, 0, 0, winWidth, winHeight, False);

        break;

    /*
     * Exit if any keyboard key is pressed.
     */
    case KeyPress:
        exitFlag = 1;
        break;
```

```
            /*
             * If the underlay window gets resized, update the width and height
             * parameters and enlarge the child overlay window.
             */
            case ConfigureNotify:
                winWidth = event.xconfigure.width;
                winHeight = event.xconfigure.height;
                winChange.width = winWidth;
                winChange.height = winHeight;
                XConfigureWindow(dpy, overlayWin, CWWidth|CWHeight, &winChange);

            }
        }

        XCloseDisplay(dpy);
        exit(0);
    }
```

9.3. USING SHARED MEMORY

The shared memory extension provides the ability to read and write `XImages` or pixmaps without incurring the overhead of having the images passed through the Xlib interprocess communications channel. This only works when the server and client are running on the same workstation.

The extension relies on the System V shared memory facility (**`shmget`**, **`shmmat`**). Unfortunately, Sun does not make it easy to use this facility under SunOS 4.1.1. The facility is not in the small kernel which Sun ships with the IPC and IPX. Shared memory support is enabled by default for Solaris 2. For both SunOS 4.1.X and Solaris 2, the default maximum shared memory segment size is limited to 1 Megabyte. Consult your Sun documentation to find out how to reconfigure your OS to support System V shared memory primitives.

Shared memory images/pixmaps are implemented using a *`server extension`*. Test for the extension before using it, using either of the following two methods to do this:

```
Status XShmQueryExtension(display)
    Display *display;
```

or

```
Status XShmQueryVersion(display, major, minor, pixmaps)
    Display *display;
    int     *major, *minor;
    Bool    *pixmaps;
```

Either will return *`True`* if the extension is available. The arguments `major` and `minor` return the version numbers of the implementation. If `pixmaps` is *`True`*, then the server supports shared memory Pixmaps in addition to `XImages` (this is supported on the RASTERFLEX, see *USE OF SHARED MEMORY PIXMAPS* late in this section)

9.3.1. USING SHARED MEMORY IMAGES.

The basic sequence of operations for shared memory `XImages` is as follows:

1. Create the shared memory `XImage` structure.
2. Create a shared memory segment to store the image data.
3. Inform the server about the shared memory segment.

4. Use the shared memory `XImage`, much like a normal one.

Include the following header files in your code:

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/extensions/XShm.h>
```

To create a shared memory `XImage`, use:

```
XImage *XShmCreateImage(display, visual, depth, depth, format, data,
                        shminfo, width, height)
    Display        *display;
    Visual         *visual;
    unsigned int   depth, width, height;
    int            format;
    char           *data;
    XShmSegmentInfo*shminfo;
```

Most of the arguments are the same as for `XCreateImage`; they are be repeated here.

NOTE
There are no `offset`, `bitmap_pad`, or `bytes_per_line` arguments. These quantities are defined by the server itself, and your code needs to abide by them. Unless you already allocated the shared memory segment (see below), you should pass in NULL for the "data" pointer.

There is one additional argument: `shminfo`, which is a pointer to a structure of type `XShmSegmentInfo`. You must allocate one of these structures such that it has a lifetime at least as long as that of the shared memory `XImage`. There is no need to initialize this structure before the call to `XShmCreateImage`.

The return value, if all goes well, is an Image structure that you can use for the subsequent steps.

The next step is to create the shared memory segment. This is best done after the creation of the `XImage`, since you need to use the information in that `XImage` to know how much memory to allocate. To create the segment, you need a call such as the following:

```
shminfo.shmid = shmget(IPC_PRIVATE, image->bytes_per_line * image->height,
                       IPC_CREAT|0777);
```

where `image` is your shared memory `XImage`.

Of course, follow the rules and do error checking on all of these system calls. Also, be sure to use the `bytes_per_line` field, not the width you used to create the `XImage` as they can be different.

NOTE
The shared memory ID returned by the system is stored in the `shminfo` structure. The server needs that ID to attach itself to the segment.

Next, attach this shared memory segment to your process:

```
shminfo.shmaddr = image->data = shmat(shminfo.shmid, 0, 0);
```

The address returned by **`shmat`** should be stored in **both** the `XImage` structure and the `shminfo` structure.

To finish filling in the `shminfo` structure, decide how you want the server to attach to the shared memory segment, and set the `readOnly` field as follows. Normally, you would code:

```
shminfo.readOnly =- False
```

If you set it to *`True`*, the server is not able to write to this segment, and `XShmGetImage` calls fail.

Finally, tell the server to attach to your shared memory segment with:

```
Status XShmAttach(display, &shminfo);
```

If all goes well, you get a non-zero status back, and your `XImage` is ready for use.

To write a shared memory `XImage` into an X drawable, use `XShmPutImage`:

```
Status XShmPutImage(display, d, gc, image, src_x, src_y, dest_x, dest_y,
                   width, height, send_event);
    Display       display;
    Drawable      d;
    GC            gc;
    XImage        *image;
    int           src_x, src_y, dest_x, dest_y;
    unsigned int  width, height;
    Bool          send_event;
```

The interface is identical to that of `XPutImage` and is not documented here.

One additional parameter is called `send_event`. If this parameter is passed as *`True`*, the server generates a “completion” event when the image write is complete. As a result, your program can know when it is safe to begin manipulating the shared memory segment again.

The completion event has type `XShmCompletionEvent`, which is defined as follows:

```
typedef struct {
    int type;              /* of event */
    unsigned long serial;  /* # of last request processed by server */
    Bool send_event;       /* true if this came from a SendEvent request */
    Display *display;      /* Display the event was read from */
    Drawable drawable;     /* drawable of request */
    int major_code;        /* ShmReqCode */
    int minor_code;        /* X_ShmPutImage */
    ShmSeg shmseg;         /* the ShmSeg used in the request */
    unsigned long offset;  /* the offset into ShmSeg used in the request */
} XShmCompletionEvent;
```

The event type value that is used can be determined at run time with a line of the form:

```
int CompletionType = XShmGetEventBase(display) + ShmCompletion;
```

If you modify the shared memory segment before the arrival of the completion event, the results you see on the screen can be inconsistent.

To read image data into a shared memory `XImage`, use the following:

```
Status XShmGetImage(display, d, image, x, y, plane_mask)
    Display       *display;
    Drawable      d;
    XImage        image;
    int           x, y;
    unsigned long plane_mask;
```

where *`display`* is the display of interest, *`d`* is the source drawable, *`image`* is the destination `XImage`, *`x`* and *`y`* are the offset within *`d`*, and *`plane_mask`* defines which planes are to be read.

To destroy a shared memory `XImage`, first instruct the server to detach from it, then destroy the segment itself, as follows:

```
XShmDetach(display, &shminfo);
XDestroyImage(image);
shmdt(shminfo.shmaddr);
shmctl(shminfo.shmid, IPC_RMID, 0);
```

9.3.2. USE OF SHARED MEMORY PIXMAPS

Unlike X images, for which any image format is usable, the shared memory extension supports only a single format, for example, `XYPixmap` or `ZPixmap`, for the data stored in a shared memory pixmap. This format is independent of the depth of the image (for 1-bit pixmaps it does not really matter what this format is) and independent of the screen. Use `XShmPimapFormat` to get the format for the server:

```
int XShmPixmapFormat(display)
   Display    *display;
```

If your application can deal with the server pixmap data format, including `bits-per-pixel`, create a shared memory segment and `shminfo` structure in exactly the same way as is listed above for shared memory `XImages`. While it is not strictly necessary to create an `XImage` first, doing so incurs little overhead and gives you an appropriate `bytes_per_line` value to use.

Once you have your `shminfo` structure filled in, simply call:

```
Pixmap XShmCreatePixmap(display, d, data, shminfo, width, height, depth)
   Display        *display;
   Drawable       d;
   char           *data;
   XShmSegmentInfo*shminfo;
   unsigned int   width, height, depth;
```

The arguments are all the same as for `XCreatePixmap`, with two additions: `data` and `shminfo`. The second of the two is the same old `shminfo` structure that has been used before. The first is the pointer to the shared memory segment, and should be the same as the `shminfo.shmaddr` field. It is unclear why this is a separate parameter.

If everything works, you get back a pixmap, which you can manipulate in all the usual ways, with the added bonus of being able to tweak its contents directly through the shared memory segment. Shared memory pixmaps are destroyed in the usual manner with `XFreePixmap`, though you should detach and destroy the shared memory segment itself as shown above.

Here is a complete example using shared memory `XImages`:

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/extensions/XShm.h>

extern char *shmat();

Display *display;
int screen;

main()
{
   Visual             visual;
```

```
XShmSegmentInfo     shminfo;
Window              window;
XEvent              event;
XMotionEvent        *motion = (XMotionEvent *)&event;
XExposeEvent        *expose = (XExposeEvent *)&event;
XImage              *image;
XGCValues           gcv;
GC                  gc;
int                 shmCompletion;
int                 busy;
int                 x,y, lastX, lastY, maxx,maxy;

display             = XOpenDisplay(“”);
screen              = DefaultScreen(display);

if (!XShmQueryExtension(display))
    fprintf(stderr, “Shared memory extension not present\n”), exit(1);

/* Get event number for shared memory operation completion */
shmCompletion = XShmGetEventBase(display) + ShmCompletion;

/* Create image and set up shared memory */
image = XShmCreateImage(display, DefaultVisual(display, screen),
                        DefaultDepth(display, screen), ZPixmap, NULL,
                        &shminfo, 256, 256);
shminfo.shmid = shmget(IPC_PRIVATE, image->bytes_per_line * image->height,
                       IPC_CREAT | 0777);
if ((int)shminfo.shmid == -1)
    perror(“shmget”), exit(1);

shminfo.shmaddr = image->data = shmat(shminfo.shmid, 0, 0);
if ((int)shminfo.shmaddr == -1)
    perror(“shmat”), exit(1);
shminfo.readOnly = False;
if (!XShmAttach(display, &shminfo))
    fprintf(stderr, “XShmAttach failed\n”), exit(1);

window = XCreateSimpleWindow(display,RootWindow(display,screen), 10, 10,
                             256, 256, 1, WhitePixel(display,screen),
                             WhitePixel(display,screen));
XSelectInput(display, window, ExposureMask | KeyPressMask |ButtonMotionMask
                              | ButtonPressMask | ButtonReleaseMask);

XMapRaised(display, window);

maxx = DisplayWidth(display, screen) - image->width;
maxy = DisplayHeight(display, screen) - image->height;
lastX = lastY = x = y = busy = 0;

XShmGetImage(display, DefaultRootWindow(display), image, x, y, AllPlanes);

gcv.graphics_exposures = False;
gc = XCreateGC(display, window, GCGraphicsExposures, &gcv);

while (1)
{
    XNextEvent(display, &event);
    switch (event.type) {
      case Expose:
        XShmPutImage(display, expose->window, gc, image,
                     expose->x, expose->y, expose->x, expose->y,
                     expose->width, expose->height, True);
        busy++;
        break;

      case KeyPress:
        exit(0);

      case MotionNotify:
        /* Skip to last queued Motion event */
        if (XEventsQueued(display, QueuedAfterReading))
        {
            XPeekEvent(display, &event);
            if (event.type == MotionNotify)
                break;
        }
```

```
            x = motion->x_root;
            y = motion->y_root;
            if (x > maxx)
                x = maxx;
            if (y > maxy)
                y = maxy;
            break;

          default:
            if (event.type == shmCompletion)
                busy--;
            break;
        }

        if (!busy && (lastX != x || lastY != y))
        {
            lastX = x;
            lastY = y;

            XShmGetImage(display, DefaultRootWindow(display), image, x, y,
                         AllPlanes);
            XShmPutImage(display, window, gc, image, 0, 0, 0, 0,
                         image->width, image->height, True);
            busy++;
        }
    }
}
```

Appendix A. SPECIFICATIONS

This appendix includes specifications for the RASTERFLEX-24, RASTERFLEX-32, and RASTERFLEX-HR raster accelerators. The specifications include physical characteristics and performance specifications.

PHYSICAL CHARACTERISTICS — RASTERFLEX-24

This following table describes physical characteristics for the single-slot RASTERFLEX-24 raster accelerator card.

Table A.1. Physical Characteristics of RF-24

Dimensions	Single Slot SBus 3.3” x 5.776”
Power	1.5A @ 5 VDC
Temperature	0-40 deg C
Humidity	5 - 80% non-condensing
Qualifications	FCC Part 15, Subpart B, Class A

PHYSICAL CHARACTERISTICS — RASTERFLEX-32

This following table describes physical characteristics for the single-slot RASTERFLEX-32 raster accelerator card.

Table A.2. Physical Characteristics of RF-32

Dimensions	Single Slot SBus 3.3” x 5.776”
Power	1.9A @ 5 VDC
Temperature	0-40 deg C
Humidity	5 - 80% non-condensing
Qualifications	FCC Part 15, Subpart B, Class A

PHYSICAL CHARACTERISTICS — RASTERFLEX-HR

This following tables describe the physical characteristics for the single-slot RASTERFLEX-HR raster accelerator card.

Table A.3. Physical Characteristics for RF-HR

Dimensions	Single Slot SBus 3.3" x 5.776"
Power	2A @ 5 VDC
Temperature	0-40 deg C
Humidity	5 - 80% non-condensing
Qualifications	FCC Part 15, Subpart B, Class A

RASTERFLEX-24/32/HR PERFORMANCE

This following tables provide the 8-bit and 24-bit performance specifications for the RASTERFLEX raster accelerator cards on a SPARC Classic.

Table A.4. 8-bit Performance Specifications for RF-24/32/HR

Rectangle Fill	165 MegaPixels per second
BitBlt	14.3 MegaPixels per second
Frame Buffer Writes	18.6 MegaPixels per second
Text (6x13 image line)	104 K Characters per second

Table A.5. 24-bit Performance Specifications for RF-24/32/HR

Rectangle Fill	58 MegaPixels per second
BitBlt	4.8 MegaPixels per second
Frame Buffer Writes	4.6 MegaPixels per second
Text (6x13 image line)	85.3 K Characters per second

Appendix B. VIDEO FORMATS

This appendix includes horizontal and vertical timing for the RASTERFLEX video formats. These formats include 1024x768 (76 Hz), 1152x900 (66 Hz), 1152x900 (76 Hz), 1280x1024 (60 Hz), 1280x1024 (67Hz) and 1280x1024 (76Hz).

1024x768 76 HZ VIDEO FORMAT

The following tables describe the horizontal and vertical timing for the 1024x768 76 Hz video format used by the RASTERFLEX-24, RASTERFLEX-32, and RASTERFLEX-HR cards.

Table B.1. Horizontal timing for 1024x768 76 Hz video format

Parameter	Pixel Clocks	Time
Pixel Clock Frequency		84.3182 MHz
Pixel Clock Period	1	11.866 nsec
Horizontal Frequency		61.999 kHz
Horizontal Period	1360	16.129 usec
Horizontal Sync	128	1.518 usec
Horizontal Back Porch	176	2.087 usec
Horizontal Active	1024	12.144 usec
Horizontal Front Porch	32	.380 usec
Horizontal Blanking	336	3.985 usec

Table B.2. Vertical timing for 1024x768 76 Hz video format

Parameter	Lines	Time
Vertical Frequency		77.017 Hz
Vertical Period	805	12.984 msec
Vertical Sync	4	64.52 usec
Vertical Back Porch	30	483.88 usec
Vertical Active	768	12.387 msec
Vertical Front Porch	3	43.39 usec
Vertical Blanking	37	596.8 usec

Frequency Tolerance:	100 ppm (.01%)
Impedance:	75 ohms
Amplitude:	.660V with no setup

Separate TTL Sync

1152x900 66 HZ VIDEO FORMAT

The following tables describe the horizontal and vertical timing for the 1152x900 66 Hz video format used by the RASTERFLEX-24, RASTERFLEX-32, and RASTERFLEX-HR cards.

Table B.3. Horizontal timing for 1152x900 66 Hz video format

Parameter	Pixel Clocks	Time
Pixel Clock Frequency		92.9331 MHz
Pixel Clock Period	1	10.76 nsec
Horizontal Frequency		61.791 kHz
Horizontal Period	1504	16.184 usec
Horizontal Sync	128	1.377 usec
Horizontal Back Porch	192	2.066 usec
Horizontal Active	1152	12.396 usec
Horizontal Front Porch	32	.344 usec
Horizontal Blanking	352	3.788 usec

Table B.4. Vertical timing for 1152x900 66 Hz video format

Parameter	Lines	Time
Vertical Frequency		65.95 Hz
Vertical Period	937	15.164 msec
Vertical Sync	4	64.74 usec
Vertical Back Porch	31	501.69 usec
Vertical Active	900	14.565 msec
Vertical Front Porch	2	32.37 usec
Vertical Blanking	37	598.8 usec

Frequency Tolerance:	100 ppm (.01%)
Impedance:	75 ohms
Amplitude:	.714V with 0.054V setup

Separate TTL Sync

1152x900 76 HZ VIDEO FORMAT

The following tables describe the horizontal and vertical timing for the 1152x900 76 Hz video format used by the RASTERFLEX-24, RASTERFLEX-32, and RASTERFLEX-HR cards.

Table B.5. Horizontal timing for 1152x900 76 Hz video format

Parameter	Pixel Clocks	Time
Pixel Clock Frequency		105.5615 MHz
Pixel Clock Period	1	9.47 nsec
Horizontal Frequency		71.713 kHz
Horizontal Period	1472	13.944 usec
Horizontal Sync	96	.909 usec
Horizontal Back Porch	208	1.970 usec
Horizontal Active	1152	10.913 usec
Horizontal Front Porch	16	.1516 usec
Horizontal Blanking	320	3.031 usec

Table B.6. Vertical timing for 1152x900 66 Hz video format

Parameter	Lines	Time
Vertical Frequency		76.048 Hz
Vertical Period	943	13.150 msec
Vertical Sync	8	111.6 usec
Vertical Back Porch	33	460.2 usec
Vertical Active	900	12.55 msec
Vertical Front Porch	2	27.89 usec
Vertical Blanking	43	599.6 usec

Frequency Tolerance:	100 ppm (.01%)
Impedance:	75 ohms
Amplitude:	.660V with no setup

Separate TTL Sync

1280x1024 60Hz VIDEO FORMAT

The following tables describe the horizontal and vertical timing for the 1280x1024 60 Hz video format used by the RASTERFLEX-HR card.

Table B.7. Horizontal timing for 1280x1024 60Hz video format

Parameter	Pixel Clocks	Time
Pixel Clock Frequency		107.3864 MHz
Pixel Clock Period	1	9.31 nsec
Horizontal Frequency		63.317 kHz
Horizontal Period	1696	15.793 usec
Horizontal Sync	176	1.639 usec
Horizontal Back Porch	200	1.862 usec
Horizontal Active	1280	11.92 usec
Horizontal Front Porch	40	0.372 usec
Horizontal Blanking	416	3.874 usec

Table B.8. Vertical timing for 1280x1024 60Hz video format

Parameter	Lines	Time
Vertical Frequency		59.96 Hz
Vertical Period	1056	16.678 msec
Vertical Sync	3	47.38 usec
Vertical Back Porch	26	410.63 usec
Vertical Active	1024	16.17 msec
Vertical Front Porch	3	47.38 usec
Vertical Blanking	32	505.4 usec

Frequency Tolerance:	100 ppm (.01%)
Impedance:	75 ohms
Amplitude:	.714V with 0.054V setup

Separate TTL sync and sync on green

1280x1024 67Hz VIDEO FORMAT

The following tables describe the horizontal and vertical timing for the 1280x1024 67 Hz video format used by the RASTERFLEX-HR card.

Table B.9. Horizontal timing for 1280x1024 67Hz video format

Parameter	Pixel Clocks	Time
Pixel Clock Frequency		117.0356 MHz
Pixel Clock Period	1	8.54 nsec
Horizontal Frequency		71.713 kHz
Horizontal Period	1632	13.944 usec
Horizontal Sync	112	0.957 usec
Horizontal Back Porch	224	1.914 usec
Horizontal Active	1280	10.937 usec
Horizontal Front Porch	16	0.137 usec
Horizontal Blanking	352	3.008 usec

Table B.10. Vertical timing for 1280x1024 67Hz video format

Parameter	Lines	Time
Vertical Frequency		66.71 Hz
Vertical Period	1075	14.990 msec
Vertical Sync	8	111.56 usec
Vertical Back Porch	37	515.95 usec
Vertical Active	1024	14.279 msec
Vertical Front Porch	6	83.667 usec
Vertical Blanking	51	711.17 usec

Frequency Tolerance:	100 ppm (.01%)
Impedance:	75 ohms
Amplitude:	.660V with no setup

Separate TTL Sync

1280x1024 76 HZ VIDEO FORMAT

The following tables describe the horizontal and vertical timing for the 1280x1024 76 Hz video format used by the RASTERFLEX-HR card.

Table B.11. Horizontal timing for 1280x1024 76Hz video format

Parameter	Pixel Clocks	Time
Pixel Clock Frequency		135 MHz
Pixel Clock Period	1	7.41 nsec
Horizontal Frequency		81.13kHz
Horizontal Period	1664	12.326 usec
Horizontal Sync	64	0.474 usec
Horizontal Back Porch	288	2.133 usec
Horizontal Active	1280	9.482 usec
Horizontal Front Porch	32	0.237 usec
Horizontal Blanking	384	2.844 usec

Table B.12. Vertical timing for 1280x1024 76Hz video format

Parameter	Lines	Time
Vertical Frequency		76.107 Hz
Vertical Period	1066	13.139 msec
Vertical Sync	8	98.607 usec
Vertical Back Porch	32	394.43 usec
Vertical Active	1024	12.622 msec
Vertical Front Porch	2	24.652 usec
Vertical Blanking	42	517.7 usec

Frequency Tolerance:	100 ppm (.01%)
Impedance:	75 ohms
Amplitude:	.660V with no setup

Separate TTL Sync

INDEX

Connectware

Document Comment Form

We would appreciate your comments about this document so that we can continue to improve our communication with our customers. Please take the time to share with us any errors which you have found or any suggestions that will help us improve this document.

Document Name	Document Number

Your Name

Title

Company Name and Address

Date

Technical Errors (Please indicate page number, correction, and text in which error was found.)

Typographical Errors (Please indicate page number and text in which error was found.)

Need Additional Information? (Please indicate which areas need more detail.)

Information Unclear? (Please indicate what information may have been unclear.)

General Comments

(Thank you for any comments you may have on this document, its layout, and discussion.)

Please copy this page and mail or fax your comments to:

Technical Publications
Connectware, Inc.
1301 East Arapaho Road
Richardson, TX 75081

Fax number 214 997 4309

Connectware

1301 East Arapaho
Richardson,
TX 75081

16-DA3019-2