

XRunner®
Installation Guide
for Sun Workstations
Version 6.0

XRunner Installation Guide

© Copyright 1994 - 1999 by Mercury Interactive Corporation

All rights reserved. All text and figures included in this publication are the exclusive property of Mercury Interactive Corporation, and may not be copied, reproduced, or used in any way without the express permission in writing of Mercury Interactive. Information in this document is subject to change without notice and does not represent a commitment on the part of Mercury Interactive.

Mercury Interactive may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents except as expressly provided in any written license agreement from Mercury Interactive.

WinRunner, XRunner, LoadRunner, TestDirector, TestSuite, and WebTest are registered trademarks of Mercury Interactive Corporation in the United States and/or other countries. Astra, Astra SiteManager, Astra SiteTest, RapidTest, QuickTest, Visual Testing, Action Tracker, Link Doctor, Change Viewer, Dynamic Scan, Fast Scan, and Visual Web Display are trademarks of Mercury Interactive Corporation in the United States and/or other countries.

This document also contains registered trademarks, trademarks and service marks that are owned by their respective companies or organizations. Mercury Interactive Corporation disclaims any responsibility for specifying which marks are owned by which companies or organizations.

If you have any comments or suggestions regarding this document, please send them via e-mail to documentation@mercury.co.il. Mercury Interactive Corporation
1325 Borregas Avenue
Sunnyvale, CA 94089 USA

Table of Contents

| | |
|--|----|
| Introduction | 1 |
| Upgrading from a Previous Version | 1 |
| Typographical Conventions..... | 2 |
| System Requirements | 3 |
| Before You Begin | 4 |
| Installing XRunner | 5 |
| Accessing the Installation Program..... | 5 |
| Running the Installation Program | 6 |
| Checking Your XRunner Installation..... | 9 |
| Troubleshooting | 10 |
| Setting Up the Environment | 11 |
| Installing Required Extensions..... | 11 |
| Setting Environment Variables | 13 |
| Checking .Xdefaults File Settings..... | 14 |
| Defining Softkeys | 15 |
| Modifying the Mouse Configuration | 17 |
| Starting XRunner | 18 |
| Troubleshooting | 18 |
| The XRunner License Manager | 19 |
| xr_elmadmin—License Manager Administration Program | 20 |
| xrelmd—License Manager Daemon | 24 |
| Creating Backup License Managers..... | 27 |
| xr_elmreport—Report Generator | 28 |
| xr_elmdecode—License Manager Code and Key Decoder..... | 31 |

| | |
|--|----|
| Linking the Context Sensitive Library | 33 |
| Linking with the xr_analyze Utility..... | 33 |
| SunOS—Motif Applications | 34 |
| SunOS—XView Applications..... | 40 |
| SunOS—OLIT Applications | 44 |
| Solaris—Motif Applications..... | 47 |
| Solaris—XView Applications..... | 52 |
| Solaris—OLIT Applications | 56 |
| Troubleshooting | 59 |
| Enabling Java Support | 63 |
| The Java Support Environment..... | 63 |
| Setting Up Java Support | 64 |
| Troubleshooting | 65 |

Introduction

This guide covers the installation of XRunner. The first three chapters describe the three main steps of the installation process:

- Downloading XRunner from the installation tape or CD-ROM
- Setting up the environment
- Invoking XRunner

At the end of each chapter you will find a troubleshooting section that provides solutions to problems you might encounter.

Depending on your application and the type of license you purchased, you might also find useful information in one of the appendixes which cover the following topics:

- The XRunner license manager
- Context Sensitive Installation

For information on configuring your keyboard, refer to the *XRunner User's Guide*.

Once you have completed installation, refer to the additional documentation for details on working with XRunner.

Upgrading from a Previous Version

The installation program automatically scans your environment to detect an existing installation of XRunner. If the environment parameter `M_ROOT` is defined, it is assumed to point to the root of the XRunner installation tree. Unless you specify otherwise, `XRinstall` will overwrite the current installation with the contents of the new release, except for the checklists and fonts and filters libraries.

After installing a new version of XRunner, be sure to kill any previously running *mc_svc* processes, to make sure the new process from the new version will now run.

Typographical Conventions

This guide uses the following typographical conventions:

| | |
|----------------|---|
| Bold | Bold text indicates function names and the elements of the functions that are to be typed in literally. |
| <i>Italics</i> | <i>Italic</i> text indicates variable names. |
| Helvetica | The Helvetica font is used for examples and statements that are to be typed in literally. |
| [] | Square brackets enclose optional parameters. |
| { } | Curly brackets indicate that one of the enclosed values must be assigned to the current parameter. |
| ... | In a line of syntax, three dots indicate that more items of the same format may be included. In a program example, three dots are used to indicate lines of a program that have been intentionally omitted. |
| | A vertical bar indicates that either of the two options separated by the bar should be selected. |

System Requirements

| | |
|--------------------------|---|
| Operating System | Sun SparcStations running the Solaris 2.5 operating system or higher. XRunner supports SPARC-compatible workstations running their versions of SunOS. |
| X servers | The OpenWindows version 3.0 or 4.0, or the MIT-based X11R5, X11R6 X or X11RG server using the XTestExtension1, the XTrap extensions or XRecord. |
| Memory | 32 megabytes (minimum), with 60 megabytes swap space (recommended) |
| Disk Requirements | 85 megabytes for SunOS 70 megabytes for Solaris Additional 5 megabytes per test (recommended) |
| Window Managers | mwm, olwm, olvwm, vewwm, twm, fvwm or any ICCCM standard window manager. |
| Media | CD-ROM |

Before You Begin...

To ensure a smooth installation, it is recommended that you perform the following before you start installing XRunner.

- ▶ Verify that an appropriate tape drive or CD-ROM drive is attached to your local machine or to a remote machine on your network.
- ▶ Find out which device driver is available for use with this drive. The device driver is usually located in the */dev* directory of the machine to which the tape or CD-ROM drive is attached. Consult your system administrator for the name of your device.
- ▶ Make sure that there is sufficient storage space in the disk partition where you intend to install XRunner. Refer to the System Requirements section.
- ▶ Determine which host machine connected to your network is to be used to install the XRunner license manager and licenses purchased for your site. Note that the license manager may not be installed on a machine which is named localhost or localhost. In addition, the internet address of the host on which the license manager is installed may not be 127.0.0.1.
- ▶ In order to be able to install the XRunner License Manager, you should login as root, so that you will have write permission for the system startup file or directory.
- ▶ Insert the Installation tape in the tape drive or the CD-ROM in the CD-ROM drive.

Installing XRunner

To install XRunner, you need to access the installation program from the tape or CD-ROM, and then run the program. This section includes a description of the `xr_verify` troubleshooting utility.

Accessing the Installation Program

XRunner is installed using the tape or CD-ROM provided with the XRunner distribution package.

- To install XRunner from a tape, you must first download the installation program (`install.xr`) onto disk, and then run this program.
- To install XRunner from the CD-ROM, run the installation program, `install.xr`, directly from the CD-ROM.

Downloading the Installation Program from a Tape

Below, an example is used to illustrate how to download the installation program. This example assumes that the tape drive is attached to a host named *elco*, and that the device name *rst0* is to be used. (Note that the device name will vary, depending on your platform.)

1 Login to the host.

Login to the host. (If the tape drive is located on a remote machine—not the one you are logged on to—then `rlogin` to the remote machine.)

2 Change directory.

Change the current working directory to the one where you want to store the `install.xr` program. For example, if you want to place `install.xr` under the `/tools/QA` directory, type: `cd /tools/QA`.

3 Download `install.xr`.

Download `install.xr` from the distribution tape, by entering the command:

```
tar xvf tape_device
```

For example, if you are using the device *rst0*, then type:

```
tar xvf /dev/rst0
```

A file named *install.xr* should now exist.

If you logged in to a remote machine in step 1, enter at the system prompt the command:

```
exit
```

Running the Installation Program

The installation program is called *install.xr*.

Starting *install.xr*

- If you downloaded the program from a tape, start *install.xr* by entering the command below.

```
install.xr [-v] [-log]
```

- If you are running from a CD-ROM drive, type:

```
<cdrom_dir>/install.xr [-v] [-log]
```

It is recommended that you use the *-v* or *-log* option while running the installation program. The *-v* option (verbose mode) provides additional information during installation about the operations being performed. The *-log* option creates a log of the current installation session, including all *install.xr* screens and user responses. This information may be helpful later in identifying difficulties with installation.

If you need any additional information about the command line options available with *install.xr*, run one of the following help utilities:

- If you downloaded the program from a tape, type:

```
install.xr -help
```

- If you are running from the CD-ROM, type:

```
<cdrom_dir>/install.xr -help
```

Once the installation program is invoked, the main menu is displayed.

Installing XRunner

To perform a complete installation of XRunner, select Option 6 from the installation menu. Note, however, that you can also choose to run specific stages of the installation process separately. For example, you may download the XRunner files from the distribution media in one session, and then install the license manager and licenses in a later session.

- 1 Define the XRunner installation directory.

If you want the XRunner files to be downloaded to a directory other than the one you specified earlier (when you downloaded `install.xr`), enter the new directory. Otherwise, press Enter.

- 2 View/modify the media setting.

This is a confirmation step, to ensure that the specified directory and device name are correct, and that you have sufficient disk space.

- If you are downloading from a CD-ROM:

Choose yes to the question, “Are you installing from a CD-ROM?”

In order to download XRunner from the CD-ROM, you must specify the location and the device drive of your media. The current settings are those you entered earlier, when you downloaded `install.xr`. Note that the CD-ROM settings vary, depending on your platform.

- If you are downloading from a tape:

Choose no to the question, “Are you installing from a CD-ROM?”

In order to download XRunner from the tape, you must specify the location and the device drive of your media. The current settings are those you entered earlier, when you downloaded `install.xr`. Note that the tape settings vary, depending on your platform.

Press ENTER when the correct settings are displayed.

3 Load XRunner from the tape or CD-ROM.

Choose “yes” to the question “OK to download XRunner?” and press ENTER. The downloading of XRunner files will begin; this process will take several minutes.

Enter the serial number when prompted. This number is listed on your installation tape.

The installation program notifies you that XRunner has been successfully downloaded.

Note: If at the end of the downloading process you receive a message notifying you that “A mismatch was found between the expected directory structure and the one downloaded from tape,” do not continue the installation process. You will not be able to complete the installation properly.

In such a case, quit the installation and examine the *install.xr.diff.tmp* file (in the installation directory) that lists the differences. If you cannot detect the reason for the mismatch, see the section, “xr_verify” at the end of this chapter.

4 Install the XRunner license manager.

Confirm the default startup file, and the directory in which the license manager will be installed. Note that the startup file is different for each platform.

5 Install a new license

The *install.xr* program provides you with a server code identifying the machine that it is running on. This is also the machine on which the license manager will run. Make note of this code, as well as your XRunner serial number, and call Mercury Interactive to receive your license key.

Select the license type to be installed.

Several types of licenses are available. Each license type must be installed separately. After installing your licenses, select quit. This will return you to the main installation menu. Select quit.

6 Run a full XRunner Installation.

Choose the Full XRunner Installation option to perform the first five steps listed in the menu.

7 Obtain additional information.

Select Mercury support to display the contact details of your closest Mercury Interactive sales and support office. Select Check Support for Your Server to clarify XRunner support for your server.

Note: Before you can start working, you must configure the XRunner environment. This is described in detail in the next chapter, “Setting Up the Environment.” It is also recommended that you consult the Product Release Notes.

Checking Your XRunner Installation

The XRunner installation program includes `xr_verify`, a self-help installation utility, located in the `/bin` subdirectory of your installation directory. If you experience difficulties with the XRunner installation, you may access `xr_verify` by entering the following command at the prompt:

```
xr_verify -h
```

Troubleshooting

The following paragraphs describe some common problems that may arise during installation:

Various problems occur when downloading XRunner from a tape located on a remote machine.

Such problems are almost always related to UNIX permissions. Note carefully the error message printed by `install.xr`. Make sure that you are a “trusted user” on the remote machine, and that the machine you are running on is a “trusted host.” If you are working as a superuser, verify that you can execute a remote command as root.

In addition, make sure that your shell startup files do not cause anything to be printed when you open a new shell. Otherwise, you will encounter problems during installation, if the output from the new shell gets mixed in with the output of the `dd` command applied remotely by XRunner. (The result will be probably be a “directory checksum” error.)

A mismatch is found between the expected XRunner directory structure and the actual files downloaded from tape.

Check that you have enough disk space.

When you try to check whether your X Server is supported (option 8 in the installation menu), the program does not detect the X Server.

The `DISPLAY` environment variable is probably undefined. You may safely ignore the message and continue with the installation. This may also happen if the `xdpyinfo` program cannot be found in your path.

The process of downloading the files from the installation tape fails.

First check that your device driver is compatible with the installation tape. See your system administrator for assistance.

If the device driver is appropriate, the block factor of your device may be different from the one on which the tape was created. For further assistance, see the section, “Checking Your XRunner Installation.”

Setting Up the Environment

This chapter describes the environment setup that you need in order to run XRunner.

Note: If your XRunner package includes a Context Sensitive license, you must perform additional steps to prepare your application for Context Sensitive Testing. Once you complete the steps described in this chapter, refer to the chapter “Linking the Context Sensitive Library.”

Installing Required Extensions

To enable record and replay, XRunner uses either the XTrap or the XTestExtension1 extensions. The only exception to this is if you are using OpenWindows under SunOS. In such a case, you do not need these extensions and you can start the server by typing the command:

```
openwin -defeateventsecurity -noauth
```

For all other cases, you need to install either the XTrap or the XTestExtension1 extensions. The installation instructions depend on whether you are using SunOS or Solaris.

SunOS

- 1 Find out whether your server contains one of the required extensions by entering the command:

```
xdpyinfo
```

You can also run the *XlistX* utility provided under `$M_ROOT/bin`.

If either DEC-XTRAP or XTestExtension1 appear in the list, you can move on to the next section, “Setting Environment Variables.”

- 2 If your server does not contain either DEC-XTRAP or XTestExtension1, you need to execute Mercury Interactive’s enhanced version of the server, which contains the necessary extension.

The enhanced version of the MIT X11R4 server is called *MIXsun* and the enhanced version of MIT X11R5 is called *MIXTrap*. Both are located under the `#{M_ROOT}/bin` directory, where `#{M_ROOT}` indicates the installation directory of XRunner.

- 3 To run Mercury Interactive’s enhanced server, you have to modify the script that starts your X server. In many cases this script is called *startx*.

Search for the command “xinit” in the start-up script. Change this command to:

```
xinit -- $M_ROOT/bin/MIXsun      (for X11 R4 server)
```

```
xinit -- $M_ROOT/bin/MIXTrap    (for X11 R5 server)
```

- 4 Note that Mercury Interactive’s R5 server (*MIXTrap*) assumes that the server installation directory is `/usr/X11R5`. If this isn’t the case in your environment, you have to add several lines to your start-up script.

Suppose your server installation directory is `/home/X/R5`. In such a case you would add the following lines to your script:

```
FP=/home/X/R5/lib/X11/fonts
FONTS=$FP/misc,$FP/75dpi,$FP/100dpi,$FP/Speedo
RGB=/home/X/R5/lib/X11/rgb
```

You should also modify the “xinit” command in the start-up script to:

```
xinit -- $M_ROOT/bin/MIXTrap -fp $FONTS -co $RGB
```

Solaris

If you are working under Solaris 2.3 or higher with OpenWindows 3.3, you need the XTrap extension.

- 1 Insert the following lines into your *OWconfig* file. (The *OWconfig* file is usually located under `/usr/openwin/server/etc`.)


```
class="XEXTENSION" name="DEC-XTRAP"
sharedObject="libxtrapext.so"
initFunc="DEC_XTRAPInit"
preLoad="YES";
```

- 2 Copy `$M_ROOT/bin/libxtrapext.so` to `/usr/openwin/server/modules`.
- 3 Restart the Xserver.

Alternatively (on Solaris 2.3 only), you can use the *OWconfig* file which is provided with XRunner. Enter the following at the command line, before you start the server (it is recommended that you insert this line in your *.cshrc* file):

```
setenv OW_CONFIG_PATH $M_ROOT/bin
```

Setting Environment Variables

Open the window shell from which you intend to invoke XRunner. Define the environment variables for this shell as described below. You can also define the variables in a global location such as *.cshrc*.

- The environment variable `$M_ROOT` points to the XRunner installation directory. For example, if XRunner is installed in the directory `/tools/QA`, then you should enter the following at the command line:

```
setenv M_ROOT /tools/QA/xrunner
```

Note: If you work with IXI Motif, do not define `$M_ROOT` in *.cshrc* or another global location. Define `$M_ROOT` only in the shell from which you are running XRunner or an application-under-test linked to Mercury Interactive's Context Sensitive library (for more information on linking, refer to the chapter "Linking the Context Sensitive Library.")

- The XRunner binaries directory is specified by your path. This is the bin directory under `M_ROOT`. You add `$M_ROOT/bin` to your path by entering:

```
set path = ($M_ROOT/bin $path)
```

- The environment variable `MERCURY_ELMHOST` is set to the name of the machine on which the XRunner license manager is installed. For

example, if the license manager was installed on the machine named venus, set MERCURY_ELMHOST to venus. Note that the MERCURY_ELMHOST environment variable is only required for large networks. In most cases, XRunner will be able to automatically locate its license manager using network broadcasts.

- If you are testing an XView or OLIT application, you must set the MIC_TOOLKIT environment variable to XView or OLIT, accordingly. For example, in the window shell in which you intend to invoke XRunner, type

```
setenv MIC_TOOLKIT XView
```

- For Solaris/x86, to enable the record and replay extensions, you must set the following environment variable:

```
setenv XRUN_TEST_EXT XTEST_XTRAP
```

Checking .Xdefaults File Settings

Check the *.Xdefaults* file for the following recommended settings. If they do not exist, add them to the file. These settings should be entered in the *.Xdefaults* file in the home directory of each XRunner user, or in the system-wide *app-defaults* or *.Xdefaults* file.

Note that when using monochrome X displays, you must add the following setting:

```
xrunner*topShadowColor:black
```

After performing the necessary changes, enter the command:

```
xrdb ~/.Xdefaults
```

OPEN LOOK

OpenWindows.MultiClickTimeout: 8

It is recommended that multiple click timeout (used for double clicks) be set to no less than 0.8 seconds, to ensure reliable replay under different workloads.

OpenWindows.Beep: notices

To avoid unwanted beeps when using XRunner softkeys, do not set this to option to “always.”

OSF/MOTIF**Mwm*doubleClickTime: 800**

It is recommended that multiple click timeout (used for double-clicks) be set to no less than 0.8 seconds, to ensure reliable replay under different workloads.

Mwm*interactivePlacement: False

Setting interactive placement to *False* reduces the number of TSL statements generated when opening new windows.

Mwm*clientAutoPlace: False

Setting auto-place to *False* causes pop-up windows to come up in the same location each time. If this parameter is set to *True*, pop-up windows will appear in different locations, and XRunner will have to move these windows to their correct location. As a result, test execution time will be somewhat slower.

Defining Softkeys

XRunner uses softkeys to invoke different commands. These command softkey assignments are configurable. You can find a table of the default bindings below. Using this table, check whether any of XRunner’s default softkey bindings interfere with a softkey your application uses. If so, reconfigure the command to another softkey, using the system parameters listed in the table.

For example, assume the CHECK WINDOW function is bound to the F2 key on your keyboard and that this interferes with a function used already by your application.

To change the default binding of the check window function, edit the *xrunner.cfg* file and locate the following line:

```
XR_SOFT_CHECK_WINDOW = F2
```

Change F2 to the name of another key which is unlikely to interfere with other applications. For example:

`XR_SOFT_CHECK_WINDOW = F10`

Once you have XRunner up and running, you can modify softkey bindings from the Configuration form. For more details, Chapter 33, “Changing System Defaults” in the *XRunner User’s Guide*.

| Command | Softkey | Configuration Parameter |
|---------------------------------|-----------|------------------------------------|
| RECORD | F5 | XR_SOFT_RECORD |
| RUN FROM ARROW | F8 | XR_SOFT_ANIMATE |
| STEP | F7 | XR_SOFT_STEP |
| STEP INTO | Ctrl_L F7 | XR_SOFT_STEP_INT0 |
| ABORT (STOP) | F6 | XR_SOFT_STOP |
| PAUSE | F8 | XR_SOFT_PAUSE |
| MARK LOCATOR | Ctrl_L F5 | XR_SOFT_MARKLOCATOR |
| GET TEXT | Ctrl_L F1 | XR_SOFT_GET_TEXT |
| CHECK WINDOW | F2 | XR_SOFT_CHECK_WINDOW |
| CHECK WINDOW (AREA) | Ctrl_L F2 | XR_SOFT_CHECK_PARTIAL_WINDOW |
| WAIT WINDOW | F4 | XR_SOFT_WAIT_WINDOW |
| WAIT WINDOW (AREA) | Ctrl_L F4 | XR_SOFT_WAIT_PARTIAL_WINDOW |
| WAIT REDRAW | F3 | XR_SOFT_WAIT_REDRAW |
| WAIT REDRAW PARTIAL WINDOW | Ctrl_L F3 | XR_SOFT_WAIT_REDRAW_PARTIAL_WINDOW |
| CHECK GUI (CHECKLIST) | Alt_L F2 | XR_SOFT_CHECK_GUI |
| INSERT FUNCTION (FROM LIST) | F9 | XR_SOFT_GEN_FUNC_CALL |
| INSERT FUNCTION (OBJECT/WINDOW) | Ctrl_L F9 | XR_SOFT_OBJ_FUNC_CALL |

Modifying the Mouse Configuration

If you work with Solaris/x86 on a PC, you may have to modify the mouse configuration. XRunner comes configured for a three-button mouse. To use a two-button mouse, make the following modifications:

- ▶ In the configuration file `xrunner.cfg`, change the value of the `XR_INP_MKEYS` configuration variable. Change the line

```
0x02 Middle S_Middle
```

```
to
```

```
0x02 Right S_Right
```

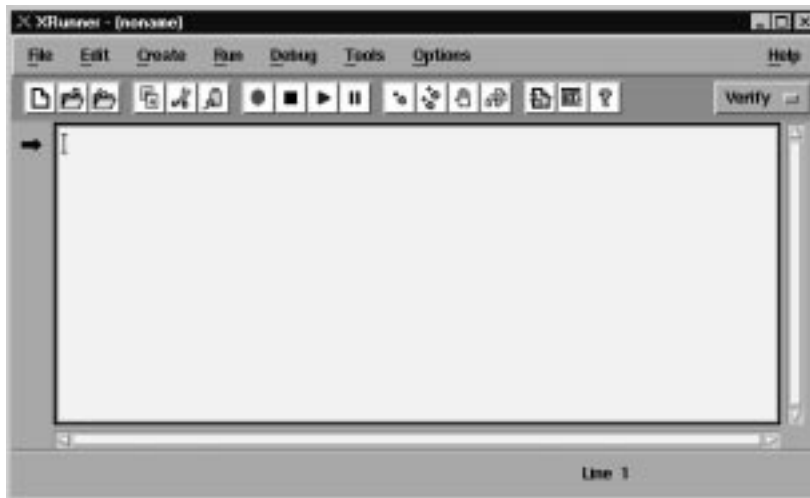
- ▶ In the `csmod` or `csmod_ol` script (located under the `$M_ROOT/lib` subdirectory), switch the values of the public constants `RIGHT` and `MIDDLE`.

Starting XRunner

Start XRunner from your working directory by entering the command:

```
xrun &
```

The XRunner window is displayed.



Troubleshooting

The following paragraphs present explanations and solutions to problems that may arise when starting XRunner:

XRunner refuses to come up and the message “system error” is displayed.

XRunner could not connect to its License Manager. Check that the MERCURY_ELMHOST environment variable points to the name of your License Manager machine and that the License Manager is running.

Using XRunner’s softkeys has an adverse effect on xterm.

Try assigning XRunner commands to different softkeys.

The XRunner License Manager

The XRunner license manager supervises the allocation and use of XRunner licenses available to your staff. Five types of licenses are offered:

- *Development* provides full test creation and replay capabilities
- *Execution* supports test replay only
- *Text Recognition* allows you to use features that locate and read text on the screen.
- *Context Sensitive* allows you to record high-level test scripts.
- *Background Testing* allows you to run test in background mode.

The XRunner license manager uses a floating license scheme: licensing is handled according to the number of users on the network, and is not linked to one or more specific machines. XRunner also supports the allocation of reserved licenses for specific users, user groups, and machines.

You may install the license manager on multiple servers, using the redundant license key. This will allow you to maintain one or more backup file servers with XRunner license control. For details, see “xrelmd—License Manager Daemon.”

The XRunner license manager incorporates four programs:

- *xr_elmadmin*, the license installation and general administration program
- *xrelmd*, the license manager daemon
- *xr_elmreport*, the license manager report generator
- *xr_elmdecode*, the license manager code and key decoder

These four programs are installed on your system by the XRunner installation program, and are stored in the `$M_ROOT/elm` directory that it generates. The XRunner installation program also automatically invokes the license manager administration program. Before doing so, it updates the

/etc/rc file (the default—during installation, you can specify an alternate file name), inserting a line which causes the license manager program to be invoked with the appropriate options the next time the machine is booted.

The following paragraphs describe these programs in greater detail.

xr_elmadmin—License Manager Administration Program

The *xr_elmadmin* license administration program is used by the XRunner installation program (*install.xr*) to install newly purchased license keys. If you are an experienced XRunner user, and you would like to store the keys to a new license in an alternative location, you can run the *xr_elmadmin* program manually. This program also performs a number of additional administrative functions. The command and option syntax for this program is:

```
xr_elmadmin [ -b cid -c -d -e keydir -h -i -k -l -q -r # -s -v # -V [A|B] -x -z ]  
[ features | alias | feature =alias,... ]
```

Note that those options marked with an asterisk may only be invoked by the owner of the current license server process, the super-user, or any member of the specified group.

-b *cid* *

Buries client. Returns all licenses used by XRunner whose client ID is *cid*, and marks this XRunner as dead. The client ID (CID) may be determined with "*xr_elmadmin -l*".

A buried XRunner will continue to run, but without licenses. All subsequent functions from XRunner that communicate with the license server will return an error.

-c

Creates a license key file. *xr_elmadmin* will print a server code which you give to Mercury Interactive which, in turn, will supply a key. Note that a single application may have one or more keys, one for each feature. In the case of XRunner, Text Recognition and Context Sensitive features require additional licenses.

If *feature=alias* appears as the final argument (in the program command line), then this name will be installed as an alias for this feature. One typically creates an alias for numeric features. Subsequent commands such as "*xr_elmadmin -l*" will print the alias instead of the feature name. A list of definitions may also appear on the command line. Upon decoding the key

input, the feature name gleaned from the key will be matched against the list of definitions. If a match is found, the name following the sign is taken as the feature alias. This provides a convenient way of specifying feature aliases within shell scripts. For example,

```
xr_elmadmin -c -n 01=execution,02=text-recognition,03=context-sensitive
```

will cause the alias execution to be selected, if the key decodes with the feature 01.

The key file is placed in the key directory `/usr/lib/xrlm` on UNIX systems, unless the `-e` option described below is used. In either case, the key directory is not created, but should already exist. Here are some examples:

1. Create a standard (alphanumeric) key:

```
xr_elmadmin -c -e /usr/lib/xrlm
```

2. Create a numeric-only key, with alias development:

```
xr_elmadmin -c -n -e /usr/lib/xrlm development
```

- d Sends an `ELMC_DUMP` command to the daemon `xrelmd`. This command causes `xrelmd` to print the contents of the `xrelmd` database to the log file. It is used mostly for testing and debugging.
- e This option is meaningful only when the option `-c`, described above, is used. By default, license keys are placed in the directory `/usr/lib/xrlm` on UNIX systems. The `-e` option specifies a different key directory. Thus,


```
xr_elmadmin -e /local/mercury -c
```

 will cause the license key to be created in the directory `/local/mercury` on a UNIX system. Please note that this directory must already exist—`xr_elmadmin` does not create it.
- h Expanded version of the `-l` listing, including the hold time of each XRunner and the shared license count. The hold time is the period that the licenses will be held (reserved) for the user after XRunner exits. If the user starts the application again before the hold period ends, the held licenses will be assigned to him. The hold time is specified in the resource file. If XRunner is held, column number one of the listing will be flagged with an "H". The shared license count is the number of XRunners that are currently sharing the license for the indicated feature. If XRunner has one or more shared licenses, column one of the listing will be flagged with an "S".
- i Prints the release number of `xr_elmadmin` and the release of the library with which it was compiled.

- k * Kills the current run of the license manager daemon (*xrelmd*) program. Note that this option may be issued only by the super-user.
- l Lists the current users and outstanding licenses available for all features known to all license managers on the network. Known features are those features used since the daemon was started (usually the last time the system was rebooted). If features are given on the command line, the listing is restricted to the features named. The following is a sample listing:

| | CID | LID | User | Feature | Group | Started |
|---|-----|-----|--|-----------|-------|--------------|
| | 1 | 1 | charlie@venus,gold:0 | XRunner-D | - | Dec 12 16:05 |
| H | 4 | 2 | jennifer@elan,mel:0 | XRunner-E | - | Dec 12 14:24 |
| | 2 | 3 | charlie@venus,gold:0 | XRunner-C | - | Dec 12 16:05 |
| | 3 | 4 | charlie@venus,gold:0 | XRunner-T | - | Dec 12 16:05 |
| | | | XRunner-D [00]: 4 licenses, 0 reserved, 1 in use | | | |
| | | | XRunner-E [01]: 5 licenses, 0 reserved, 1 in use | | | |
| | | | XRunner-C [02]: 4 licenses, 0 reserved, 1 in use | | | |
| | | | XRunner-T [03]: 4 licenses, 0 reserved, 1 in use | | | |

An "H" in column one indicates that this client is a held client. Such a client is one that has exited, but whose licenses are held for a certain period. The hold period may also be listed with the -h option below.

A "D" in column one indicates that this client is connected but disabled (redundant servers mode but there are too few servers running currently.)

An "S" in column one indicates that this client has a shared license for the indicated feature. Shared license groups share the same LID.

The next field, CID, is the client ID. Each client application will be assigned a unique ID. The CID may be used with the -b option described above.

The LID is the license ID. These are unique license numbers. Each client will normally have a unique LID for each licensed feature, except when the license is shared. If clients share a license, they share the LID.

The user is displayed as *user@host,display* where *display* is the value of the X-11 DISPLAY when the application was invoked. The *display* may be missing for non X-11 applications.

The `-l` option scans all servers on the network, as well any others defined by the MERCURY_ELMHOST environment variable.

- `-q` Query. Prints various environment information for debugging. Output includes the current host name, its IP address, the name of the MERCURY_ELMHOST environment variable and its current setting, the current port, the license service name, the Vendor ID, and the hostcode value.
- `-r #` Ready-key mode, for use with the `-c` option. Depending on the parameter `#`, does not print the code(s) or prompt for the key: If the number `#` is positive, then does not print the server code(s); only prompts for the server key. The `#` in this case is the number of servers that will be running, usually 1. (It will be greater than 1 only when backup servers are used.) Thus, "`-r 1`" prompts for a key for a single server configuration.

If the number `#` is negative, then no key prompt will be issued and only the number of codes indicated will be printed. Thus, "`-r -1`" prints the current server code but does not prompt for a key.

- `-s` Finds all license servers using server address resolution and prints their names. If a feature appears on the command line, the hosts responding with this feature available are printed.
- `-V version` Sets encryption version to the argument, a single character A though E.
- `-v # *` Changes the license manager log file verbose level to `#`. This may be used to change the level after the daemon has started. See the `xrelmd` documentation for the list of levels and more details.
- `-x` Inhibits copyright notice.
- `-z *` Zeroes the log file. One method to clean up the log file every evening on a UNIX system is:

```
cp elm.log elm.log.old
```

```
MERCURY_ELMHOST=somehost
xr_elmadmin -z
```

where `somehost` is the host name where the `xrelmd` is running. Alternately, the `-m` option of `xrelmd` may be used to automatically keep the log file under control.

Note that truncating the log file by other means would be ineffectual since `xrelmd` keeps the log file open at all times. The next `xrelmd` log file write would simply use the old file pointer, filling the file to the original length. Therefore the `-z` option of `xr_elmadmin` or the `-m` option of `xrelmd` must be used to truncate the file.

xrelmd—License Manager Daemon

When you install XRunner, you must designate the machine on which the license manager will be installed. This machine will run the license manager daemon. In order to be able to invoke XRunner on any other station, the daemon must first be active. If you want to maintain a backup system with XRunner license control, you may run the daemon on multiple servers by using a redundant license key (as discussed below).

The daemon handles requests from network users to invoke XRunner by issuing and returning licenses. It also supplies the data required by the `xr_elmadmin` program to display the XRunner licenses currently in use.

The daemon continually updates a log file (stored by default under the name `/usr/adm/xr_elm.log`) that documents actual use of XRunner licenses, as well as all denied access. This log file also serves as the basis of a load analysis that is performed when you generate a license manager report.

The daemon is activated by running the `xrelmd` program. During installation, a line is written into the system startup script which automatically invokes the daemon. If you would like to change the default invocation line, you can edit the startup file.

The command and option syntax for the daemon program is:

```
xrelmd [ -D -e path -f -i -l file -m size -n -p # -r file -s # -v # -z # ]
```

- D Debug mode: 1) Causes the log file to be flushed after each message. Normally it is flushed after each packet is processed. There can be several messages per packet, depending on the verbosity (`-v`) level. 2) Allows SIGQUIT to take the default action of creating a core file and exiting. 3) In background mode, does not perform `chdir`. Normally, `xrelmd` runs with the current working directory set to `.`. The message "DEBUG mode is on" will be printed to the log file if `-D` is used.

Please also see the `-v` flag below for details on changing the verbosity level of `xrelmd`.

- `-e path` Specify the key directory, or set of directories, and load all keys found in the directories (optional).

On UNIX systems, directories are specified as a list of directory names separated by a colon (:). For example:

```
xrelmd -e /local/elan:/local/express:/usr/lib/xrlm
```

Note that there is no default key path. Therefore, the `-e` option must be specified.
- `-f` Normally on UNIX, `xrelmd` will run in the background (it forks a child process, detaches from the terminal, and the parent exits). The `-f` option inhibits this, and causes `xrelmd` to run in the foreground. You may, for example, wish to run `xrelmd` from `inittab`, instead of `/etc/rc.local`, with the `-f` option.
- `-i` Prints the release number, the vendor ID, and exits. The vendor ID provides a way to verify that the same vendor salt is being used among all the commands. The vendor ID should be the same for `xrelmd`, `xr_elmadmin`, `elmkey`, etc.
- `-l file` Creates a log file named *file* and writes all relevant information to that file. If the file already exists, `xrelmd` will append to the end of the file. Note that *file* must be a complete path name. Use of a log file is highly recommended as it provides a way of tracking problems if they occur. A log file must be enabled for `xr_elmreport` to provide reporting information. Please see also the `-m` option, below.
- `-m size` Limit the log file to the size specified. With this option, the log file is “self maintained” by `xrelmd` such that `xrelmd` limits the size that the log file may reach. When this size is reached, the log file named *file* is moved to *file.old* on UNIX systems and the current log file is truncated.

The *size* is an integer or floating point number. The default units are bytes. It may also be specified in kilobytes by suffixing a “k” or megabytes by suffixing an “m” to the size. For efficiency, the log file size is actually checked only every 100 lines, so it might exceed the limit slightly before it is backed up and truncated.
- `-n` Non-networked (stand-alone) operation. (Not applicable for Windows). This tells `xrelmd` that there is no network and to use local communication only. (This will force `xrelmd` to use System V native interprocess communication on UNIX.) This option is applicable to small systems such as SCO and Inter-

active UNIX without TCP/IP installed. When `-n` is used, you must set `MERCURY_ELMHOST` to "local".

Note that on System V, the default interprocess communication queue is quite small; adequate for about two client processes talking to `xrelmd` concurrently. `xrelmd` will attempt to increase the queue size, but since only root can raise the IPC queue size, it must be run as super-user to be effective.

`-p #` Cause `xrelmd` to use the port address `#` instead of the default or one specified by the system service. One might use this option if there were a port address conflict and the user did not have permission to alter the system port services file.

`-r file` Designates the License Manager resource file to be used when issuing licenses to users or client machines. Each line of this file has the following syntax:

```
license_type:group:client1,...,clientn:k:m
```

This translates to: `k` licenses of the specified `license_type` are to be reserved for the named group composed of the enumerated clients. The licenses will be reserved for `m` seconds after XRunner has been exited. Note the following points:

- The `license_type` specified by each line will be either 00 (XRunner-D), 01 (XRunner-E), 02 (XRunner-T), 03 (XRunner-C) or 07 (Background testing).
- The named group is an arbitrary name (up to 20 characters long) assigned to a group of clients; it bears no connection to the UNIX system group names stored in `/etc/group`.
- The clients comprising these groups are enumerated by their login names or, in the case of hosts, by their host name. Host names should be preceded by the `@` character.
- The number of licenses (`k`) reserved for the group concludes the line entry.
- A line beginning with the `#` character is recognized as a comment and is ignored.

`-s #` Sets the startup time to `#` in seconds. This is the period that `xrelmd` waits and listens for XRunner to reconnect after a possible crash. The default is 180 seconds (3 minutes) If you are not on a network, this may be set to zero.

-v # Selects the verbosity of the messages printed to the log file. When a level is selected, all levels less than or equal to the level are printed. Levels 5 and above are considered debugging levels. *xrelmd* starts at level 3. The levels are:

| Level | Messages written to log file |
|-------|-----------------------------------|
| 1 | Only error messages |
| 2 | License failures |
| 3 | License activity |
| 4 | Client connects and disconnects |
| 5 | Message for each packet received |
| 6 | Message for each packet sent |
| 7 | Additional client information |
| 8 | Data about key and resource files |
| 9 | Other details |

-z # Set the zombie to # seconds. If XRunner is not heard from within this time period, it is declared dead and its licenses are returned. The default zombie interval is 3 minutes. Note that *xrelmd* cleans up zombies only every 60 seconds, so it may take up to 1 minute more for the licenses to be returned.

Creating Backup License Managers

You may run the license server *xrelmd* on multiple servers by using a redundant license key. To create a redundant license key:

1 Install the license manager.

Using *install.xr*, individually install the License Manager on each of the redundant servers. During the installation on each machine, you will be prompted for the name of the key license directory. The key must be readable by all redundant license servers. This can be ensured by:

- placing the key in a file system shared by all machines, or
- locally copying the key to a location on each server.

For the sake of convenience, it is preferable to place the key in the same directory path on each system. If this is not practical, use the `-e` option of `xrelmd` on each server to specify the key location.

2 Name redundant servers.

Verify that all redundant servers are named in the resource file. Note that this step is not performed automatically by the installation program. You must edit the resource file manually by placing lines of the form `%SERVER hostname` in the resource file, where `hostname` is the name of one of the hosts. There should be one line for each redundant server.

3 Start the license manager with the `-r` option.

Verify that, in the startup file for each server, the License Manager is activated using the `-r` option while designating the appropriate resource file.

xr_elmreport—Report Generator

The `xr_elmreport` program generates a report of XRunner license activity based on the log file. To run this program, enter the following command at the system prompt:

```
xr_elmreport [ -d daterange -f feature -h -i -l -t -u category -w # ]  
[ logfiles | logfiledirs ]...
```

For some examples of reports, see below.

`-d` *daterange* The *daterange* is in the format [*from* - *to* | *from*]. Limit reporting for dates between *from* and *to* inclusive. The *from* and *to* dates may be specified in one of two formats: Feb 3 12:30:00 or 2/3 12:30:00. In the first format you specify the first three letters of the month. In the second format, the month number, followed by a slash is used. For example: `xr_elmreport -d "Mar 15 - Apr 15 5:30"` or `xr_elmreport -d "3/15 - 4/15 5:30"`

Note that in either format, only the month is required. The day, hour, minutes, and seconds will default to zero if missing. If the hyphen and to date are missing, the report will run to the last date.

`-f` *feature* Lists only activity for the feature named. Multiple `-f` options may be used to limit display to several features.

- h Print a usage histogram instead of numeric data (see also -w option below).
- i Prints the release number of `xr_elmreport` and the release of the library with which it was compiled, then exits.
- l License listing. Displays a summary of licenses available along with the number of tokens of expiration dates, if any. No other options are meaningful with -l.
- t *timeunit* Displays total license activity for each *timeunit*, where *timeunit* may be month, day, hour, minute, second. (A unique abbreviation, such as "min", is ok.) For example:


```
xr_elmreport -d "3/15 - 4/15 5:30" -t hour
```

 If the -t option is not specified, a sun total only is printed.
- u *category* Details report by each user, specified by category. The category may contain one or more key letters "u", "h", and "d" for user name, host, and display, respectively. Key letters may be combined in any combination of one to three letters. For example, -uu details per user name only; -uh details per host name only; and -uuh provides the most detail, with results displayed for each distinct user@host.
- w # Specifies a display width of # character units. The default is 78. This option is currently only effective with the -h (histogram) option.
- logfiles* The remaining arguments are zero or more license server log file names or directories. If any argument is a directory, then all log files in that directory are digested. `xr_elmreport` will silently skip files that are not license manager log files. If no files are specified on the command line, then the default `/usr/adm/elm.log` will be used.

Examples

`xr_elmreport` produces a report of licenses available or license activity from one or more license server log files. The default log file is `/usr/adm/elm.log` on UNIX systems. It is compatible with all older versions as well as with Flexlm. See `xr_elmadmin` (-l option) for a snapshot of current license activity. `xr_elmreport` can display its information in two formats: numerically

and as a histogram. Here is sample of numeric output from `xr_elmreport -t day`:

| Feature | Requests | Issued | Denied | %Denied | Total Time Used |
|---------------------|----------|--------|--------|---------|-------------------|
| Apr 14 XRunner-D | 20 | 20 | 0 | 0 | 0d 05h 02m 27s |
| Apr 14 XRunner-E | 50 | 45 | 5 | 10% | 0d 08h 09m 55s |
| Apr 16 XRunner-E | 35 | 35 | 0 | 0 | 0d 04h 23m 03s |

The Requests column shows the total number of license requests for the feature. The Issued column shows the number of licenses successfully issued. The Denied column shows the number denied, probably because the maximum number of licenses were already in use. The %-Denied column indicates the frequency of denied licenses out of the total number of requests. The final column shows the total time that the feature was checked-out by application(s).

The default report is detailed per feature. Reports can be further detailed by user, `user@host`, and/or `user@host+display`, if desired. Use of the `-h` option produces a histogram instead. Here is a sample histogram from `xr_elmreport -h -t day`:

```

Feature          * = issued, 0 = denied
May 15|XRunner-E ***** 46
May 15|XRunner-D *****ooo 25
    
```

When displayed as a histogram, the number of successful requests is displayed as a series of asterisks ("*") and denied requests as a series of the letter "o", followed by the actual number of successful requests. The graph is normalized for a screen width of 78, which may be adjusted with the `-w` option. A summary of licenses installed may also be obtained with the `-l` option.

xr_elmdecode—License Manager Code and Key Decoder

The *xr_elmdecode* program decodes license server codes and keys into readable form. For instance, *xr_elmdecode* can be used to determine the Host ID and/or CPU ID of a license. Note that the decoded code and host ID are printed in hexadecimal, followed by its meaning—the internet number and CPU ID PROM's low order low bytes.

xr_elmdecode is a diagnostic tool which may be used to decode license server codes and keys. This tool is frequently used by Mercury Interactive to determine the Host ID and/or CPU ID of a license. Non-numeric keys include the full host ID information. Numeric keys include a hashed version of the host ID information which we call the hostlock.

The command and option syntax for this program are:

xr_elmdecode [-i -q -c code -k key]

- i Print the release number.
- q Print the vendor ID and current hostcode.
- c Decode the code that follows.
- k Decode the key that follows.

Examples

An example of a server code decoding of a numeric code:

```
% xr_elmdecode -n -c "0074 7410 4751 1860 8" Code = 0074
7410 4751 1860 8 Decode = 3224006414163 = a5c1cf53 (hex) = (192.42.111.14
a3)
Numkey hostlock = 8079
```

The following is an example decoding of a numeric key:

```
% xr_elmdecode -n -k "4778 1085 3418 0847 80"
Key = 4778 1085 3418 0847 80
Feature = 99
Vendordata = (none)
Licenses = 5
Hostlock = 8079 (hashed)
Ndays = 1540 days past 1/1/89
```

Expires = Sun Apr 21 00:00:00 1999 (31.51 days from now)

The following is an example decoding of a non-numeric key:

```
% xr_elmdecode -k "wxabr94723abc6754"
```

```
Key = wxabr94723abc6754
```

```
Feature = avi
```

```
Vendordata = h1
```

```
Licenses = 5
```

```
Hostid = c12c6f33e5 (192.44.111.19 a5)
```

```
Expires = Fri Jul 26 01:00:00 1999 (89.37 days from now)
```

Linking the Context Sensitive Library

This chapter provides step-by-step instructions for preparing your application for Context Sensitive testing on Sun machines.

Since Context Sensitive testing is enabled by a library, you need to link your application with the appropriate version of this library before you begin. The linking procedure differs, depending on your operating system (SunOS or Solaris) and the toolkit on which your application is based (Motif, XView or OLIT).

This chapter contains procedures for the following types of applications:

- SunOS—Motif applications (versions 1.1/1.2/2.0)
- SunOS—XView applications (versions 3/3.2)
- SunOS—OLIT applications (version 3.1)
- Solaris—Motif applications (version 1.2/2.0)
- Solaris—XView applications (version 3.2)
- Solaris—OLIT applications (version 3.2)

Refer to the section that is appropriate for your application.

Linking with the `xr_analyze` Utility

To shortcut the step-by-step guide, you can run a self-help linking utility called `xr_analyze`. The utility resides in the `/bin` directory of your installation directory. To start `xr_analyze`, enter the following command at the prompt:

```
xr_analyze <Application Under Test executable>
```

SunOS—Motif Applications

This section is relevant if your application is running under SunOS and is based on the Motif widget set.

Overview

To link your application to Mercury Interactive's Context Sensitive library, you have three options. The option you select depends on how your application is linked with the X11 (Xlib), Xt (X Toolkit) and Xm (Motif) libraries:

| | |
|----------------|---|
| Dynamic | Use this option if your application is dynamically linked with the X11 (Xlib), Xt (X Toolkit), and Xm (Motif) libraries. |
| Dlopen | Use this option if your application is statically linked with any one of the three libraries. |
| Static | Use this option if your application is statically linked with either one of the three libraries, and you cannot use the Dlopen option. |

In the sections that follow, you will first determine how your application is linked. Then, you will choose the appropriate option that will link your application to Mercury Interactive's Context Sensitive library.

Determining Which Variant You Should Use

First, note that there are four legal combinations (variants) for working with the Xt and Xm libraries:

- Xt Release 4 (X11.4) with Xm Release 1.1 (Xm1.1)
- Xt Release 5 (X11.5) with Xm Release 1.1 (Xm1.1)
- Xt Release 5 (X11.5) with Xm Release 1.2 (Xm1.2)
- Xt Release 5 (X11.5) with Xm Release 2.0 (Xm2.0)

Mercury Interactive provides four versions of the Context Sensitive library that support these four variants. It is essential that your application is linked to the correct Mercury Interactive library. In the instructions that follow, *<variant>* refers to one of four locations in which the Mercury Interactive libraries reside:

- \$M_ROOT/arch/X11.4/Xm1.1
- \$M_ROOT/arch/X11.5/Xm1.1
- \$M_ROOT/arch/X11.5/Xm1.2
- \$M_ROOT/arch/X11.5/Xm 2.0

Before you proceed, find out which versions of the Xt library (X11.4 or X11.5), and the Xm library (Xm1.1, Xm1.1 or Xm 2.0) are used by your application. Determine accordingly which library path to use.

Determining How Your Application Is Linked

- 1 Determine whether your application is dynamically linked with the Motif, X Toolkit, and X11 Libraries by entering the following command:

ldd *<Application Under Test Executable>*

The result of this command will be a list of all dynamic libraries with which your application is currently linked. For example:

```
-lMrm.4 => /usr/lib/X11/libMrm.so.4.0
-IXm.4 => /usr/lib/X11/libXm.so.4.0           # Motif library
-IXt.4 => /usr/lib/X11/libXt.so.4.0         # XToolkit library
-IX11.4 => /usr/lib/X11/libX11.so.4.3      # X11 library
-lc.1 => /usr/lib/libc.so.1.8
-ldl.1 => /usr/lib/libdl.so.1.0
```

- 2 If **all** three libraries: -lXm, -lXt and -lX11 appear in this list (as in the above example), your application is dynamically linked with these libraries. In such a case, use the **Dynamic** option to link your application with the Context Sensitive library. Follow the instructions in the next section, “Dynamic Option.”

If **any one** of the -lXm, -lXt or -lX11 libraries does not appear in the list, your application is statically linked with that library. In such a case, you should use either the **Dlopen** option or the **Static** option in order to link your application with the Context Sensitive library.

Dynamic Option

Using the Dynamic option, you dynamically link Mercury Interactive's Context Sensitive library to your application. You create two symbolic links between the Mercury Interactive library and the original Xt library:

- Between Mercury Interactive's *liborig.so.4.0* and the original *libXt.so* library
 - Between Mercury Interactive's *libXt.sa.4.0* and the original *libXt.sa* file
- 1 Locate the original dynamic Xt library in your system by repeating the `ldd` command as shown in step 1 of the previous section. Let's assume that in the `ldd` list you get the following line:

```
-lXt.4 => /usr/lib/X11/libXt.so.4.0
```

- 2 Enter the following commands:

```
In -s /usr/lib/X11/libXt.so.4.0 <variant>/liborig.so.4.0
```

```
In -s /usr/lib/X11/libXt.sa.4.0 <variant>/libXt.sa.4.0
```

For Xt release 5 applications, replace `libXt.sa.4.0` with `libXt.sa.4.10`.

The `<variant>` represents the path of the appropriate Mercury Interactive library (listed in the Overview). Make sure that the `<variant>` you use matches the Xt and Xm releases used by your application. For example, for an application based on Xt release 4 with Motif 1.1, enter the commands:

```
In -s /usr/lib/X11/libXt.so.4.0 $M_ROOT/arch/X11.4/Xm1.1/liborig.so.4.0
```

```
In -s /usr/lib/X11/libXt.sa.4.0 $M_ROOT/arch/X11.4/Xm1.1/libXt.sa.4.0
```

Note that once you create these links, you do not have to repeat this step again.

- 3 Change the dynamic library path to include the appropriate Mercury Interactive library, by entering the command:

```
setenv LD_LIBRARY_PATH <variant>:$LD_LIBRARY_PATH
```

The `<variant>` should be the same one you entered in step 1. For example, for an application based on Xt release 4 with Motif 1.1, enter the command:

```
setenv LD_LIBRARY_PATH
```

```
$M_ROOT/arch/X11.4/Xm1.1:$LD_LIBRARY_PATH
```

Note: It is recommended that you do not set `LD_LIBRARY_PATH` globally in your start-up file. You may have trouble invoking other applications.

- 4 Check that your application uses the Mercury Interactive library by issuing the `ldd` command again. This time you should receive the following entries for the Xt and liborig libraries:

```
-lMrm.4 => /usr/lib/X11/libMrm.so.4.0
-lXm.4 => /usr/lib/X11/libXm.so.4.0
-lXt.4 => <variant>/libXt.so.4.0
-lX11.4 => /usr/lib/X11/libX11.so.4.3
-lc.1 => /usr/lib/libc.so.1.8
-ldl.1 => /usr/lib/libdl.so.1.0
-lorig.4 => <variant>/liborig.so.4.0
```

- 5 Your application is now ready for Context Sensitive testing. Before you invoke your application, make sure that `$M_ROOT` is defined correctly.

When you start your application, you should receive the message:
 “Loaded Mercury Interactive’s Context Sensitive Library: <library version>.”

If this message does not appear, your application is not properly linked with the Mercury Interactive library. Refer to the “Troubleshooting” section at the end of the chapter.

Dlopen Option

Using the `Dlopen` option, you re-link your application with Mercury Interactive’s object file, by modifying your application’s makefile. During runtime, the object file dynamically opens the appropriate Context Sensitive library.

- 1 First, note that Mercury Interactive supplies three versions of the object file, appropriate for the three different ways your application might be compiled:

| | |
|-----------------------|--|
| <i>mic_xm.o</i> | for applications compiled with the <code>-pic</code> flag |
| <i>mic_xm_PIC.o</i> | for applications compiled with the <code>-PIC</code> flag |
| <i>mic_xm_nopic.o</i> | for applications compiled with neither <code>-pic</code> nor <code>-PIC</code> . |

Check how your application is linked and determine which object file you need. If you are not sure which object file to use, try first using *mic_xm.o*. If you encounter problems, refer to the Troubleshooting section.

- 2 Edit your application's makefile to include:

The `<variant>/<Mercury object file>` (must appear before other libraries).

The `-Bdynamic` flag (must appear at the end of the linking string).

- 3 The following example shows the relevant part of the makefile before and after the necessary modifications.

Before:

```
cc $(CFLAGS) -o <application_name> \
  <application_source>.o \
  -Bstatic \
  -L/usr/lib/X11
  -lMrm -lXm -lXt -lX11 \
```

After:

(Bold characters indicate the additions that were made)

```
cc $(CFLAGS) -o <application_name> \
  <application_source>.o \
  <variant>/<Mercury_object_file> \
  -Bstatic \
  -L/usr/lib/X11 \
  -lMrm -lXm -lXt -lX11 \
  -Bdynamic
```

The `<variant>` represents one of the three available Mercury Interactive libraries (listed in the Overview). Make sure that the `<variant>` you use matches the Xt and Xm releases used by your application. The `<Mercury_object_file>` represents one of the object files listed in step 1.

- 4 Your application is now ready for Context Sensitive Testing. Before you invoke your application, make sure that `$M_ROOT` is defined correctly.

When you invoke the application you will get a message:

“Loaded Mercury Interactive’s Context Sensitive Library: `<library version>`.”

If the message does not appear, your application is not properly linked with the Mercury Interactive library. Refer to the “Troubleshooting” section at the end of the chapter.

Static Option

Using the Static option you statically link your application with Mercury Interactive’s object file and library (*mic_xm_s.o* and *libmic_xm.a*), by modifying your application’s makefile.

It is recommended that you use this option only if you encounter problems using the Dlopen option, or if your application must be fully, statically linked.

- 1 Edit your application’s makefile to include:

<variant>/mic_xm_s.o (must appear before libraries).

<variant>/libmic_xm.a (must appear before other libraries).

- 2 The following example shows the relevant part of the makefile before and after the necessary modifications:

Before:

```
cc $(CFLAGS) -o <application_name> \
  <application_source>.o \
  -Bstatic\
  -L/usr/lib/X11 \
  -IMrm -IXm -IXt -IX11
```

After:

(Bold characters indicate the additions that were made)

```
cc $(CFLAGS) -o <application_name> \
  <application_source>.o \
  -Bstatic\
  <variant>/mic_xm_s.o \
  <variant>/libmic_xm.a \
  -L/usr/lib/X11 \
  -IMrm -IXm -IXt -IX11
```

The *<variant>* represents the path of the appropriate Mercury Interactive library (listed in the Overview). Make sure that the *<variant>* you use matches the Xt and Xm releases used by your application.

- 3 Your application is now ready for Context Sensitive testing. When you start your application, you should receive the message:
“Loaded Mercury Interactive’s Context Sensitive Library: *<library version>*.”

If this message does not appear, your application is not properly linked with the Mercury Interactive object file and library. Refer to the “Troubleshooting” section at the end of this chapter.

SunOS—XView Applications

This section is relevant if your application is running under SunOS and is based on the XView widget set.

Overview

To link your application to Mercury Interactive’s Context Sensitive library, you have two options. The option you select depends on how your application is linked with the X11 (Xlib) and XView libraries.

| | |
|----------------|---|
| Dynamic | Use this option if your application is dynamically linked with both the X11 (Xlib) and XView libraries. |
| Static | Use this option if your application is statically linked with either X11 or XView. |

In the section that follows, you will first determine how your application is linked. Then, you will choose the appropriate option that will link your application to Mercury Interactive’s Context Sensitive library.

Note: Before you invoke XRunner, set the MIC_TOOLKIT environment variable to XView. In the window shell in which you intend to invoke XRunner, type:

```
setenv MIC_TOOLKIT XView.
```

Determining which Variant You Should Use

First, note that there are two versions of the XView libraries:

- XView 3
- XView 3.2

Mercury Interactive provides two versions of the Context Sensitive library to support these two variants. It is essential that your application is linked to the correct Mercury Interactive library. In the instructions that follow, *<variant>* refers to one of two locations in which the Mercury Interactive libraries reside:

- \$M_ROOT/arch/XView3
- \$M_ROOT/arch/XView3.2

Before you proceed, find out which XView library is used by your application. Accordingly, determine which library path to use.

Determining How Your Application is Linked

- 1 Determine whether your application is dynamically linked with the XView and X11 Libraries by entering the following command:

```
ldd <Application Under Test Executable>
```

The result of this command will be a list of all dynamic libraries with which your application is currently linked. For example:

```
-lvt.1 => /tools/openwin_3/lib/libvt.so.1.1
-ice.0 => /tools/openwin_3/lib/libice.so.0.0
-ldl.1 => /usr/lib/libdl.so.1.0
-lxview.3 => /tools/openwin_3/lib/libxview.so.3.1# XView
-lX11.4 => /tools/openwin_3/lib/libX11.so.4.3 # X11
-lolgx.3 => /tools/openwin_3/lib/libolgx.so.3.1
-ldeskset.0 => /tools/openwin_3/lib/libdeskset.so.0.1
-lc.1 => /usr/lib/libc.so.1.8
```

- 2 If **both** the -lxview and -lX11 libraries appear in this list (as in the above example), your application is dynamically linked with these libraries. In such a case, use the Dynamic option to link your application with the Con-

text Sensitive library. Follow the instructions in the next section, “Dynamic Option.”

If **either** `-lxview` or `-lX11` does not appear in the list, your application is statically linked with that library. Skip the next section and follow the instructions under “Static Option.”

Dynamic Option

Using the Dynamic option, you dynamically link Mercury Interactive’s Context Sensitive library to your application. You create two symbolic links between the Mercury Interactive library and the original Xt library:

- Between Mercury Interactive’s *liborig.so.3.1* and the original *libxview.so* library
 - Between Mercury Interactive’s *libxview.sa.3.1* and the original *libxview.sa* library
- 1 Locate the original dynamic XView library in your system by repeating the `ldd` command as shown in step 1 of the previous section.

Let’s assume that in the `ldd` list you get the following line:

```
-lxview.3 => /tools/openwin_3/lib/libxview.so.3.1
```

- 2 Enter the following commands:

```
In -s /tools/openwin_3/lib/libxview.so.3.1 <variant>/liborig.so.3.1
```

```
In -s /tools/openwin_3/lib/libxview.sa.3.1 <variant>/libxview.sa.3.1
```

Note that once you create these links, you do not have to repeat this step again.

- 3 Change the dynamic library path to include the appropriate Mercury Interactive library, by entering the command:

```
setenv LD_LIBRARY_PATH <variant>:$LD_LIBRARY_PATH
```

- 4 Check that your application uses Mercury Interactive libraries by issuing the `ldd` command again. This time you should receive the following entries for the Xt and `liborig` libraries:

```
-lXt.1 => /tools/openwin_3/lib/libXt.so.1.1
```

```
-liborig.so.0 => /tools/openwin_3/lib/liborig.so.0.0
```

```
-ldl.1 => /usr/lib/libdl.so.1.0
-lxview.3 => <variant>/libxview.so.3.1
-IX11.4 => /tools/openwin_3/lib/libX11.so.4.3
-lolgx.3 => /tools/openwin_3/lib/libolgx.so.3.1
-ldeskset.0 => /tools/openwin_3/lib/libdeskset.so.0.1
-lc.1 => /usr/lib/libc.so.1.8
-lorig.3 => <variant>/liborig.so.3.1
```

- 5 Your application is now ready for Context Sensitive testing. When you start your application, you should receive the message:

“Loaded Mercury Interactive’s Context Sensitive Library: *<library version>*.”

If this message does not appear, your application is not properly linked with the Mercury Interactive library. Refer to the “Troubleshooting” section at the end of the chapter.

Static Option

Using the Static option you statically link your application with the library provided by Mercury Interactive (*libmic_xv.a*).

- 1 Edit your application’s makefile to include:
<variant>/libmic_xv.a (must appear before other libraries.)
- 2 The following example shows the relevant part of the makefile before and after the necessary modifications:

Before:

```
cc $(CFLAGS) -o <application_name> \  
  <application_source>.o \  
  -Bstatic\  
  -L/tools/openwin_3/lib \  
  -lxview -lolgx -lXext -lX11
```

After:

(Bold characters indicate the addition that was made)

```
cc $(CFLAGS) -o <application_name> \  
  <application_source>.o \  
  -Bstatic\  
  <variant>/libmic_xv.a \  
  -L/tools/openwin_3/lib \  
  -lxview -lolgx -lXext -lX11
```

```
-lxview -lolgx -lXext -lX11
```

- 3 Your application is now ready for Context Sensitive testing. When you start your application, you should receive the message:

“Loaded Mercury Interactive’s Context Sensitive Library: *<library version>*.”

If this message does not appear, your application is not properly linked with the Mercury Interactive object file and library. Refer to the “Troubleshooting” section at the end of this chapter.

SunOS—OLIT Applications

This section is relevant if your application is running under SunOS and is based on the OLIT 3.1 widget set.

Overview

To link your application to Mercury Interactive’s Context Sensitive library, you have two options. The option you select depends on how your application is linked with the X11 (Xlib), Xt (X Toolkit), and Xol (OLIT) libraries.

Dynamic Use this option if your application is dynamically linked with the X11 (Xlib), Xt (X Toolkit) and Xol (OLIT) libraries.

Static Use this option if your application is statically linked with any one of the above libraries.

In the section that follows, you will first determine how your application is linked. Then, you will choose the appropriate option that will link your application to Mercury Interactive’s Context Sensitive library.

Before you invoke XRunner, set the MIC_TOOLKIT environment variable to OLIT. In the window shell in which you intend to invoke XRunner, type:

```
setenv MIC_TOOLKIT Olit
```

Determining How Your Application is Linked

- 1 Determine whether your application is dynamically linked with the OLIT, X Toolkit, and X11 Libraries by entering the following command:

ldd <Application Under Test Executable>

The result of this command will be a list of all dynamic libraries with which your application is currently linked. For example:

```
-lXol.3 => /tools/openwin3/lib/libXol.so.3.1    # OLIT library
-lXt.4 => /usr/lib/X11/libXt.so.4.0           # Xt library
-lX11.4 => /usr/lib/X11/libX11.so.4.3        # X11 library
-lc.1 => /usr/lib/libc.so.1.8
-ldl.1 => /usr/lib/libdl.so.1.0
```

- 2 If **all** three libraries: -lXol, -lXt and -lX11 appear in this list (as in the above example), your application is dynamically linked with these libraries. In such a case, use the Dynamic option to link your application with the Context Sensitive library. Follow the instructions in the next section, “Dynamic Option.”

If **any one** of the -lXol, -lXt or -lX11 libraries does not appear in the list, your application is statically linked with that library. Skip the next section and follow the instructions under “Static Option.”

Dynamic Option

Using the Dynamic option, you dynamically link Mercury Interactive’s Context Sensitive library to your application. You create two symbolic links between the Mercury Interactive library and the original Xt library:

- Between Mercury Interactive’s *liborig.so.4.0* and the original *libXt.so* library
- Between Mercury Interactive’s *libXt.sa.4.0* and the original *libXt.sa* library

- 1 Locate the original dynamic Xt library in your system by repeating the ldd command as shown in step 1 of the previous section.

Let’s assume that in the ldd list you get the following line:

```
-lXt.4 => /usr/lib/X11/libXt.so.4.0
```

- 2 Enter the following commands:

```
ln -s /usr/lib/X11/libXt.so.4.0 $M_ROOT/arch/X11.4/Xol3.1/liborig.so.4.0  
ln -s /usr/lib/X11/libXt.sa.4.0 $M_ROOT/arch/X11.4/Xol3.1/libXt.sa.4.0
```

Note that once you create these links, you do not have to repeat this step again.

- 3 Change the dynamic library path to include the Mercury Interactive library, by entering the command:

```
setenv LD_LIBRARY_PATH  
$M_ROOT/arch/X11.4/Xol3.1:$LD_LIBRARY_PATH
```

- 4 Check that your application uses Mercury Interactive libraries by issuing the ldd command again. This time you should receive the following entries for the Xt and liborig libraries:

```
-lXol.3 => /tools/openwin3/lib/libXol.so.3.1  
-lXt.4 => $M_ROOT/arch/X11.4/Xol3.1/libXt.so.4.1  
-lX11.4 => /usr/lib/X11/libX11.so.4.3  
-lc.1 => /usr/lib/libc.so.1.8  
-ldl.1 => /usr/lib/libdl.so.1.0  
-lorig.4 => $M_ROOT/arch/X11.4/Xol3.1/liborig.so.4.0
```

- 5 Your application is now ready for Context Sensitive testing. When you start your application, you should receive the message:

“Loaded Mercury Interactive’s Context Sensitive Library: *<library version>*.”

If this message does not appear, your application is not properly linked with the Mercury Interactive library. Refer to the “Troubleshooting” section at the end of the chapter.

Static Option

Using the Static option you statically link your application with Mercury Interactive’s object file and library (*mic_ol_s.o* and *libmic_ol.a*), by modifying your application’s makefile.

- 1 Edit your application’s makefile to include:

```
$M_ROOT/arch/X11.4/Xol3.1/mic_ol_s.o (must appear before libraries).
```

`$(M_ROOT)/arch/X11.4/Xol3.1/libmic_ol.a` (must appear before other libraries).

- 2 The following example shows the relevant part of the makefile before and after the necessary modifications:

Before:

```
cc $(CFLAGS) -o <application_name> \
    <application_source>.o \
    -Bstatic\
    -L/usr/lib/X11 \
    -lXol -lXt -lX11
```

After:

(Bold characters indicate the addition that was made)

```
cc $(CFLAGS) -o <application_name> \
    <application_source>.o \
    -Bstatic\
    $(M_ROOT)/arch/X11.4/Xol3.1/mic_ol_s.o \
    $(M_ROOT)/arch/X11.4/Xol3.1/libmic_ol.a \
    -L/usr/lib/X11 \
    -lXol -lXt -lX11
```

- 3 Your application is now ready for Context Sensitive testing. When you start your application, you should receive the message: “Loaded Mercury Interactive’s Context Sensitive Library: *<library version>*.”

If this message does not appear, your application is not properly linked with the Mercury Interactive object file and library. Refer to the “Troubleshooting” section at the end of this chapter.

Solaris—Motif Applications

This section is relevant if your application is running under the Solaris operating system, using the Motif widget set.

Overview

To link your application to Mercury Interactive's Context Sensitive library, you have three options. The option you select depends on how your application is linked with the X11 (Xlib), Xt (X Toolkit) and Xm (Motif) libraries:

| | |
|----------------|---|
| Dynamic | Use this option if your application is dynamically linked with the X11 (Xlib), Xt (X Toolkit), and Xm (Motif) libraries. |
| Dlopen | Use this option if your application is statically linked with any one of the three libraries. |
| Static | Use this option if your application is statically linked with either one of the three libraries, and you cannot use the Dlopen option. |

In the sections that follow, you will first determine how your application is linked. Then, you will choose the appropriate option that will link your application to Mercury Interactive's Context Sensitive library.

Determining How Your Application is Linked

- 1 Determine whether your application is dynamically linked with the Motif, X Toolkit, and X11 Libraries by entering the following command:

```
ldd <Application Under Test Executable>
```

The result of this command will be a list of all dynamic libraries with which your application is currently linked. For example:

```
libXm.so.2 => /opt/SUNWmotif/lib/libXm.so.2      # Motif
libXt.so.4 => /usr/openwin/lib/libXt.so.4      # XToolkit
libX11.so.4 => /usr/openwin/lib/libX11.so.4    # X11
libc.so.1 => /usr/lib/libc.so.1
libsocket.so.1=>/usr/lib/libsocket.so.1
libnsl.so.1 => /usr/lib/libnsl.so.1
libdl.so.1 => /usr/lib/libdl.so.1
libw.so.1 => /usr/lib/libw.so.1
libintl.so.1 => /usr/lib/libintl.so.1
```

- 2 If all three libraries: -lXm, -lXt and -lX11 appear in this list (as in the above example), your application is dynamically linked with these libraries. In

such a case, use the Dynamic option to link your application with the Context Sensitive library. Follow the instructions in the next section, “Dynamic Option.”

If **any one** of the -lXm, -lXt or -lX11 libraries does not appear in the list, your application is statically linked with that library. In such a case, you should use either the Dlopen option or the Static option in order to link your application with the Context Sensitive library.

Dynamic Option

Using the Dynamic option, you dynamically link Mercury Interactive’s Context Sensitive library to your application. This is done by creating a symbolic link and setting an environment variable.

First you need to create a symbolic link between the Mercury Interactive library and the original Xt library:

`$M_ROOT/arch/X11.5/Xm1.2/liborig.so` → the original libXt.so library

- 1 Locate the original dynamic Xt library in your system by repeating the ldd command as shown in step 1 of the previous section. Let’s assume that in the ldd list you get the following line:

`libXt.so.4 => /usr/openwin/lib/libXt.so.4`

- 2 Enter the following command:

In -s /usr/openwin/lib/libXt.so.4 \$M_ROOT/arch/X11.5/Xm1.2/liborig.so

Note that once you create this link, you do not have to repeat this step again.

- 3 Change the dynamic library path to include the appropriate Mercury Interactive library, by entering the command:

**setenv LD_LIBRARY_PATH
\$M_ROOT/arch/X11.5/Xm1.2:\$LD_LIBRARY_PATH**

Note: It is recommended that you do not set LD_LIBRARY_PATH globally in your start-up file. You may have trouble invoking other applications.

- 4 Check that your application uses Mercury Interactive libraries by issuing the `ldd` command again. This time you should receive the following entries for the Xt and liborig libraries:

```
libXm.so.2 => /opt/SUNWmotif/lib/libXm.so.2
libXt.so.4 => $M_ROOT/arch/X11.5/Xm1.2/libXt.so.4
libX11.so.4 => /usr/openwin/lib/libX11.so.4
libc.so.1 => /usr/lib/libc.so.1
liborig.so => $M_ROOT/arch/X11.5/Xm1.2/liborig.so
libdl.so.1 => /usr/lib/libdl.so.1
libsocket.so.1 => /usr/lib/libsocket.so.1
libnsl.so.1 => /usr/lib/libnsl.so.1
libw.so.1 => /usr/lib/libw.so.1
libintl.so.1 => /usr/lib/libintl.so.1
```

- 5 Your application is now ready for Context Sensitive testing. When you start your application, you should receive the message:

“Loaded Mercury Interactive’s Context Sensitive Library: *<library version>*.”

If this message does not appear, your application is not properly linked with the Mercury Interactive library. Refer to the “Troubleshooting” section at the end of the chapter.

Dlopen Option

Using the `Dlopen` option, you re-link your application with Mercury Interactive’s object file, by modifying your application’s makefile. During runtime, the object file dynamically opens the appropriate Context Sensitive library.

- 1 Edit your application’s makefile to include:

The *mic_xm.o* Mercury Interactive object file (must appear before other libraries).

The *-Bdynamic* and *-ldl* flags (must appear at the end of the linking string).

- 2 The following example shows the relevant part of the makefile before and after the necessary modifications.

Before:

```
cc $(CFLAGS) -o <application_name> \
    <application_source>.o \
```

```
-Bstatic \
  -IMrm -lXm -lXt -lX11 -lnsl \
  -lgen -lw -lintl -lsocket
```

After:

(Bold characters indicate the additions that were made)

```
cc $(CFLAGS) -o <application_name> \
  <application_source>.o \
  $(M_ROOT)/arch/X11.5/Xm1.2/mic_xm.o \
  -Bstatic \
  -IMrm -lXm -lXt -lX11 -lnsl \
  -lgen -lw -lintl -lsocket\
-Bdynamic -ldl
```

- 3 Your application is now ready for Context Sensitive Testing. When you start your application, you should receive the message:
“Loaded Mercury Interactive’s Context Sensitive Library: *<library version>*.”

If this message does not appear, your application is not properly linked with the Mercury Interactive library. Refer to the “Troubleshooting” section at the end of the chapter.

Static Option

Using the Static option you re-link your application with Mercury Interactive’s object file and library (*mic_xm_s.o* and *libmic_xm.a*), by modifying your application’s makefile.

It is recommended that you use this option only if you encounter problems using the Dlopen option.

- 1 Edit your application’s makefile to include:

`$(M_ROOT)/arch/X11.5/Xm1.2/mic_xm_s.o` (must appear before libraries).

`$(M_ROOT)/arch/X11.5/Xm1.2/libmic_xm.a` (must appear before other libraries).

- 2 The following example shows the relevant part of the makefile before and after the necessary modifications:

Before:

```
cc $(CFLAGS) -o <application_name> \
```

```
<application_source>.o \  
-Bstatic \  
-lMrm -lXm -lXt -lX11 -lnsl \  
-lgen -lw -lintl -lsocket
```

After:

(Bold characters indicate the additions that were made)

```
cc $(CFLAGS) -o <application_name> \  
<application_source>.o \  
$(M_ROOT)/arch/X11.5/Xm1.2/mic_xm_s.o \  
$(M_ROOT)/arch/X11.5/Xm1.2/libmic_xm.a \  
-Bstatic \  
-lMrm -lXm -lXt -lX11 -lnsl \  
-lgen -lw -lintl -lsocket
```

- 3 Your application is now ready for Context Sensitive testing. When you start your application, you should receive the message:

“Loaded Mercury Interactive’s Context Sensitive Library: *<library version>*.”

If this message does not appear, your application is not properly linked with the Mercury Interactive object file and library. Refer to the “Troubleshooting” section below.

Solaris—XView Applications

This section is relevant if your application is running under Solaris and is based on the XView 3.2 widget set.

Overview

To link your application to Mercury Interactive’s Context Sensitive library, you have two options. The option you select depends on how your application is linked with the X11 (Xlib) and XView libraries.

- | | |
|----------------|---|
| Dynamic | Use this option if your application is dynamically linked with both the X11 (Xlib) and XView libraries. |
| Static | Use this option if your application is statically linked with either X11 or XView. |

In the section that follows, you will first determine how your application is linked. Then, you will choose the appropriate option that will link your application to Mercury Interactive's Context Sensitive library.

Note: Before you invoke XRunner, set the MIC_TOOLKIT environment variable to XView. In the window shell in which you intend to invoke XRunner, type:

```
setenv MIC_TOOLKIT XView.
```

Determining How Your Application is Linked

- 1 Determine whether your application is dynamically linked with the XView and X11 Libraries by entering the following command:

ldd <Application Under Test Executable>

The result of this command will be a list of all dynamic libraries with which your application is currently linked. For example:

```
libdstt.so.0 => /usr/openwin/lib/libdstt.so.0
libtt.so.1 => /usr/openwin/lib/libtt.so.1
libw.so.1 => /usr/lib/libw.so.1
libsocket.so.1 => /usr/lib/libsocket.so.1
libnsl.so.1 => /usr/lib/libnsl.so.1
libintl.so.1 => /usr/lib/libintl.so.1
libdl.so.1 => /usr/lib/libdl.so.1
libdeskset.so.0 => /usr/openwin/lib/libdeskset.so.0
libxview.so.3 => /usr/openwin/lib/libxview.so.3      /*Xview*/
libolgx.so.3 => /usr/openwin/lib/libolgx.so.3
libX11.so.4 => /usr/openwin/lib/libX11.so.4      /*X11*/
libc.so.1 => /usr/lib/libc.so.1
```

- 2 If **both** the **libxview** and **libX11** libraries appear in this list (as in the above example), your application is dynamically linked with these libraries. In such a case, use the Dynamic option to link your application with the Context Sensitive library. Follow the instructions in the next section, "Dynamic Option."

If **either libxview** or **libX11** do not appear in the list, your application is statically linked with that library. Skip the next section and follow the instructions under “Static Option.”

Dynamic Option

Using the Dynamic option, you dynamically link Mercury Interactive’s Context Sensitive library to your application. You create a symbolic link between the Mercury Interactive library and the original Xt library:

- 1 Between Mercury Interactive’s *liborig.so* and the original *libxview.so* library, locate the original dynamic XView library in your system by repeating the `ldd` command as shown in step 1 of the previous section.

Let’s assume that in the `ldd` list you get the following line:

```
libxview.so.3 => /usr/openwin/lib/libxview.so.3
```

- 2 Enter the following command:

```
In -s /usr/openwin/lib/libxview.so.3 $M_ROOT/arch/XView3.2/liborig.so
```

Note that once you create these links, you do not have to repeat this step again.

- 3 Change the dynamic library path to include the appropriate Mercury Interactive library, by entering the command:

```
setenv LD_LIBRARY_PATH $M_ROOT/arch/XView3.2:$LD_LIBRARY_PATH
```

- 4 Check that your application uses Mercury Interactive libraries by issuing the `ldd` command again. Assuming that the Context Sensitive libraries are stored in `u/qa`, this time you should receive the following entries for the `libxview` and `liborig` libraries:

```
libdstt.so.0 => /usr/openwin/lib/libdstt.so.0
libtt.so.1 => /usr/openwin/lib/libtt.so.1
libw.so.1 => /usr/lib/libw.so.1
libsocket.so.1 => /usr/lib/libsocket.so.1
libnsl.so.1 => /usr/lib/libnsl.so.1
libintl.so.1 => /usr/lib/libintl.so.1
libdl.so.1 => /usr/lib/libdl.so.1
libdeskset.so.0 => /usr/openwin/lib/libdeskset.so.0
libxview.so.3 => /u/qa/arch/XV3.2/libxview.so.3
```

```
libolgx.so.3 => /usr/openwin/lib/libolgx.so.3
```

```
libX11.so.4 => /usr/openwin/lib/libX11.so.4
```

```
libc.so.1 => /usr/lib/libc.so.1
```

```
liborig.so => /u/qa/arch/XV3.2/liborig.so
```

- 5 Your application is now ready for Context Sensitive testing. When you start your application, you should receive the message:

“Loaded Mercury Interactive’s Context Sensitive Library: *<library version>*.”

If this message does not appear, your application is not properly linked with the Mercury Interactive library. Refer to the “Troubleshooting” section at the end of the chapter.

Static Option

Using the Static option you statically link your application with the library provided by Mercury Interactive (*libmic_xv.a*).

- 1 Edit your application’s makefile to include:

```
$M_ROOT/arch/XView3.2/libmic_xv.a (and -lgen, if necessary)
(must appear before other libraries.)
```

- 2 The following example shows the relevant part of the makefile before and after the necessary modifications:

Before:

```
cc $(CFLAGS) -o <application-name> \
  <application_source.o> \
  -L/usr/openwin/lib -Bstatic -lxview -lolgx -lXext -lX11 \
  -Bdynamic -lw -lintl -lsocket -lnsl -ldl
```

After:

```
cc $(CFLAGS) -o <application-name> \
  <application_source.o> \
  $(M_ROOT)/arch/XView3.2/libmic_xv.a \
  -L/usr/openwin/lib -Bstatic -lxview -lolgx -lXext -lX11 \
  -Bdynamic -lw -lintl -lsocket -lnsl -ldl -lgen
```

- 3 Your application is now ready for Context Sensitive testing. When you start your application, you should receive the message:

“Loaded Mercury Interactive’s Context Sensitive Library: *<library version>*.”

If this message does not appear, your application is not properly linked with the Mercury Interactive library. Refer to the “Troubleshooting” section at the end of this chapter.

Solaris—OLIT Applications

This section is relevant if your application is running under Solaris and is based on the OLIT 3.2 widget set.

Overview

To link your application to Mercury Interactive’s Context Sensitive library, you have two options. The option you select depends on how your application is linked with the X11 (Xlib), Xt (X Toolkit), and Xol (OLIT) libraries.

Dynamic Use this option if your application is dynamically linked with the X11 (Xlib), Xt (X Toolkit) and Xol (OLIT) libraries.

Static Use this option if your application is statically linked with any one of the above libraries.

In the section that follows, you will first determine how your application is linked. Then, you will choose the appropriate option that will link your application to Mercury Interactive’s Context Sensitive library.

Note: Before you invoke XRunner, set the MIC_TOOLKIT environment variable to OLIT. In the window shell in which you intend to invoke XRunner, type:

```
setenv MIC_TOOLKIT Olit.
```

Determining How Your Application is Linked

- 1 Determine whether your application is dynamically linked with the OLIT, X Toolkit, and X11 Libraries by entering the following command:

ldd <Application Under Test Executable>

The result of this command will be a list of all dynamic libraries with which your application is currently linked. For example:

```
libXol.so.3 => /usr/openwin/lib/libXol.so.3      #OLIT library
libXt.so.4 => /usr/openwin/lib/libXt.so.4      #Xt library
libX11.so.4 => /usr/openwin/lib/libX11.so.4    #X11 library
libc.so.1 => /usr/lib/libc.so.1
libsocket.so.1 => /usr/lib/libsocket.so.1
libnsl.so.1 => /usr/lib/libnsl.so.1
libdl.so.1 => /usr/lib/libdl.so.1
libw.so.1 => /usr/lib/libw.so.1
libintl.so.1 => /usr/lib/libintl.so.1
```

- 2 If all three libraries: libXol, libXt and libX11 appear in this list (as in the above example), your application is dynamically linked with these libraries. In such a case, use the Dynamic option to link your application with the Context Sensitive library. Follow the instructions in the next section, “Dynamic Option.”

If any one of the libXol, libXt and libX11 libraries does not appear in the list, your application is statically linked with that library. Skip the next section and follow the instructions under “Static Option.”

Dynamic Option

Using the Dynamic option, you dynamically link Mercury Interactive’s Context Sensitive library to your application. You create a symbolic link between the Mercury Interactive library and the original Xt library:

- 1 Between Mercury Interactive’s *liborig.so.4* and the original *libXt.so* library. Locate the original dynamic Xt library in your system by repeating the `ldd` command as shown in step 1 of the previous section.

Let’s assume that in the `ldd` list you get the following line:

```
libXt.so.4 => /usr/openwin/lib/libXt.so.4
```

- 2 Enter the following command:

```
ln -s /usr/openwin/lib/libXt.so.4 liborig.so
```

Note that once you create this link, you do not have to repeat this step again.

- 3 Change the dynamic library path to include the Mercury Interactive library, by entering the command:

```
setenv LD_LIBRARY_PATH  
$M_ROOT/arch/X11.5/Xol3.2:$LD_LIBRARY_PATH
```

- 4 Check that your application uses Mercury Interactive libraries by issuing the ldd command again. This time you should receive the following entries for the Xt and liborig libraries:

```
libXol.so.3 => /usr/openwin/lib/libXol.so.3  
libXt.so.4 => $M_ROOT/arch/X11.5/Xol3.2/libXt.so.4  
libX11.so.4 => /usr/openwin/lib/libX11.so.4  
libc.so.1 => /usr/lib/libc.so.1  
liborig.so => $M_ROOT/arch/X11.5/Xol3.2/liborig.so  
libdl.so.1 => /usr/lib/libdl.so.1  
libsocket.so.1 => /usr/lib/libsocket.so.1  
libnsl.so.1 => /usr/lib/libnsl.so.1  
libw.so.1 => /usr/lib/libw.so.1  
libintl.so.1 => /usr/lib/libintl.so.1
```

- 5 Your application is now ready for Context Sensitive testing. When you start your application, you should receive the message:

“Loaded Mercury Interactive’s Context Sensitive Library: *<library version>*.”

If this message does not appear, your application is not properly linked with the Mercury Interactive library. Refer to the “Troubleshooting” section at the end of the chapter.

Static Option

Using the Static option you statically link your application with Mercury Interactive’s object file and library (*mic_ol_s.o* and *libmic_ol.a*), by modifying your application’s makefile.

- 1 Edit your application’s makefile to include:

`$M_ROOT/arch/X11.5/Xol3.2/mic_ol_s.o` (must appear before libraries).

`$M_ROOT/arch/X11.5/Xol3.2/libmic_ol.a` (must appear before other libraries).

- 2 The following example shows the relevant part of the makefile before and after the necessary modifications:

Before:

```
cc -o <executable> <application_source>.o \
    -Bstatic -lXol -lXt -lolgx -lXext -lX11 \
    -Bdynamic -lw -lintl -lsocket -lnsl -ldl -lgen
```

After:

(Bold characters indicate the addition that was made)

```
cc -o <executable> <application_source>.o \
    $(M_ROOT)/arch/X11.5/Xol3.2/mic_ol_s.o \
    $(M_ROOT)/arch/X11.5/Xol3.2/libmic_ol.a \
    -Bstatic -lXol -lXt -lolgx -lXext -lX11 \
    -Bdynamic -lw -lintl -lsocket -lnsl -ldl -lgen
```

- 3 Your application is now ready for Context Sensitive testing. When you start your application, you should receive the message: “Loaded Mercury Interactive’s Context Sensitive Library: <library version>.”

If this message does not appear, your application is not properly linked with the Mercury Interactive object file and library. Refer to the “Troubleshooting” section at the end of this chapter.

Troubleshooting

The paragraphs below provide solutions to some problems you may encounter during installation.

The first thing you should check when your link fails, is that the `$M_ROOT` environment variable is defined correctly, as explained below. Depending on which type of linking you performed (Dynamic, Dlopen, or Static), there might be more specific problems. Refer to the appropriate section.

Check Your `M_ROOT` Setting

- If you are using the Context Sensitive libraries which came with XRunner, then `$M_ROOT` should point to the XRunner installation directory.

- If you are using Context Sensitive libraries outside of the XRunner tree (libraries which were downloaded from tape separately from XRunner), then you should have two window shells.
- In one window shell (in which you invoke XRunner), `$M_ROOT` should point to the XRunner installation directory. For example, if XRunner is installed under `/tools/QA`, then you should enter:

```
setenv M_ROOT /tools/QA
```

- In the second window shell (in which you invoke your application), `$M_ROOT` should point to the directory in which you downloaded the Context Sensitive libraries. For example, if you downloaded the libraries into the directory `/tools/QA/cs_libs`, then type:

```
setenv M_ROOT /tools/QA/cs_libs
```

Dynamic or Static Linking

If you performed the SunOS—Motif linking procedure, the reason for failure may be a mismatch between the X11, Xt or Xm libraries used by your application and those used by the Mercury Interactive library. In such a case, you probably received one of the following messages.

| Error Message | Meaning |
|--|---|
| “Undefined symbol XrmPermStringToQuark” | There is a mismatch between the original Xt library and the X11 library. Your liborig points to the Xt release 5 library, while your application is linked with the X11 release 4 library. |
| “Undefined symbol XtCvtStringToFontStruct” | There is a mismatch between the original Xt library and the X11 library. Your liborig points to the Xt release 4 library, while your application is linked with the X11 release 5 library. |

| Error Message | Meaning |
|-----------------------------------|---|
| “Undefined symbol XtShellStrings” | There is a mismatch between the Xt library used by the Mercury Interactive <variant> and the original Xt library pointed by the liborig <variant>. The Mercury Interactive <variant> uses the Xt release 5 library, while the liborig <variant> points to the Xt release 4 library. |
| “Undefined symbol XmStrings” | There is a mismatch between the Xm library used by the Mercury Interactive <variant> and the Xm library used by the application. Your application is dynamically linked with the Xm 1.1 library, while the Mercury Interactive <variant> points to the Xm 1.2 library. |

In a dynamic link, the solution to these errors involves one of the following:

- Modifying the LD_LIBRARY_PATH, so that it uses the correct Mercury Interactive <variant>.
- Modifying the symbolic links, so that they use the correct Xt library used by your application together with the corresponding Mercury Interactive <variant>.

In a static link, you should modify the <variant> you used in the makefile.

Dlopen Linking

You may have used the wrong Mercury Interactive object file. In such a case, you probably received one of the following error messages:

| Error Message | Meaning |
|-------------------------------------|--|
| “Cannot mix pic and PIC” | Your application is compiled using a -PIC flag, and you linked the <i>mic_xm.o</i> object file which supports a -pic flag compilation. Use the <i>mic_xm_PIC.o</i> object file. |
| “no of realloc != no of alloc” | Your application is compiled without a -pic flag, and you linked the <i>mic_xm.o</i> object file which supports a -pic flag compilation. Use the <i>mic_xm_nopic.o</i> object file. |
| “Undefined symbol: _dlsym, _dlopen” | You did not insert the <i>-Bdynamic</i> flag in the makefile. Insert the <i>-Bdynamic</i> flag at the end of the linking string. |

Enabling Java Support

When you install XRunner, Java support is automatically installed. This section provides instructions for enabling the Java support. You can then use XRunner to record and replay Context Sensitive functions on Java objects.

This section describes:

- The Java Support Environment
- Setting Up Java Support
- Troubleshooting

The Java Support Environment

XRunner provides full Context Sensitive testing of Java applets and applications using various environments, platforms, and browsers.

Supported Environments

XRunner supports the following Java development toolkits:

- Java Development Kit (JDK) version 1.1 and 1.2, including AWT providing the standard GUI objects for Java
- Java Foundation Class (JFC), which provides the “Swing” UI for Java
- Symantec Visual Café 2.5 and 3.0.

Supported Browsers

XRunner supports the following browsers:

- Netscape 4.08 and 4.51.

Note: Netscape browser only supports JFC version 1.02 or higher. There are no JFC limitations for LoadApp and standalone applications.

Setting Up Java Support

In order to test Java objects, you must perform the following steps:

- 1 Check that your version of Netscape can run JDK 1.1 and JDK 1.2 applets.
- 2 Verify that there is a minimum of 5 MB disk space available in the browser installation directory.
- 3 Set the LD_LIBRARY_PATH and CLASSPATH environment variables as follows:
 - `setenv LD_LIBRARY_PATH $M_ROOT/arch/;$LD_LIBRARY_PATH`
 - `setenv CLASSPATH $M_ROOT/classes`
or if your web server is not configured to work with Mercury Interactive support:
`setenv CLASSPATH $M_ROOT/classes:$M_ROOT/classes/srv`
- 4 Open the file *XRunner installation folder/classes/mercury.properties* and make sure that the mic_toolkit is configured to the correct toolkit environments.
- 5 If you want to record a standalone Java application or Java applet, you must start it by entering one of the following strings:
 - to record a **Java applet** if you have **JDK** installed and it runs under AppletViewer:
`java LoadApp <URL address>`
 - to record a **Java applet** if you have **JRE** installed and it runs under AppletViewer:
`jre -cp $CLASSPATH LoadApp <URL address>`

- 6 Once you have started the standalone application or Java applet, verify that the following message appears:

```
init mercury support
Loading <development toolkit> support...
```

- 7 To install support for Netscape 4.08 or Netscape 4.51 Professional Edition , you must copy the following files:

```
copy ~/.netscape/java40.jar to ~/.netscape/java40.jar.orig
```

```
copy $M_ROOT/Java_patches/ns4[08,51]-java40.jar to ~/.netscape/java40.jar
```

Troubleshooting

If you experience problems recording from Netscape, begin by checking that XRunner works with the LoadApp appletviewer.

- If you have JDK installed, enter:

```
java LoadApp <URL address>
```

- If you have JRE installed, enter:

```
jre -cp %classpath% LoadApp <URL address>
```

Then check the classpath and verify that it contains the *XRunner installation folder/classes* directory. Check the path and verify that it contains the *XRunner installation folder/arch* directory.



Mercury Interactive Corporation

1325 Borregas Avenue
Sunnyvale, CA 94089 USA

Main Telephone: (408) 822-5200

Sales & Information: (800) TEST-911

Customer Support: (408) 822-5400

Fax: (408) 822-5300

Home Page: www.merc-int.com

Customer Support: web.merc-int.com



* 408 822 5400 * 911 *