XTester - A System for Testing X Applications

Ellis Horowitz and Zafar Singhera

Department of Computer Science
University of Southern California
Los Angeles, California 90089-0781

USC-CS-93-549

Abstract

The testing of graphical user interfaces (GUI) is different from traditional software testing in a number of ways. New techniques and tools are required for testing graphical user interfaces. This document introduces a library of functions called XTester library, which can be used to build tools for GUI testing. The document also provides a detailed description of a tool, built on top of XTester library. This tool is used to perform regression testing on graphical user interfaces.

The functions provided by XTester library address the GUI objects by a symbolic name, instead of their absolute screen coordinates. It provides functions to capture user sessions with a GUI application in a user readable format. The syntax of the scripts generated by XTester functions is called User Interface Description Language (UIDL). XTester also provides functions to replay UIDL scripts to simulate user sessions with a GUI. Functions are also provided to capture and verify object parameters or screen images for the verification of the GUI behavior. XTester also provides functions to include audio commentary during the replay of a script.
XTester is a library of functions which can be used to develop tools for testing and demonstrating X applications. It provides functions to capture and replay user sessions for an X application. It provides a language which can be used to write scripts for simulating user sessions and a syntax to specify the object hierarchy of a graphical user interface (GUI). The set of functions provided by XTester can be used in a C program to write customized, application specific testing tools. XTester operates at the X protocol level which makes it independent of the toolkit used for the development of the application under test (AUT). For example, the tools developed using XTester functions are equally good for testing Athena, Motif or Openlook applications. XTester does not require any modification, recompilation or relinking of the application under test. It works on any standard Xserver and does not require any extensions. The user has the liberty to use any of the standard development and debugging tools for the C language, during the development and debugging of XTester based testing tools and even during replay of the failed test scripts.

This manual describes the features of the first release of XTester. Section 1 provides instructions on installing XTester on a workstation. Section 3 consists of a tutorial on the basic functionality of XTester. Section 4 introduces a sample test suite, included in the release. Section 5 provides some suggestions for getting maximum benefit from XTester. It also mentions a few situations where XTester might fail. Section 6 and Section 7 describe details about the functionality and options of XTester in capture and replay mode, respectively. Section 8 introduces the Model window which is used to display and modify the model supported by XTester. Section 9 talks about the script window, which is used to display the script of the current session. Section 10 briefly describes the building of customized tools, using XTester library functions. Section 11 to Section 14 describe the resource and command line options, programming interface, script command and UIDL syntax. Section 15 mentions the weaknesses and bugs to the current version of XTester. Section 16 is used to acknowledge the contributions of people and tools during the development of XTester.

1 Installing XTester

The current release of XTester only runs on Sun Sparc workstations. It is distributed on a 1/4 inch tape or 3.5 inch floppies. It is also placed in the /pub/XTester directory on host usc.edu, from where it can be grabbed by anonymous ftp. The installation consists of two steps:

1. Load XTester from the distribution media to the hard disk. This step requires the following tasks to be performed:
   - Go to the directory where you want to install XTester.
   - Make sure that you have at least 6MBytes of disk space available
   - If you have received XTester on a tape or floppy, load it in the corresponding device and issue the following command:
tar xvf <device name>

• If you have got XTester by anonymous ftp then issue the following command:
  
zcat XTRel.tar.Z | tar xvf -

This will make a directory called XTester in your current working directory. To verify that the installation has been performed correctly, make sure that XTester has the same hierarchy as shown in Figure 1. XTester is distributed in two versions, basic and development. If you have received a basic version then you will have only shaded portions of the hierarchy otherwise you should get the entire hierarchy.

2. The second step modifies the user environment. The following tasks are required for this step.

• Define an environment variable XTesterHome and assign it your current 
XTester directory location. For example, if you have placed XTester directory in 
/usr/usc then /usr/usc/XTester should be assigned to variable XTesterHome. It can be done by using the following command:
  
setenv XTesterHome /usr/usc/XTester

• Modify your executable search path to include $XTesterHome/bin 
directory also. It can be done by using the following command:
  
set path=($path $XTesterHome/bin)

• Modify environment variable LD_LIBRARY_PATH to include 
$XTesterHome/lib. If using csh, the following command will do that:
  
setenv LD_LIBRARY_PATH $XTesterHome/lib:$LD_LIBRARY_PATH

• Load XTester resources from $XTesterHome/XTester.Defaults resource file, provided with the release, by using the following command:
  
xrdb -merge $XTesterHome/XTester.Defaults

To make sure that your corresponding startup files are modified properly so that these steps are automatically executed on future logins, a script, called ModifyEnv, is provided. The script modifies your ~/.login and ~/.Xdefaults files. If you have a customized environment, you might want to perform these steps manually instead of using the provided script. To use the script, do the following:

• Modify the line defining XTesterHome variable in ModifyEnv script. For example, if you have placed XTester in /usr/usc then the corresponding line in the script should look like:

setenv XTesterHome /usr/usc/XTester

• Change your current working directory to the location specified in environment variable $XTesterHome and run the script by issuing the following command:

./ModifyEnv
• Reflect the desired changes in the environment to the current working environment also, either by logging out and then logging back in or using modified ~/.login file, by saying,

```
source ~/.login
```
2 A Demo of XTester

$XTesterHome/bin/capdemo and $XTesterHome/bin/repdemo are provided in the release to get an introduction to the XTester user interface. These scripts automatically steer through all the windows of XTester, by running XTester under itself. The demo also includes audio information about the main features of XTester windows. $XTesterHome/demo directory provides the supporting files for the demo. To run the demo successfully, XTester must be installed properly following the instructions in the previous section. To get an overview of the capabilities of XTester in capture mode, type:

    capdemo

It will automatically steer through the XTester windows in capture mode and explain the functionality of XTester, while capturing a script for Xman application.

To get an overview of the capabilities of XTester in replay mode, type:

    repdemo

It will automatically steer through the XTester windows in replay mode and explain the functionality of XTester, while replaying the script captured by capdemo for Xman application.

3 A Tutorial on XTester

One of the standard and the most commonly used testing methods is regression testing. Regression testing helps in locating any newly introduced bugs in an application and involves capturing of a user session and then replaying that session automatically while verifying the behavior of the application being replayed. caprep is a simple regression testing tool built on top of the XTester library. It captures and replays individual user sessions and is included in the XTester distribution tape. It is one of the simplest, but probably the most useful GUI testing tool built using XTester library functions. Following is a tutorial to demonstrate the basic capabilities of XTester, as used by the caprep tool.

Setting the Environment

• Make sure you are running X windows and that XTester has been installed properly, as instructed in the previous section.

• Make sure that the X resources related to XTester are loaded. If not, run:

    xrdb -merge $XTesterHome/XTester.Defaults

• Check your shell environment for XTesterHome variable.

Capturing a Script for later replay

• Execute in an xterm the command
which starts XTester and the xman application. xman is a program which is contained in the standard X distribution and contains an X interface to the normal UNIX man pages utility. It takes a little while for the caprep window to appear.

The main XTester window appears along with the main xman window as shown in Figure 2.

![Initial XTester window and initial xman window](image)

**Figure 2. Initial XTester window and initial xman window**

- **Click on the capture button in XTester Capture Window** and all future keystrokes and mouse clicks will be saved.

- **Click on Manual Page button in xman main window** and a new window labeled Manual Page appears. If it covers the caprep window, you may move the Manual Page window to any convenient spot.

- **In Manual Page window Click on Options and drag to Display Directory command.** Instead of a man page, a list of man pages in the current section appears.

- **Click on Options and drag to Search and a dialog box appears.** This dialog box is labeled Search and is shown in Figure 3.

- **Enter cc in the text field and click on Manual Page** The man page for the cc command appears.

- **Click on Options, and drag to Show Both Screens** and a horizontal line with a box at the right hand end appears at the bottom of the Manual Page.

- **Press on the box and drag it up about half way, and then release**
• Click on Options, Remove this ManPage
and the manual page window will be removed.

• Click on exit in XTester main window
Since xman is not exited, you will see a “fatal IO error” message, but this can be ignored
for now. At this point XTester has saved two files; a script in the file xman.scr and statistics
in xman.stt. The script can be viewed by displaying it on the screen, and should be
similar to what is shown in Figure 4.

The scripts starts with a file header that records the time and date the script was created,
the name of the user who created the script, and other useful information. The remaining
commands are a sequence of button press, button release, and key presses that simulate the
actions taken by the user. The arguments of the commands refer to buttons, keys, and par-
ticular user interface objects.

**Replaying a script**

• Execute the command

  \texttt{caprep -rep -cmd xman}

and the main XTester replay window appears, and the main xman window appears as
shown in Figure 5.

• click on Replay
and the script that was saved earlier is replayed. Watch as the various commands are executed.
Since xman is not exited, you will see a “fatal IO error” message, but this can be ignored
for now.

• Rename the file xman.scr to xman.basic.scr

---

Figure 3. xman dialog box for searching
Capturing Images

In addition to capturing and replaying scripts, XTester can capture images and compare those reference images with later images during replay. For example:

```plaintext
FileHeader("1.0(0)-0","Mon Oct 4 14:08:38 1993","Zafar Singhera","xman","xman.img","User specified information, to be added later");
PressButton(Button1, "Xman-1.1.1", 61, 57, SameGroupMask);
ReleaseButton(Button1, "Xman-1.1.1", 61, 57, Button1Mask|SameGroupMask);
PressButton(Button1, "ManualPage-1.1.1.1", 60, 60, NoneMask);
ReleaseButton(Button1, "9", 26, 22, Button1Mask);
PressButton(Button1, "ManualPage-1.1.1.1", 32, 46, NoneMask);
ReleaseButton(Button1, "9", 23, 48, Button1Mask);
PressButton(Button1, "search-2.1.4", 5, 63, NoneMask);
ReleaseButton(Button1, "search-2.1.4", 5, 63, Button1Mask|SameGroupMask);
PressKey(c, "search-2.1.4", 5, 63, NoneMask);
PressKey(c, "search-2.1.4", 5, 63, SameGroupMask);
PressButton(Button1, "search-2.1.3", 17, 57, NoneMask);
ReleaseButton(Button1, "search-2.1.3", 17, 57, Button1Mask);
PressButton(Button1, "ManualPage-1.1.1.1", 35, 60, NoneMask);
ReleaseButton(Button1, "9", 21, 56, Button1Mask);
PressButton(Button1, "ManualPage-1.1.5", 62, 62, NoneMask);
ReleaseButton(Button1, "ManualPage-1.1.5", 62, 814575, Button1Mask);
PressButton(Button1, "ManualPage-1.1.1.1", 35, 60, NoneMask);
ReleaseButton(Button1, "9", 21, 64, Button1Mask);
EndOfSession(1);
```

![Figure 4. XTester script exercising some xman features](image)

In addition to capturing and replaying scripts, XTester can capture images and compare those reference images with later images during replay. For example:

```
caprep -cap -cmd xman
```

which starts XTester again with xman as the application under test.

- **Click on Capture**
- **Click on Grab**
  and the cursor changes shape to cross hairs.
- **Move the cursor into the xman main window and click the left mouse button.**
  You hear 3 beeps, indicating the image is captured.
In Xman, Click on Manual page

In XTester, Click on Grab

Place the cursor in the middle of the manual page information and click. Wait for the three beeps.

In Manual Page Click on Options, Search

Click on Grab
If you place the cursor over the label “Type string to search for:” and click, then only the label is saved. If you place the cursor over the Manual Page button, then only the Manual Page button is saved. To save the entire window

Place the cursor between the three buttons and click
This time the dialog box is saved.

Enter cc in the search field and hit Return

Click on grab
and place the cross hair cursor in the man page description of cc, and click.

Click on Options, Remove this ManPage
and the manual page window will be removed.

Click on Exit in XTester main window
XTester and the script with images is saved. Ignore the broken pipe message which appears.

XTester now saves three files: xman.scr, containing the script, xman.img, containing the images, and xman.stt, containing the statistics.

Figure 5. Main XTester replay window
The new script is shown in Figure 6.

```plaintext
FileHeader("1.0(0)-0","Mon Oct 4 14:43:33 1993","Zafar Singhera","xman","xman.img","User specified information, to be added later");
ObjectInfo(Pixmap,"Xman-1.1");
PressButton(Button1, "Xman-1.1.1", 42, 31, SameGroupMask);
ReleaseButton(Button1, "Xman-1.1.1", 42, 31,
Button1Mask|SameGroupMask);
ObjectInfo(Pixmap,"ManualPage-1.1.2");
PressButton(Button1, "ManualPage-1.1.1.1", 45, 20, NoneMask);
ReleaseButton(Button1, "9", 42, 47, Button1Mask);
ObjectInfo(Pixmap,"search-2.1");
PressKey(c, "search-2.1.4", 47, 47, NoneMask);
PressKey(c, "search-2.1.4", 47, 47, NoneMask);
PressKey(Return, "search-2.1.4", 47, 47, NoneMask);
ObjectInfo(Pixmap,"ManualPage-1.1.2");
PressButton(Button1, "ManualPage-1.1.1.1", 35, 60, NoneMask);
ReleaseButton(Button1, "9", 21, 64, Button1Mask);
EndOfSession(1);
```

**Figure 6. XTester script of xman that includes capturing screens**

The ObjectInfo command refers to a Pixmap as the data format and refers to the window that was captured.

- **To replay this script, type:**
  
  ```plaintext
caprep -rep -cmd xman
  ```

- **Click on Replay**

Since a .img file exists, images will be captured during replay and compared against the images in xman.img. If they are identical, nothing will happen. If they are different, then appropriate error messages are sent to the display console. When run on a color IPC with “focus grabbing” the messages produced were:

```
39381 pixels out of 321500 are different. Images differ on first attempt
```

Three attempts were tried. The script should then end successfully. Xman and XTester should both disappear.

- **Rename the file xman.scr to xman.image.scr**
- **Rename the file xman.img to xman.image.img**
Replaying at different speeds

One can replay a script and instead of watching it execute at the default speed, 1 second between events, we can select single step mode, by:

• **Start XTester in replay mode, by executing:**

  \[
  \text{caprep -rep -cmd xman -scr xman.basic.scr}
  \]

  and the main window of XTester appears.

• **Click on Options.**

  and the dialog box shown in Figure 7 appears.

At the lower right you see a Single Step button.

![Figure 7. XTester Replay Options Dialog Box](image)

• **Click on the Single Step button, click on Done**

• **Click on Replay**

  and for each script command to be executed click on the Next Step button.

• **Click on Next Step button until the script has completed.**

Organized testing of a user interface

In this section we show how to use the model capabilities of XTester to better organize your tests of the user interface.

• **Type:**
A Tutorial on XTester

caprep -cap -cmd xman

- click on Model and the XTester Graph Window, shown in Figure 8 appears.

There is a node labeled root, which corresponds to the entire xman application. It has four children: search, pleaseStandBy, likeToSave, and Xman. These names come from the application and represent corresponding window names. Only the Xman window is visible on the screen. The other windows have been created by xman, but not mapped. Notice that the Xman node has one child labeled 1, which represents a container form for all the elements that are visible in the Xman window. There are four children of the container form. Our job at this point is to assign meaningful names to these nodes, and to communicate how actions on these nodes effect other nodes.

- Click on node 1 of the Xman window
and a dialog box appears as shown in Figure 9.

The External Name represents the name given to the object by the user. The Internal name is the name given to the object by the application. In this case there is neither an external nor an internal name.

- Assign the External name to be XmanForm
Assign the Object type to Form

Click on Apply

Notice how the name is updated in the Model window.

Now assign the names to nodes 1, 2, 3, and 4, as shown in Table 1. You may have to scroll the Model window so the node is visible.

<table>
<thead>
<tr>
<th>Node</th>
<th>External Name</th>
<th>Object Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ManualPage</td>
<td>Button</td>
</tr>
<tr>
<td>2</td>
<td>Quit</td>
<td>Button</td>
</tr>
<tr>
<td>3</td>
<td>Help</td>
<td>Button</td>
</tr>
<tr>
<td>4</td>
<td>XmanLabel</td>
<td>Text</td>
</tr>
</tbody>
</table>

Table 1: Assign External Names to Nodes in Model Window

If you click on a node, the object it corresponds to is surrounded by a rectangle. This permits you to easily determine which object needs to be named.

Cancel the object information dialog box

In the Model window, click on Update and a file is saved, called xman.uidl. UIDL stands for user interface description language. It is fully explained in the XTester Reference Manual. Basically it contains a formal description of the user interface to assist in generating and evaluating test suites.
The tree shown in the model window is a dynamic representation of the windows of the application. For example, right now the root has only four children.

- **click on the Help button in the xman main window.**

Now look at the model window and you will see that a new child of the root has been added, called Help. You may scroll the Model window to locate this new node. The children of this node can be named in a similar way.

Names of objects in the UIDL file will be used in two places; in scripts and in statistics file. In both cases it makes it easier to read and understand the scripts and the results of tests if objects have been named.

- **Click on Exit in XTester**

### Executing a group of tests

XTester provides a mechanism to execute a script from another script. This mechanism allows the user to write high level scripts, which call other scripts in a particular sequence. The script shown in Figure 10 calls the previously shown scripts. The sections following the next demo section describe these features in more detail.

- **In your text editor create the following file whose name is Both.scr**

The FileHeader command in such scripts is not required. Each script can be run any number of times, provided that the script does not exit the application.

```plaintext
#
# Both.scr: This script executes the previous scripts
#

ExecScript("xman.basic.scr", "Both.img", Entire);
ExecScript("xman.image.scr", "xman.image.img", Entire);
```

*Figure 10. Grouping of scripts*

- **Type**

  caprep -rep -cmd xman -scr Both.scr

- **Click on Replay**

  and both script are replayed.
Gathering Statistics

Look at the xman.stt file in your directory. An example of a statistics file is shown in Figure 11.

A statistics file provides information about how many times a window has been created, destroyed, mapped and unmapped, and how many times a button press/release or a key-press/release have been exercised. In Figure 11, you see that every node of xman has been named. For example in line 1, one sees that the Xman object was created once, never destroyed, mapped twice, and unmapped once. By examining the statistics file, one gets a better picture of how thorough your test suite exercises the user interface.

XTester provides a number of options for capture and replay of user sessions, discussed in Section 6 and Section 7. It also provides a library of functions which can be used to build customized tests for a particular application or customized tools, Section 10.

4 A sample test suite

$XTesterHome/xmantests directory in the XTester release provides an example test suite, created by XTester’s caprep tool. This particular test suite provides a basic set of tests for testing xman. Xman has been slightly modified to provide names to the windows associated with menus. The modified version of Xman is called Xman (as opposed to xman) and is located in $XTesterHome/bin directory. $XTesterHome/xmantests/Xman.uidl provides the UIDL description for Xman. The test scripts for this particular example are located in $XTesterHome/xmantests/TestSuite directory. Although individual scripts in the hierarchy can be run independently, an All.scr script has been written for each window of Xman and it runs all the scripts which are related to the testing of that particular window. $XTesterHome/xmantests/TestSuite/All.scr executes the entire script, except the ones which force the application to exit. This release only provides the scripts and not the images (to keep the distribution size small). The Update mode of XTester can be used to capture the related verification data.

Once XTester has been installed, the following steps can be used to replay the scripts in the sample test suite:

• Make sure you have the required disk space to capture images.
  If you want to capture verification data and use it for verification in future, then make sure that you have at least 10MBytes of disk space in your account. However if you are only interested in replay of user sessions in the test suite, without verification, then you do not need any more disk space.

• Change directory to $XTesterHome/xmantests/TestSuite.

• Start caprep by issuing the following command:
Legend:
CW=Create Window DW=Destroy Window MW=Map Window
UMW=Unmap Window BP=Button Press BR=Button Release
KP=Key Press KR=Key Release

<table>
<thead>
<tr>
<th>Object Name</th>
<th>CW</th>
<th>DW</th>
<th>MW</th>
<th>UMWBp</th>
<th>BR</th>
<th>KP</th>
<th>KR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xman</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Xman*Form</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Xman*ManualPage</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>50</td>
<td>50</td>
<td>0</td>
</tr>
<tr>
<td>Xman*Quit</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Xman*Help</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>10</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>Xman*Label</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>StandBy</td>
<td>28</td>
<td>23</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>StandBy*Message</td>
<td>28</td>
<td>23</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>LikeToSave</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>LikeToSave*Form</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>LikeToSave*Message</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>LikeToSave*Yes</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>LikeToSave*No</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Help</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Help*Form</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Help*MenuBar</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Help*Options</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>7</td>
<td>20</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Help*Sections</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Help*Message</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Help*TextArea</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Help*Scrollbar</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>7</td>
<td>10</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>Help.Form.3</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ManPage</td>
<td>27</td>
<td>23</td>
<td>27</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ManPage*Form</td>
<td>27</td>
<td>23</td>
<td>27</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ManPage*MenuBar</td>
<td>27</td>
<td>23</td>
<td>27</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ManPage*Options</td>
<td>27</td>
<td>23</td>
<td>27</td>
<td>1</td>
<td>92</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>ManPage*Sections</td>
<td>27</td>
<td>23</td>
<td>27</td>
<td>1</td>
<td>18</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>ManPage*Message</td>
<td>27</td>
<td>23</td>
<td>27</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ManPage*TextArea</td>
<td>27</td>
<td>23</td>
<td>27</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ManPage*Scrollbar</td>
<td>27</td>
<td>23</td>
<td>27</td>
<td>1</td>
<td>8</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>ManPage.Form.3</td>
<td>27</td>
<td>23</td>
<td>27</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ManPage*DirArea</td>
<td>27</td>
<td>23</td>
<td>27</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ManPage*DirList</td>
<td>27</td>
<td>23</td>
<td>27</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ManPage*List</td>
<td>27</td>
<td>23</td>
<td>27</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 11: Statistics file created by XTester
caprep -rep -cmd Xman -dir ../ -scr All -log All

- Click on Options button in the main replay window.

- Select the appropriate Verify Option
  The release does not include images. Therefore selecting Verify will cause caprep to give an error message. To simply see how caprep runs the test suite, select the Don’t Verify option. To capture images for future verification, run caprep in Update mode.

- Remove the options window, by clicking on Done button

- Click on Replay button in the main replay window
  It will start the replay of the entire test suite.

XTester will create two files, $XTesterHome/xmantests/Xman.stt and $XTesterHome/xmantests/Xman.log. Xman.stt contains statistics about the current session while Xman.log contains the log information. $XTesterHome/xmantests/TestSuite/Cleanup command in $XTesterHome/TestSuite can be used to remove all the verification data from the test suite.

5 Getting the most out of XTester

Now that XTester has been successfully installed, you are ready to start using it on an application. Your first effort will be to run caprep, the XTester application that comes with the system. Caprep is a program for capturing and replaying test scripts. It has its own X-based user interface, which makes it especially easy to use. The application you intend to test we refer to as the AUT (Application Under Test).

5.1 Preparing to use Caprep

Before attempting to create any scripts, you should start caprep in capture mode. E.g. caprep -cap -cmd AUT, would start caprep in capture mode and start the application under test.

- Click on the button labeled Model
  A new window appears with a tree representing the window hierarchy of the AUT.

- Scan the names in the tree
  and look especially for any use of the characters: period (.), hyphen (-), and asterisk (*). These characters are used by caprep as the object separator (ObjSep), instance separator (InstSep), and wild card separator (WildSep). If they occur anywhere in the window names, then caprep needs to use another character, one that is not used in any of the window names in the AUT. This is done by editing the XTester.Defaults file and looking for the definitions of ObjSep, InstSep, and WildSep. When you start caprep in capture mode, the window hierarchy shown in the model window will automatically include
all windows created so far by the AUT. As you exercise the AUT and more windows are created, these are added to the tree in the Model window. You can attach symbolic names to each node in the tree, replacing caprep’s use of numbers.

• While the AUT is visible, click on a node in the model window. A box appears and you may enter a symbolic name. When you click on the node, the area of the application is highlighted, so it will be clear which area the name you are providing, refers to. Remember to save this tree by clicking on the Update button. The symbolic names are stored in a UIDL file and are used by caprep each time the AUT is exercised.

**Warning:** There are several features of an X-based user interface that are especially difficult for XTester to deal with.

• XTester cannot handle second and higher level menus. If the AUT uses a second level menu, then caprep will NOT be able to faithfully reproduce the user events, related to those menus.

• XTester cannot handle the case when the AUT grabs the focus of the server and prevents all other forms of input except those entered at the point of focus. If the AUT issues such a grab, then XTester will not replay the event properly.

• XTester is smart enough, that it will not send an event unless the window that is supposed to receive the event is already visible. HOWEVER, XTester is not able to recognize when a portion of a window has been displayed. E.g. XTester cannot handle scrolling. *If a script fails to replay, one can increase the default minimum delay between consecutive events.* This is done by altering the value of minimum delay in the Options dialog box.

### 6 Capture Mode

Figure 12 shows the main window which is displayed when caprep is started in capture mode. This section will discuss features of the window and its related functionality. The main window primarily consists of a set of command buttons. A brief description of the functionality of these buttons is given below:

**Options:** The Options button pops up an options window which allows the user to adjust various options related to capturing object information and inserting delays in the test script. The details of Capture Options Window will be described later in this section.

**Script:** When selected, this button pops up a window, called script window, which displays the script as it is captured. The script window is described in detail in Section 9.

**Model:** When selected, this button pops up a window, called Model Window. The Model Window displays the object hierarchies of the application under test. The details about the Model Window are described in Section 8.
Grab: When selected, this button causes information about an object or screen area to be captured and placed in the designated .img file. The mechanism of selecting the object and the format of the grabbed image are decided according to the currently chosen options in Capture Options Window, as explained later. This button is active only when not in PAUSE mode, otherwise it is grayed out.

WaitFor: When selected, this button places a delay command in the script that is being produced. XTester provides various types of delays which will be described in detail during the description of the options window. This button is also grayed out in PAUSE mode.

Capture/Pause: This button allows the user to pause the capturing of the script temporarily. When in PAUSE mode, XTester does not capture user actions, the application under tests responds in a normal way. When selected the label of this button and hence the mode of capture flips between Capture and Pause.

Exit: When selected, this button terminates the current capture session, saves the required information into appropriate files and exits.

The bottom area of the Main Capture Window displays various options currently selected, and provides messages during capturing. Grab message in the bottom area specifies the options currently selected for capturing object information from the screen. For example "Image by Cursor" under Grab message specifies that if the Grab button is selected then the object will be selected by clicking on it and its information will be stored as a pixmap. The Script File message displays the name of the file to which a captured script will be stored. The WaitFor message displays the current options selected for inserting delays in the script. For example, "Object by Cursor" under the WaitFor message represents that if the WaitFor button is selected, a delay event will be added to the script being generated. The object to wait for is selected by clicking a mouse button on it. When replayed, this delay event will cause the tool to wait for the object which was selected by mouse click during capturing. Another message string is displayed between the command buttons and the bottom status messages which provides general information about the current activities and instructs the user about the next possible events.
Figure 13 displays Capture Options Window. This window is logically divided in two halves, Image Options and Wait Options. The Image Options are related to capturing object information from the screen and its details are provided in Section 6.1 on page 19. The Wait Options control the generation of delay events in the generated script and its details are discussed in Section 6.2.

<table>
<thead>
<tr>
<th>Image Options</th>
<th>Wait Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format: Image, Attributes, Checksum</td>
<td>Wait for: Object, Duration, User</td>
</tr>
<tr>
<td>Grab By: Cursor, Key, Name, ID, Area, Max. Area</td>
<td>Select By: Cursor, Key, Name, ID</td>
</tr>
</tbody>
</table>

**Figure 13. Capture Options Window**

### 6.1 Capturing Object Information

XTester provides facility to capture information about objects in capture mode and store them in the .img file while placing an ObjectInfo command in the generated script. In replay mode, when the ObjectInfo event is replayed, XTester reads the stored object information from the .img file, automatically grabs the current object information from the screen and compares the two. If the current information does not match with the stored one then an error is reported. The details about object information verification are described in Section 7.

XTester is capable of storing object information in three different formats, Pixmap, Attributes or Checksum. The user can select the appropriate format to store object information by selecting the corresponding option from Format options set in Capture Options Window, shown in Figure 13. If Pixmap is selected, the images will be stored as pixmaps. If Attributes is selected, the information about object attributes, like location, dimensions, state, foreground and background colors, will be captured. If Checksum option is chosen, then a checksum of the object information is calculated and stored instead of the entire information about the object.

Most of the time object information has to be captured in pixmap format which take a lot of disk space. Moreover changes are usually limited to a particular object or a smaller area of a window. By default, XTester captures information about the smallest possible area of the screen. However this default behavior can be changed by turning on the Max. Area...
button in Capture Options Window. If Max. Area toggle button is on then the largest enclosing object of the application is selected.

XTester also provides various methods for selecting an object whose information should be captured by XTester. The user can select the appropriate method from Grab By set of exclusive options.

If Cursor option is selected and user clicks on Grab button in the Main Capture Window, the cursor is changed to cross hair and a message is displayed in the Message Area of the main window to move cursor on the object to be selected and press any mouse button. Once the user presses a mouse button, XTester verifies if the window under the cursor is related to the application under test or not. If the window is not related to AUT and is not the root window of the server then XTester beeps and allows the user to select another window. If the window is related to the application under test then it is highlighted by drawing a rectangle around it so that the user can verify that his selection covers the area he wants information about. Now if the user releases the mouse button then the highlighted object is selected and its information is stored in the .img file. However if the user changes his mind and wants to cancel his selection he can depress a key on the keyboard which is specified in KeyMask resource, before releasing the mouse button. In this case XTester will cancel this selection and allow the user to select another object.

If Key option is selected and user clicks on Grab button in the Main Capture Window, the user is instructed to move to the desired object and press any key except the ones specified in the KeyMask to select the object. The object verification, highlighting and cancelling procedures are the same as in Cursor option.

If Name or ID option is selected and user clicks on Grab button in the Main Capture Window, a dialog box pops up for the user to provide appropriate name or ID. If the user clicks on OK button in the dialog box then the string in the text field of the dialog box is verified and if it is root window or represents one of the AUT windows, then the information about the corresponding object is captured. If the user selects Cancel button in the dialog box then the object information is not captured and the dialog box disappears. Figure 14 shows the dialog box which appears to get an object name from the user. Dialog box to get object ID is the same except that “Object Name” string is replaced by “Object ID”.

If Area option is selected and user clicks on Grab button in the Main Capture Window, then the user is allowed to select an area of the screen to be captured by stretching a rectangle around it. The mechanism to stretch a rectangle is to move to the position of one of the corners of the required area and press a mouse button. With the mouse button pressed move to the diagonally opposite corner of the intended area and release the mouse button. XTester makes sure that the selected rectangle covers some area of AUT window and/or root screen. The location of the selected area is determined with respect to the smallest enclosing window.
6.2 Inserting Delays in Script

XTester provides a mechanism to insert delays in the captured script. The `WaitEvent` script command represents these inserted delays. Three types of delays are supported by XTester, which are `WaitForObject`, `WaitForDuration` and `WaitForUser`. During replay, `WaitForObject` stops XTester from simulating the next event in the script, until the corresponding object appears on the screen. `WaitForDuration` stops XTester from simulating the next event in the script for the specified number of seconds. `WaitForUser` stops XTester from simulating the next event in the script until a user response is received. Section 13 discusses `WaitEvent` in detail.

The `Wait Options` half of the Capture Options Window provides options for the creation of `WaitEvents`. The `Wait For` set of exclusive selections provides the user the ability to choose one of the three types of delays described above. When `WaitFor` button in the Capture Main Window is clicked, the type of the `WaitEvent` created corresponds to the current selection in `Wait For` exclusive options. The `Select By` options provide the method by which the object to wait for should be selected and it works in exactly the same manner as `Grab By` options in `Image Options` half.

7 Replay Mode

Figure 15 shows the main window displayed during replay of a user session. This section will discuss features of the window and its related functionality. The main replay window consists of a set of command buttons. A brief description of the functionality of these buttons is given below:

- **Options:** The `Options` button pops up an options window which allows the user to adjust various options related to replay mode. The details of Replay Options Window will be described later in this section.

- **Script:** When selected, this button pops up Script Window, described in Section 9, which displays the script being replayed.
**Model:** When selected, this button pops up Model window, described in Section 8, to display object hierarchies of the application under test.

**Next Step:** This button is used during single stepping or to get a user response when a `WaitEvent` with `WaitForUser` type is being replayed. During single stepping, selection of this button causes the execution of the next script command while when waiting for user its selection is interpreted as a user response. This button only becomes active when in single step mode or when user response is required.

**Replay/Pause:** This button allows the user to pause the replay of a script temporarily. When in PAUSE mode, XTester does not replay user actions. However the user can manually play with the application under test or any other application running on the server. When selected the label of this button and hence the mode of replay flips between *Replay* and *Pause*.

**Exit:** When selected, this button terminates the current replay session, saves the statistics file and exits.

Replay Main Window also displays various options currently selected and provides messages during replay in the bottom area of the window. Mode message in the bottom area specifies the options currently selected for the mode of replay. Modes of replay are described in detail later in this section. The Script File message displays the name of the file which contains the script being replayed. Another message string is displayed between the command buttons and bottom status messages which provided general information about the current activities and instructs user about the next possible events.

Figure 16 displays Replay Options Window. This window is logically divided in two halves, *Verify Options* and *Wait Options*. The *Verify Options* are related to verifying the behavior of the AUT during replay. XTester has the capability to replay scripts with/without comparing object information for behavior verification. User can choose to simply replay the script without doing any object information comparison or object information can be compared to the corresponding stored information. User can also select to replace the object information stored in the `.img` file with the current information grabbed from the screen. One of the three modes is selected by choosing the corre-
sponding option from `Mode` options. While verifying object information, XTester provide the choice to make more than one attempts to compare the current and the stored information. On each attempt, information is grabbed from the screen and compared with the information extracted from the `.img` file until the information is matched or the number of attempts exceed a certain number, provided by the user. `Max. Comparison` field allows the user to change the number of comparisons to be performed before announcing a mismatch.

![Replay Options Window](image)

**Figure 16. Replay Options Window**

XTester provides the capability to change the speed at which a script is being replayed. `Timing Options` half in the Replay Options Window specifies the options related to the speed of replay. The number in `Min. Delay` text field specifies the minimum number of seconds to wait before replaying the next event. However if the next event is connected to the current event, which is represented in the script by `SameGroupMask`, then the next event is immediately executed. The number in `Max. Delay` text field specifies the maximum number of seconds to wait for the target object to appear on the screen before declaring a failure. The number in `Max. Tries` text field specifies the number of consecutive failures to simulate a stored event before aborting the current script.

The `Single Step` button allows the user to toggle between the single step and the continuous mode. When in single step mode, the next script command is executed only when the user clicks on the `Next Step` button in the Main Replay Window.

### 8 Model Window

Figure 17 displays the model window which appears when the user clicks on the `Model` button in XTester’s main Capture or Replay windows. This window displays the window hierarchy of the application under test. In this paper the model window is displaying part
of the window hierarchy for the xman application. Actually there are two window hierarchies handled by Xtester, the static and the dynamic. The user can display the desired hierarchy by choosing the corresponding exclusive option from the button set in the middle of the top button area of the window. The static window hierarchy is created from a UIDL file. Details about the syntax of UIDL specification are given in Section 14. Although XTester does not rely on the UIDL file for normal capture/replay, it is required for writing manual scripts. XTester provides the capability to build this file incrementally. Each window has a name provided by the tester. This name will appear in the script file when the object is activated.

The dynamic tree hierarchy is the one that currently exists for the application under test. This tree represents all the windows that are currently created, though not necessarily mapped. Generally the static hierarchy represents all the unique objects in the application, no matter whether they currently exist or not. Once complete, the static hierarchy never changes during a particular session or across sessions. The dynamic hierarchy represents the current population of objects in the application. Each node in the dynamic hierarchy represents an application window which is currently alive. The nodes of the dynamic hier-
archy are in fact instances of the nodes of the static hierarchy and corresponding to a node in the static hierarchy, the dynamic hierarchy can have none, one or many instances at a particular time during a session.

When a node in the dynamic tree is clicked, it highlights the corresponding window in the application and gives information about that window in the footer area. When a node in the static hierarchy is clicked, it highlights all of the application windows which are instances of that particular node and displays their IDs in the footer area. It also pops up a dialog box, shown in Figure 18, so that the user can modify the information about the node. This feature is particularly useful when the UIDL file is being built. Figure 18 shows the dialog box. The External Name is the name given by the tester to the corresponding window. The Internal Name describes the internal name of the window, if one has been assigned by the application. This name can also be a comma separated list of names if the application gives multiple names to the window. Object Type is a pulldown menu which allows the user to specify the object type related to the window. Object types include some of the common widgets available in existing toolkits. Effects is also a popup menu which contains all the possible events which can be exercised to the object. The user can select a particular event, like PressButton, and then describe in the text field a comma separated list of object names which are affected when the chosen object receives such an event.

Figure 18. Node Information Dialog Box

9 Script Window

Figure 19 shows the window which pops up when the Script button is pressed in the main capture or replay windows. The window is used to display the script which is being captured or replayed. The script is editable during capture mode so that the user can make modifications to the script being captured. During replay the script is not editable. The
Figure 19. XTester Script Window

Save button in the top left corner of the window is used to save the script in the window to the script file during capture mode. XTester also automatically saves the script at the end of a capture session. The Save button is grayed out during replay mode. The Hide button pops down the script window. Section 13 provides a brief description of all the commands which can appear in a script.

10 Writing Testing Tools

XTester provides a rich set of functions which can be used to build testing tools for X applications. This set of function include very high level functions like FollowUser(), which simulates an entire user session, to low level functions like PressKey() which simulates pressing a single keyboard key. A full set of these functions and their brief introduction is included in Section 12.
Figure 20 shows source code for caprep application discussed in the previous sections. The code includes two *.h files, xtlib.h and xtmacros.h. The program starts with

```c
/*
 * Caprep source code
 */

/************************ Include Files ************************/
#include <xtlib.h>
#include <xtmacros.h>

************************* Forward Declarations *****************
extern XTInfo *InitializeXTester();

************************ Start of Program **********************
main(argc, argv)
int argc;
char **argv;
{
XTInfo *hdr; /* XTInfo structure pointer */

/* Initialize XTester */
hdr = InitializeXTester(argc, argv);

/* Capture or Replay user session */
FollowUser(hdr);

/* Cleanup */
CloseTestingSession(hdr);
}
************************** End of the program ******************
```

**Figure 20. C code for caprep**

initializing XTester library with a call to InitializeXTester(). This routine parses the command line options and returns a pointer to an XTInfo data structure. The next routine, FollowUser(), simulates a complete user session. If caprep is started in capture mode then this routine starts capturing the current user session until the user decides to exit from the session. If in replay mode, FollowUser(), replays a script from beginning to the end. CloseTestingSession() saves statistics collected by the
current session, closes all the opened files and connection which are related to the current user session and does cleanup to prepare XTester to exit.

Figure 21 provides another sample C program which simulates user actions on the xman main window to verify the response of all the buttons in the main window of xman. This program simulates some of the user actions and demonstrates the use of object comparison.

11 Resources and Command Line Options

In addition to the standard X resources and command line options, applications built on top of XTester provides some additional resources and options, which are described below. XTester distribution includes a default resource file, called XTester/XTester.Defaults.

**Startcmd:** This resource is a string which is used to start the application under test. Its corresponding command line option is `-cmd`. If no StartCmd is given then XTester application will wait for the user to manually start the application under test.

**AppName:** This resource describes the name of the application under test. Its corresponding command line option is `-app`.AppName is used to determine default names for application specific files accessed by the XTester. These files are common to all the sessions of a particular application. The details on building default names follows. Its default value is the first word i.e. `argv[0]` of StartCmd.

**Sess:** This resource describes the name of a particular session. Its corresponding command line option is `-sess`. It is used to determine the default names for various files created/accessed by the XTester which are session specific. The details on building default names follows. The default value for this resource is the `AppName` resource.

**TestDir:** This resource provides a directory name which is used for various files accessed and/or generated by XTester. Its corresponding command line option is `-dir`. The default value of this resource is the current working directory i.e. “./”.

**Mode:** This resource sets the testing mode for a session, i.e. capture or replay. Its corresponding command line option is `-mode`. The default value for this resource is replay.

**Pause:** This boolean resource determines if the application should start in Pause mode or not. Its corresponding command line option is `-pause` and default value is `False`. It is particularly useful in replay mode when the user wants control over the beginning of event replays.
Resources and Command Line Options

Figure 21. An example test script for xman

```c
/********************** Include Files ***********************/
#include <xtlib.h>
#include <xtmacros.h>

/********************** Forward Declarations *****************/
extern XTInfo *InitializeXTester();

/******************* Start of Program ***********************/
main(argc, argv)
int argc;
char **argv;
{
    XTInfo *hdr; /* XTInfo structure pointer */

    hdr = InitializeXTester(argc, argv);

    /* Verify if Xman window appeared */
    if (VerifyObject(hdr, "Xman-1") == FAILURE)
        return(FAILURE);

    /* Click on Manual Page button */
    ClickMBOOnObj(hdr, Button1, "Xman-1*ManualPage",
                  DontCare, DontCare, NoneMask)

    /* Verify if the manual page appreaded and its image is right */
    if (VerifyObject(hdr, "ManualPage-1") == FAILURE)
        return(FAILURE);

    /* Click on Help button */
    ClickMBOOnObj(hdr, Button1, "Xman-1*Help",
                  DontCare, DontCare, NoneMask)

    /* Verify if the Help page appreaded and its image is right */
    if (VerifyObject(hdr, "Help-1") == FAILURE)
        return(FAILURE);

    /* Click on Exit button */
    ClickMBOOnObj(hdr, Button1, "Xman-1*Quit",
                  DontCare, DontCare, NoneMask)

    /* Verify if the Xman window disappears or not */
    if (VerifyObject(hdr, "Xman-1") == SUCCESS)
        return(FAILURE);
}

/******************* End of Program **********************/
```
**Silent:** This boolean resource determines if the application should display the user interface and provide audio warnings and signals. Its corresponding command line option is `-silent` and default value is `False`. It is particularly useful when the tool is being used to replay a batch of scripts.

**SingleStep:** This boolean resource determines if the application should start in single step mode or not. Its corresponding command line option is `-singlestep` and default value is `False`.

**Sections:** This resource determines the sections of a script, which should be replayed. The value of this resource can be a combination of section names of the script, i.e., Entering, Core, Leaving and Entire. Entire represents all sections of the script and it is the default value for this resource.

**GroupInterval:** This resource determines the minimum amount of time, in milliseconds, between two consecutive events, to make them unrelated. Its corresponding command line option is `-groupinterval` and default value is 200 ms. It is provided to simulate clicks. If the duration between the two consecutive events is smaller than this value then they are considered to be related and are replayed immediately one after the other in replay mode.

**MinDelay:** This resource provides the minimum amount of delay, in seconds, between two consecutive events in the script. Its corresponding command line option is `-minwait` and its default value is 1 second. It can be changed from Replay Options window.

**MaxDelay:** This resource provides the maximum amount of time, in seconds, to wait for the target object during replay before giving up on the corresponding event. Its corresponding command line option is `-maxwait` and default value is 30 seconds. It can be changed from Replay Options window.

**MaxTries:** This resource provides the maximum amount of consecutive failures during replay before giving up on the script being replayed. Its corresponding command line option is `-maxtries` and default value is 5. It can be changed from Replay Options window.

**SelBy:** This resource provides the default method to select an object to capture its image or wait for it. Its corresponding command line option is `-selby` and default value is "cursor". The possible values for this resource are "cursor", "key", "ID", "Name" and "Area". It can be changed from Capture Options window which provides separate options to select object for Grab or WaitFor.

**Format:** This resource describes the default format in which capture information about an object should be stored. Its corresponding command line option is `-format` and default value is "pixmap". Its possible values are "Pixmap", "Attributes" and "Checksum". It can be changed from Capture Options window.
**MaxComps:** This resource provides an upper bound on the number of comparisons to be performed before announcing a difference in object information. Its corresponding command line option is `-maxcomps` and default value is 5. It can be changed from Replay Options window.

**WaitType:** This resource describes the default value for the type of waits in `WaitEvents`. Its corresponding command line option is `-waitfor` and its default value is "object". Its possible values are "object", "duration" and "user". It can be changed from Capture Options window.

**KeyMask:** This resource provides the set of keys which can be used to cancel an object selection when it is being selected by cursor. Its default value is "control". Its possible values are "control" and "shift". The corresponding command line option for this resource is `-keymask` followed by a comma separated list of possible values.

**Events:** This resource provides names of the events which should be captured. The names can be separated by any punctuation character. XTester is currently capable of capturing the following events and all of them are included as default for this resource. *ButtonPress, ButtonRelease, KeyPress, KeyRelease, MotionNotify, EnterNotify, LeaveNotify, MapNotify, UnmapNotify* and *ConfigureNotify*. The corresponding command line option is `-events` followed by a list of comma separated event names.

**DspScript:** This boolean resource indicates if the script window should be displayed automatically when the tool is started or not. Its corresponding command line option is `-dspscript` and default value is False.

**DspModel:** This boolean resource indicates if the model window should be displayed automatically when the tool is started or not. Its corresponding command line option is `-dspmodel` and default value is False. It is recommended to keep this resource value false for better performance.

**HorizontalSpace:** This resource provides the minimum horizontal distance in pixels between any two nodes in the graph. Its default value is 10.

**VerticalSpace:** This resource describes the minimum vertical distance in pixels between two consecutive levels in the graph. Its default value is 10.

**FullName:** This boolean resource provides if fully qualified object names should be built or only object name and the name of its top level window should be used to build the name of the corresponding object. Its default value is False.

**ObjSep:** This resource provides a character which will be used by XTester to separate name of an object from its parent when building full names in object hierarchy. Its corresponding command line option is `-objsep` and default value is ‘.’.

**InstSep:** This resource provides a character which will be used by XTester to separate the name of an object from its instance number. Its corresponding command line option is `-instsep` and default value is ‘-’.
WildSep: This resource provides a character which will be used as a wild card between object names to match multiple object. Its corresponding command line option is `-wildsep` and default value is `'*'`.

ModelFile: This resource provides the name of the file to read/write the UIDL specification for the application under test. The corresponding command line option for this resource is `-mod` and its default value is obtained by concatenating the `AppName` string with `.uidl` extension.

ScriptFile: This resource provides the name of the file to store the captured script during capture mode or read the script to be replayed during replay mode. Its corresponding command line option is `-scr` and default value is determined by concatenating the `AppName` string with `.scr` extension.

ImageFile: This resource provides the name of the file to store the captured images during capture mode or access the images to be compared with currently displayed images on the screen, in replay mode. The name should always be provided relative to the location of the corresponding script file. Its corresponding command line option is `-img` and default value is determined by replacing `.scr` extension of the `ScriptFile` with `.img`.

StatisticsFile: This resource gives the name of the file to which the statistics about the current test session will be stored. The name of the file is relative to the test directory specified with `TestDir` resource. Its corresponding command line option is `-stt` and default value is determined by concatenating the `AppName` string with `.stt` extension.

LogFile: This resource provides the name of the file to store results related to the behavior of the application under test. The name of the file is relative to the test directory specified with `TestDir` resource. Its corresponding command line option is `-log` and default value is standard output.

ErrFile: This resource provides the name of the file to store any errors caused by the environment or XTester itself but are not related to the application under test. The name of the file is relative to the test directory specified with `TestDir` resource. Its corresponding command line option is `-err` and default value is standard error.

12 Programming Interface

XTester is a library of functions written in the C language. These functions can be used by any C program as far as guidelines for using the library are followed. The major requirement for the use of XTester library, is to initialize it, like one have to initialize `Xt` by using `XtInitialize()`. This section introduces some of the major routines provided by XTester.
InitializeXTester(int *argc, char **argv)
This routine is very similar to XtInitialize() in its functionality. It allocates and initializes data structures and returns a pointer to the XTInfo data structure which acts as a central resource for XTester, like Display structure in XLib. This function also scans the command line arguments passed to it, extracts the arguments of its interest and modifies the argument count. It also establishes a connection with the actual X server, starts the virtual server, creates the user interface objects and starts the application to be tested. If no command string is specified to start the application under test then this routine waits until an application is started on the virtual server manually from the command line. It returns SUCCESS or FAILURE.

ConnectToServer(XTInfo *header, int *argc, char **argv)
This routine takes the values in the XTInfo structure and tries to establish a connection to the actual X server running on the machine. This routine calls OlInitialize() to initialize the OpenLook toolkit and also extracts resources specific to XTester. It returns SUCCESS or FAILURE.

StartVirtualServer(XTInfo *header)
This routine takes a pointer to XTInfo structure as argument and tries to start a virtual X server on the host machine. The application(s) to be tested are started on this virtual server. It returns FAILURE or SUCCESS.

StartAUT(XTInfo *header)
This routine extracts the starting command string from XTInfo structure, modifies it so that the application is started on the virtual instead of real X server, forks a process and starts the application from the forked process. It returns FAILURE or SUCCESS.

CloseTestingSession(XTInfo *header)
This routine saves the session statistics in the statistics file, closes all the open files and connections and does the other clean up jobs. It returns FAILURE or SUCCESS.

CreateAllUserInterfaceWindows(XTInfo *header)
This routine creates all XTester standard user interface windows. It returns FAILURE or SUCCESS.

DisplayInterface(XTInfo *header)
This routine displays the user interface objects on the screen depending on the values in the XTInfo structure. It returns FAILURE or SUCCESS.

HideUserInterface(XTInfo *header)
This routine pops down all the currently displayed XTester user interface objects. It returns FAILURE or SUCCESS.
FollowUser(XTInfo *header)

This routine takes over control of XTester, when called, it displays the appropriate windows, if not in SILENT mode, and then captures or replays a user session, depending on the mode of testing. It returns SUCCESS or FAILURE.

CaptureUserSession(XTInfo *header)

This routine captures a user session in appropriate .scr file. It does not display user interface windows for the capture session but only does the core capture functions. It returns SUCCESS or FAILURE.

ReplayUserSession(XTInfo *header)

This routine replays a user session from the specified .scr file. It does not display user interface windows for the replay session but only does the core replay functions. It returns SUCCESS or FAILURE.

ScanInterestingEvents(XTInfo *header)

This routine acts as a MainLoop() for both the tool and the AUT. It handles the messages queued both for the tool and for the AUT. It should be called frequently, especially in loops. It returns SUCCESS or FAILURE.

SimulateEvent(XTInfo *header, XTEvent *event)

This routine simulates the passed event. It calls one of the following event specific simulation routines. It returns SUCCESS or FAILURE.

SimulateFileHeaderEvent(XTInfo *header, XTEvent *event)

This routine is called when a FileHeader event is seen in a script. It opens the appropriate .img file, if required, and also performs any version specific translations. It also logs a message about starting a new session, in the .log file. It returns SUCCESS or FAILURE.

SimulateCoreEvent(XTInfo *header, int Type, int Detail, char *ObjName, int XCoord, int YCoord, int State)

This routine simulates the core X events during replay. It is called whenever a PressButton, ReleaseButton, PressKey, ReleaseKey or MovePointer commands are seen in the script. Its parameters represent event type, event detail, target object name, X and Y coordinate in the target object and the state of the modifier keys for the simulation of the event. It returns SUCCESS or FAILURE.

WaitEvent(XTInfo *header, int Detail, char *ObjName, int MinWait, int MaxWait)

This routine simulates a delay in the script. Detail field indicates the delay type and its possible values are WaitForObject, WaitForDuration and WaitForUser. Min.Time is the minimum time to wait, whether the wait condition is satisfied or not. Max.Time is the maximum time to wait before declaring a failure. Window gives the name of the object to wait for if the detail field is WaitForObject. It returns SUCCESS or FAILURE.
MovePointer(XTInfo *header, char *WinName, int X, int Y)
This function physically moves the cursor to location (X, Y) with respect to origin of the object named WinName. It returns SUCCESS or FAILURE.

MapWindow(XTInfo *header, char *WinName)
This function maps the named window on the screen. It returns SUCCESS or FAILURE.

UnmapWindow(XTInfo *header, char *WinName)
This function iconifies the named window. It returns SUCCESS or FAILURE.

ConfigureWindow(XTInfo *header, char *WinName, int X, int Y, int Width, int Height)
This function changes the configuration of the named window according to the passed values. It returns SUCCESS or FAILURE.

InputStringInObj(XTInfo *header, char *ObjName, char *String)
This function inputs String in the object ObjName. It returns SUCCESS or FAILURE.

SimulateWaitForEvent(XTInfo *header, WaitForEvent *event)
This routine is called by WaitForEvent routine described above. It simulates the kind of wait specified by the parameters of WaitForEvent structure passed to it. It returns a pointer to ObjInfo structure if successful and NULL in case of FAILURE.

SimulateCommandEvent(XTInfo *header, XTEvent *event)
This routine is called when an ExecCommand event is seen in a script. It executes the appropriate command and if a comparison is required then it compares the results with the stored ones. It returns SUCCESS or FAILURE.

SimulateMessageEvent(XTInfo *header, XTEvent *event)
This routine is called when a Message command is seen in a script. It simply logs the message string in the log file. It returns SUCCESS or FAILURE.

IsRelatedObject(XTInfo *header, Window ID)
This routine returns TRUE or FALSE depending on if the passed window ID belong to the AUT or not.

CompareObjClassNames(char *ObjName, char *ClassName)
This routine returns TRUE or FALSE depending on if the passed object name corresponds to the passed class name or not.

SaveStatisticsInfo(XTInfo *header, ObjInfo *TopNode)
This routine saves the statistics information in the statistics file, starting at TopNode in the object hierarchy. It returns SUCCESS or FAILURE.
InitDynamicModel(XTInfo *header)
    This routine initializes the dynamic model hierarchy of the AUT. It returns SUCCESS or FAILURE.

InitStaticModel(XTInfo *header)
    This routine initializes the static model hierarchy of the AUT. It reads in all the information from the .uidl file and builds the initial static graph. It returns SUCCESS or FAILURE.

ObjInfo *FillObjInfo(XTInfo *header, char *name, Window id, int x, int y, int width, int height, int state)
    This routine allocates and fills in an ObjInfo structure with the passed information. It returns a pointer to the allocated structure in case of SUCCESS or NULL in case of FAILURE.

ApplyToAllObjects(ObjInfo *Obj, int (*Func)(), char *Data)
    This routine applies the passed function with Data as a parameter to all the objects in a hierarchy. The function should accept two parameters, the first one points to the object on which it is being applied and the second one is Data. It returns SUCCESS or FAILURE.

AddObject(XTInfo *header, ObjInfo *Parent, ObjInfo *Obj)
    This routine adds an object as a child of Parent in the hierarchy. It returns SUCCESS or FAILURE.

DeleteObject(XTInfo *header, ObjInfo *Obj, int RootAlso)
    This routine deletes all the object in a hierarchy rooted at Obj. RootAlso flag indicates if the root of the hierarchy, i.e. Obj, should also be deleted or not. It returns SUCCESS or FAILURE.

SearchObjectByID(XTInfo *header, ObjInfo *TreeRoot, Window ID)
    This routine searches for the object with specified ID in the hierarchy rooted at TreeRoot. It returns a pointer to the ObjInfo structure of the found object or NULL if the object is not found.

SearchObjectByName(XTInfo *header, ObjInfo *TreeRoot, char *Name)
    This routine searches for the object with specified Name in the hierarchy rooted at TreeRoot. It returns a pointer to the ObjInfo structure of the found object or NULL if the object is not found.

GetNextNameElem(XTInfo *header, char *FullName, char *NameElem, char **RemName)
    This routine extracts the next name element representing a unique object from and string specifying the full name for some object. RemName is a pointer to the remaining string of FullName after removing the NameElem. It returns SUCCESS or FAILURE.
NextLineInScript(XTInfo *header)

   This function moves the cursor to the next line in the script. It returns SUCCESS or
   FAILURE.

LoadNewFileInScript(XTInfo *header)

   This function loads a new file in the script window. The script file name in XTInfo
   structure must be set to the appropriate file name before calling this routine. It
   returns SUCCESS or FAILURE.

AppendToScript(XTInfo *header, char *string)

   This function appends the passed string at the end of the script. It returns SUCCESS
   or FAILURE.

StoreEvent(XTInfo *header, XTEvent *event)

   This function takes information in the event structure and looks if the current mode
   of operation requires the storage of this particular event in the script. If yes, it trans-
   lates the event into human readable form i.e. script command, and stores in the script
   being generated. It returns SUCCESS or FAILURE.

IntToExtEvent(XTInfo *header, XTEvent *event, char Output)

   This function converts an XTEvent structure into one of the valid script commands
   and copies the converted command in the Output string. It returns SUCCESS or
   FAILURE.

RetrieveNextEventEvent(XTInfo *header, XTEvent *event)

   This function retrieves the next event from script file, translates it from external to
   internal format and copies the retrieved information in the passed XTEvent struc-
   ture. It returns SUCCESS or FAILURE.

ExtToIntEvent(XTInfo *header, XTEvent *event, char Output)

   This function converts a script command from external to internal format. It stores
   the converted information in the XTEvent structure. It returns SUCCESS or FAIL-
   URE.

ObjInfo *ObjNameToObjID(XTInfo *header, char *name, Window *ID)

   This function provides an ID for the named object. If named object exists, then this
   routine returns a pointer to the ObjInfo structure for the object, otherwise it
   returns NULL and sets ID to 0.

ObjInfo *ObjIDToObjName(XTInfo *header, Window ID, char *name)

   This function provides a name for the object whose ID is given. If the object exists,
   this routine returns a pointer to the corresponding ObjInfo structure, otherwise it
   returns NULL and Name string is cleared.

BuildFullName(XTInfo *header, ObjInfo *Obj, char *Name)

   This function provides a full name for the object corresponding to Obj. It returns
   SUCCESS or FAILURE.
VerifyObjectInfo(XTInfo *header, char *ObjName)

This function captures or verifies the object information following the current options. In capture mode this function grabs the information about the named object while in replay mode it extracts the next object information from .img file, grabs the current information from the screen and compares the two. It returns 0 for SUCCESS or 1 for FAILURE.

CompareObjectInfo(XTInfo *header, ObjInfoEvent *event)

This function captures information about the object according to the values in the event structure and compares the grabbed information with the information passed in the event. It returns 0 for SUCCESS or 1 for FAILURE.

GetSelectedArea(XTInfo *header, SelStruct *selOpts)

This function allows the user to select area or object on the screen by the method passed in selOpts structure and returns its name, ID, coordinates and/or dimensions in selOpts. It returns SUCCESS or FAILURE.

getRectangle(XTInfo *header, SelStruct *selOpts)

This function allows the user to select area by stretching a rectangle around it. The information about the selected area is copied in the appropriate fields of selOpts. It returns SUCCESS or FAILURE.

getWindow(XTInfo *header, SelStruct *selOpts)

This function allows the user to select a window from the screen by clicking a mouse button on it. The information about the selected area is copied in appropriate fields of selOpts. It returns SUCCESS or FAILURE.

getWindowWhenKeyIsPressed(XTInfo *header, SelStruct *selOpts)

This function allows the user to select a window from the screen by clicking a key on it. The information about the selected area is copied in appropriate fields of selOpts. It returns SUCCESS or FAILURE.

compute_checksum(long *result, unsigned char *buffer, unsigned int size)

This function compute checksum for the data in buffer of size size and copies the result in result.

13 Script Commands

This section includes a brief introduction to all the commands which can be captured in a script. Some of the commands which are similar in their structure and/or function are grouped together. The user can insert these commands in the script manually also.
FileHeader (version, time, creater name, start Command, Image File, Misc. Info)

FileHeader designates the beginning of the script and contains information such as version number of Xtester, date of creation of the script, author of the script, command string to start AUT, and name of the .img file. A miscellaneous field is present which can be filled in by the tester for a brief description of the test or other miscellaneous information.

PressKey (key, window, x_coord, y_coord, state)
ReleaseKey (key, window, x_coord, y_coord, state)
PressButton (button, window, x_coord, y_coord, state)
ReleaseButton (button, window, x_coord, y_coord, state)

These commands indicate which mouse/key button was pressed/released, upon what window at what (x,y) coordinates relative to the upper left corner of the window, and state of the modifiers at the time button/key was pressed/released. State field also indicates if the event is related to the previous event or not.

MovePointer (detail, window, x_coord, y_coord, state)
EnterWindow (detail, window, x_coord, y_coord, state)
LeaveWindow (detail, window, x_coord, y_coord, state)

These commands represent motion related events. Detail field provides some event specific information (i.e. Motion was a hint or normal motion event, etc.), while the other fields are similar to the ones described for the previous commands. XTester does not capture all the motion specific events but only stores ones which are absolutely required for a successful replay.

MapWindow (window)
UnmapWindow (window)

MapWindow and UnmapWindow represents mapping or unmapping of window by the window manager or another application. Normal mapping and unmapping, done by the application under test, is not captured.

ConfigureWindow (window, x_coord, y_coord, width, height)

ConfigureWindow represents the movement or resizing of window by window manager or another application. x_coord, y_coord, width and height represent new location and size of the window.

WaitEvent (detail, window, Min. Time, Max. Time)

WaitEvent represents delay in a script. Detail field indicates the delay type and its possible types are WaitForObject, WaitForDuration and WaitForUser. Min.Time is the minimum number of seconds to wait, whether the wait condition is satisfied or not. Max.Time is the maximum number of seconds to wait before declaring a failure. Window gives the name of the object to wait for, if the detail field is WaitForObject.
ObjectInfo(format, window)

ObjectInfo verifies the object information. Format indicates whether the information is captured as Pixmap, Checksum or Attributes. Window is the name of the corresponding window.

ExecScript(Script, Image, Sections)

ExecScript provides a way to invoke a script from another script. The Script parameter provides the name of script file to be invoked. The Image parameter is the file name which will be used for verification data. The file name in the Image should always be relative to the location of the script file being called. The Sections give the sections of the script which should be exercised. After the completion of the invoked script, the next command in the invoking script is executed. If the string passed in Data parameter is “SameFile”, then the image file of the calling script is used in the called script also.

ExecCommand(command, mode)

ExecCommand provides a way to verify or modify environment during playback. The mode parameter indicates if the results obtained from the command should be verified or not. The string provided in the Command parameter is executed and if the mode parameter indicates verification, the results of the command are either stored in .img file, if in capture mode or compared to the already stored results, if in replay mode.

Message(“audio filename”, “messagestring”)

Message command allows the user to insert messages in a script. The messages can be audio or string messages. The first argument supplies the audio file name and the second argument gives the message string. The audio message is played if the name of the file is provided, the file exists, and the machine has audio capabilities. The string is printed in the log file and in the message area of the main replay window. It provides the user a way to make the log file more legible.

EndOfSession(i)

EndOfSession indicates the end of the i th session in the currently active script.

% Start of a section

A line beginning with a percent sign (%) indicates start of a new section of the script. Three sections, Entering, Core, and Leaving, are supported in the current version of XTester.

# Comments

It is possible to add comments in a script. A line beginning with # (pound sign) is considered to be a comment and is ignored by the script during playback. Blank lines are also ignored.

A click is represented by adjacent occurrences of PressButton and ReleaseButton where the state field in the ReleaseButton event contain SameGroupMask in addi-
tion to other information. For example, the following script commands represent a double click on Help button in main window of xman.

PressButton(Button1, Xman.Form.Help, 40, 45, NoneMask);
ReleaseButton(Button1, Xman.Form.Help, 40, 45, Button1Mask|SameGroupMask);
PressButton(Button1, Xman.Form.Help, 40, 45, SameGroupMask);
ReleaseButton(Button1, Xman.Form.Help, 40, 45, Button1Mask|SameGroupMask);

A click-and-drag is represented by a combination of PressButton, MovePointer and ReleaseButton commands. For example the following commands represent selection of text in xman’s Help window by click-and-drag.

PressButton(Button1, Help.Form.SCWindow, 99, 29, NoneMask);
MovePointer(Normal, Help.Form.SCWindow, 0, 0, Button1Mask|SameGroupMask);
ReleaseButton(Button1, Help.Form.SCWindow, 0, 0, Button1Mask|SameGroupMask);

The left mouse button is initially pressed while the cursor is over window Help.Form.SCWindow with offset (99,29). The cursor is then moved to window Help.Form.SCWindow at location (0,0), and then the mouse button is released.

14 UIDL Description

The UIDL file for an application describes a hierarchy of objects, starting with root which represents the root window of the display. Each object description is organized in three sections, Parameters, Effects and Children. All the sections and all the entries in a section are optional. The parameters section describes parameters of the object, like its internal name, full name, object type, etc. The effects section describes the potential effects of different events on the object to the other objects in the application. The children section lists all the children of the object.

Figure 22 provides a UIDL description for a trivial, but well known X application, xterm. xterm is a very simple application and has only one top level window. For simplicity, if we ignore the menus associated with an xterm, then the total number of X windows created by the xterm application are two. One of them is the top level window and the default name assigned to it by xterm is "xterm". The other window displays the actual text and is a child of that top level window. The UIDL description in Figure 22 displays this information. It starts from the root window. The Root window is given an Internal Name of Root by default, but the user may care to change this name and perhaps supply some platform specific information. This would help to translate images across different types of screens. This application has a single top level window whose external name is supplied by the tester to be in this case "Xterm". As Xterm is a direct child of Root, its FullName parameter is also "Xterm" while its Internal name is provided by the application and in this case it is "xterm". As this is the top level window, the ObjectType parameter of the window is MainWindow. This window is completely overlapped by its only child, which displays the text so the Effects section for this win-
Figure 22. UIDL Description of \texttt{xterm}
window has been ignored. In the Children section of Xterm, the only child of Xterm is described with name TextArea. As TextArea is a child of Xterm, its FullName parameter is "XTerm.TextArea". TextArea is not a top level window and so has not been given any name by the application, its InternalName parameter is ignored. TextArea is a text widget so its ObjectType parameter is Text. It accepts all the core events and the only object effected by these events is TextArea itself. However if the user types "exit" then the top level window Xterm is also affected. This information is presented in the Effects section. TextArea does not have a child, so the children section is ignored.

15 Problems

X window system provides a very flexible environment for building graphical user interfaces. It does not impose any restriction on the presentation, and/or behavior of an application. Such flexibility makes it impossible to build a tool which is capable of addressing elements of the graphical user interface by reference, and still can deal with all the low level details of presentation and behavior. XTester has been built to operate at the X protocol level. It addresses an element of a graphical user interface by its symbolic name, and captures and replays events with reference to the elements of the graphical user interface, instead of absolute screen coordinates. It captures the minimum possible information, necessary to replay the user events. XTester has its limitations which makes it impossible to successfully reproduce a user session under certain circumstances. Some of these limitations are described below:

- The window IDs in X are dynamic and vary from session to session. XTester tries to associates a symbolic name with each window ID. Some special characters are used in building these symbolic names. If the name of a window in an application contains any of these special characters, then the mapping between the window names and IDs cannot be performed reliably. These special characters specified for the replay of a script should be exactly the same as used during the capture of the script. During the replay of a script, most of the messages about the failure to find a particular object are related to this limitation of XTester. One must choose appropriate special characters which do not appear in any of the window names in the application under test. The values for these special characters can be specified either by setting the corresponding X resources to their appropriate values, or by using appropriate command line options. The X resources which represent these special characters are ObjSep, InstSep, and WildSep, and the corresponding command line options are -objsep, -instsep, and -wildsep. The XTester model window can be used during capturing to make sure that none of the objects appearing in the widget hierarchy of the application contain any of the special characters.
• The association of window names to internal IDs is done by using the top-level window names along with the location of the window in the hierarchy. Names are always stored as absolute names starting from the topmost objects. To make this association unique, XTester requires that each top-level window must have a name and these names should be unique. XTester works without this restriction if no manual tests are created and all the scripts are captured and replayed in separate invocations of XTester. This restriction becomes compulsory when a user is writing test scripts manually or scripts captured in different XTester sessions are to be replayed in a single, continuous session of XTester. XTester builds a name for a particular object by traversing along the path from root down to the object. If the application is modified so that the object hierarchy changes, then some of the existing tests will be invalid and must be modified.

• XTester provides synchronization only at the window level. If the window required for the replay of an event is not present on the screen, XTester will wait for it to appear before replaying the event. XTester is not capable of synchronizing events which depend on a particular image displayed in a window. For example, if the replay of an event depends on the image displayed in the window, then XTester will not be able to synchronize events reliably. It also implies that XTester cannot reliably perform scrolling and other functions whose outcome depends on the system load. The effect of the synchronization problem can be minimized by increasing the default minimum delay between consecutive events. The default minimum delay can be changed by specifying an appropriate value for the MinDelay X resource, or for -minwait command line option. It can also be changed from the options window in replay mode.

• X window systems provides a rich set of messages which are exchanged between the X client and the X server. The volume of communication between the X server and a client is tremendous. It is not feasible to capture each and every detail of the communication. XTester captures and replays only the core events, i.e. events produced by user like ButtonPress, ButtonRelease, KeyPress and KeyRelease. This approach reduces the size of the data which needs to be captured, and makes the captured script legible. However the applications which make use of other events may not be replayed faithfully by XTester. For example, the current version of XTester does not work for second level menus.

• X provides a mechanism to allow a particular client to grab server resources. XTester tries to override such grabs, to keep its communication open with the server. If the behavior of an application depends on such a grab, then the events related to the grab cannot of replayed faithfully.

• The current version of XTester performs image, attribute or checksum comparisons. However images are not compared for 100% match. If two images are 95% similar then the they are considered to be the same. Checksum is the most useful on images but current version always computes checksum on attributes.
Images that are captured using a given monitor will likely not work on another monitor, even one supplied by the same vendor. E.g. if images are captured on a Sparc station 19” color monitor, then the same test running on a Sparc station 19” monochrome monitor will produce different images. Hence XTester requires that although the same test scripts can be used across multiple platforms, the verification data must be platform specific.

While Xtester is capable of simulating cursor movements during replay but it never physically moves the cursor until the user uses the XT library function `LocatePointer()` to physically relocate the cursor. It provides the full control over the application during replay and one can mix ones own actions with the stored script events during replay. However sometimes it is annoying for the user when application under test grabs the input device.

16 Acknowledgments

Some of the XTester code is borrowed from Xmon, a public domain utility to monitor communication between an X server and its clients, after some modifications. Xmon was also used extensively for debugging and analysis during XTester development. We are thankful to the developers of Xmon and acknowledge their efforts in developing such a useful application. We are also thankful to the developers of xgrabsc application, also in the public domain. We borrowed some of the routines from xgrabsc and modified them to grab objects from the screen. Special thanks to Douglas Young and John Pew for writing great books on OpenLook and Motif programming, which cover from very basic concepts to writing customized widgets. The tree widget in their book gave us a good start in visually representing the UIDL hierarchy.