

Replay Xcessory™ *User's Guide*



**Integrated Computer
Solutions Incorporated**

Copyright © 1995-2010 Integrated Computer Solutions, Inc.

Integrated Computer Solutions, Inc. (ICS) reserves the right to make changes in specifications and other information contained in this publication without prior notice. The reader should in all cases consult ICS to determine whether any such changes have been made.

This Manual contains proprietary information that is the sole property of Integrated Computer Solutions, Inc. This Manual is furnished to authorized users of Replay Xcessory solely to facilitate the use of Replay Xcessory as specified in written agreements.

No part of this publication may be reproduced, stored in a retrieval system, translated, transcribed, or transmitted, in any form, or by any means without prior explicit written permission from Integrated Computer Solutions.

The software programs described in this document are copyrighted and are confidential information and proprietary products of Integrated Computer Solutions, Inc.

CenterLine-C, CenterLine-C++, CodeCenter, CenterStage, CodeVision, ObjectCenter, QualityCenter, QC/Coverage, QC/Recall, QC/Replay, TestCenter, and Replay Xcessory and are trademarks of Integrated Computer Solutions, Inc.

Motif and UNIX are registered trademarks of The Open Group.

PostScript is a registered trademark of Adobe Systems, Inc. in the USA and other countries.

Tcl is a copyright of The Regents of the University of California.

X Window System and X11 are trademarks of the Massachusetts Institute of Technology.

Other trademarks mentioned in this document are trademarks or registered trademarks of their respective holders.

Integrated Computer Solutions, Inc.

54 B Middlesex Turnpike, Bedford, MA 01730

Tel: 617.621.0060

Fax: 617.621.9555

E-mail: info@ics.com

WWW: <http://www.ics.com>



Preface

Overview

This manual is a guide to the use of the Replay Xcessory product. It also includes reference information.

What This Manual is About

We have organized the manual as follows:

- *Chapter 1* provides an overview of the Replay Xcessory product capabilities and usage model. It also includes a brief example of how to run tests of your application in Replay Xcessory.
- *Chapter 2* provides a detailed description of concepts you need to understand all of Replay Xcessory's features.
- *Chapter 3* focuses on the Replay Xcessory Test Manager, describing how to set up your environment and how to create and use test packages.
- *Chapter 4* describes the new Replay Xcessory Test Manager and how it differs from the old Test Manger.
- *Chapter 5* describes how to record and play back tests using the Replay Xcessory record and play control panels. It also describes the use of the command line interface to the record and play features.
- *Chapter 6* is an introduction to Tcl (tool command language), which is used for writing Replay Xcessory test scripts.
- *Chapter 7* provides information about the commands used by the Replay Xcessory command language—the language used to create Replay Xcessory scripts. This command language is based on Tcl.
- *Chapter 8* describes extended commands of the Replay Xcessory command language.
- *Chapter 9* tells you how to use the Replay Xcessory Script Debugger, and provides reference information for all the debugger commands.
- *Chapter 10* provides information about several advanced topics, such as the use of custom widgets, that go beyond the every day usage model.

Using Sample Programs

See the Replay Xcessory *Quick Start* guide for a brief introduction to the use of the sample programs that are part of Replay Xcessory. See Chapters 1-5 of this book for additional details.

What You Should Know Before Starting

This book is designed for readers who are familiar with UNIX/Linux, X-based applications, and the use of an OSF/Motif[®] Graphical User Interface.

For More Information

See the Replay Xcessory *Quick Start* guide for information that will help you get started with Replay Xcessory quickly.

See the installation notes for Replay Xcessory for information about installing this product and for troubleshooting tips concerning the license manager.

See the *Release Notes* for information specific to a particular release of Replay Xcessory, such as system requirements and supported platforms, and additional information needed to install Replay Xcessory.

Documentation Conventions

Unless otherwise noted in the text, we use the following symbolic conventions:

literal names	Bold words or characters in command descriptions represent words or values that you must use literally.
<i>user-supplied values</i>	Italic words or characters in command descriptions represent values that you must supply. Italic words in text also indicate the first use of a new term, or emphasis
sample user input	In interactive examples, information that you must enter appears in this typeface .
output/source code	Information that the system displays appears in <i>this typeface</i> .
...	Horizontal ellipsis points indicate that you can repeat the preceding item one or more times.



Preface	v
List of Figures	xiii
List of Tables	xvii
Chapter 1—Getting Started	
Introduction	2
License File Notes	2
Replay Xcessory Capabilities	3
Replay Xcessory Roadmap	3
Tutorials	6
Chapter 2—Concepts of Operation	
Introduction	30
Replay Xcessory architecture	30
Results Verification	41
Event Synchronization	44
Chapter 3—Replay Xcessory Test Manager	
Introduction	48
Setting up the Environment	48
Starting the Test Manager	51
Test Suite Commands	51
Test Package Commands	57
Test Case Commands	65
Chapter 4—The New Replay Xcessory Test Manager	
Introduction	68
Starting the New Replay Xcessory Test Manager (rtm)	68
Preferences	73
Replay Xcessory Tag Manager	79
Vrdump Editor Window	94
Chapter 5—Record and Play Sessions	
Introduction	98
Preparing Applications for Replay Xcessory	98
Understanding the Replay Xcessory Property Files	101
Controlling Session Properties	103
Controlling Snapshot Scope and Granularity	122
Image Snapshots	128
Record Control Panel	130

Recording Widget Tags	142
Tag File Generation	145
Streamlined Recorded Scripts	148
Play Control Panel	150
Replay Xcessory Driver — Command-Line Interface	161
Obtaining a Test Suite Report.....	167

Chapter 6—Introduction to the Scripting Language

Introduction	170
Command and Script Basics.....	170
Controlling Character Interpretation	173
Variable Manipulation Commands	176
Expressions	177
Control Flow Commands	180
Procedures.....	184
List Commands	187
String Manipulation Commands	191
File Access Commands	193
Extended Tcl Commands.....	196

Chapter 7—Replay Xcessory Command Language

Introduction	198
User Interaction Commands	198
Test Management Commands.....	204
Widget Information Commands	206
Session Management Commands	213
Initialization Scripts	219
Accessing Command-Line Arguments.....	219
Accessing Run Time Parameters	219

Chapter 8—Replay Xcessory Extended Commands

Introduction	224
Menu Selection Commands.....	224
Scroll Bar Commands	227
Scale Commands.....	229
List Commands	230
Text Manipulation Commands	232
Tabs Control Commands	234
XmTree Control Commands	235
XmContainer Control Commands	235

Chapter 9—Script Debugger

Introduction	238
Debugger Interfaces	238
Debugger Commands.....	247
Enhanced Tcl Debugger	252
Debugging Using the Command-Line.....	259

Chapter 10—Advanced Topics

Using Terminal Emulators	262
Using VNC, Nested and Virtual X Servers.....	263
Monitoring Background Tests	268
Using Replay Xcessory with Source Debuggers	268
Using Custom Widgets	269
Using Type Converters	269

Index..... 279

List of Figures

- Figure 1 Test Manager Window: Suite Directory Listing 7
- Figure 2 Test Package Dialog Box 8
- Figure 3 A New Test Package 9
- Figure 4 Deleting a Test Package 9
- Figure 5 Record Control Panel, Description Tab 10
- Figure 6 Record Control Panel, Settings Tab 11
- Figure 7 The Calculator 12
- Figure 8 Record Control Panel, Record Tab 13
- Figure 9 Test Package After Recording 14
- Figure 10 Test Package, “Show All” 15
- Figure 11 Play Control Panel 16
- Figure 12 Examining the Report File 18
- Figure 13 Examining the Script File 19
- Figure 14 Examining the Baseline and Result Snapshots 20
- Figure 15 Replay Xcessory Architecture 30
- Figure 16 Test Package Report for Calculator Example 34
- Figure 17 Widget Hierarchy 38
- Figure 18 Test Suite Window 51
- Figure 19 Change Test Suite Directory 52
- Figure 20 Test Package Dialog Box 53
- Figure 21 Open Test Package and its Icons 54
- Figure 22 Utility Preferences 56

- Figure 23 Test Package Window for **xmcalc** 58
- Figure 24 Batch File Creation Dialog 61
- Figure 25 Script and Snapshot View Panes 62
- Figure 26 Report View Pane 64
- Figure 27 Test Case Popup Menu 65
- Figure 28 Test Case Icon 65
- Figure 29 The New Test Manager 68
- Figure 30 Test Manager File Menu 69
- Figure 31 New Test Suite Dialog Box 70
- Figure 32 Open Test Suite Dialog Box 71
- Figure 33 New Test Package Dialog Box 72
- Figure 34 Utility Preferences Dialog Box 73
- Figure 35 Edit Menu 75
- Figure 36 Record/Play Menu 75
- Figure 37 Batch File Dialog Box 77
- Figure 38 Snapshot Pane 78
- Figure 39 Tcl File Editor 79
- Figure 40 Tags Editor Window 80
- Figure 41 Tag Button After Modification 82
- Figure 42 Tag Button After Modification Confirmation 83
- Figure 43 Information for “fake_widget” 84
- Figure 44 Tag Manager Sub-Menu 85
- Figure 45 Start Custom App File Selection Box 86
- Figure 46 Tag Manager Sub-Menu After the Application Starts 86
- Figure 47 xmcalc Flash Widget 87

Figure 48 Failed Widget Search 88

Figure 49 Add Widget Dialog Box 89

Figure 50 Widget Deletion Warning Box 90

Figure 51 Widget After Deletion 91

Figure 52 Rename Widget Dialog Box 92

Figure 53 Save Changes Warning Box 93

Figure 54 Vrdump Editor Window 94

Figure 55 Session Properties Window, General Tab 104

Figure 56 Session Properties Window, Record Tab 105

Figure 57 Session Properties Window, Playback Tab 106

Figure 58 Session Properties Window, Recognition Tab 107

Figure 59 Session Properties Window, Recognition Tab (additional fonts) 108

Figure 60 Replay Properties Page 119

Figure 61 Snapshot Properties Window for **xmcalc1** Snapshot 124

Figure 62 Record Control Panel, Description Tab 131

Figure 63 Record Control Panel, Settings Tab 132

Figure 64 Record Control Panel, Record Tab 133

Figure 65 Record Control Panel While Recording 138

Figure 66 Snapshot Specification Entry 141

Figure 67 Learn Widget Tag Name 143

Figure 68 Displaying a Tagged Widget During a Record Session 144

Figure 69 Delete Tag Dialog Box 145

Figure 70 Play Control Panel, Settings Tab 151

Figure 71 Play Control Panel—While Playing 157




Figure 72 Play Control Panel- Debug Tab	158
Figure 73 Snapshot Mismatch Window	159
Figure 74 Image Mismatch Window	160
Figure 75 Text Area	232
Figure 76 The Test Manager Main Window	239
Figure 77 The Test Package Window	239
Figure 78 The Main Debug Play Control Panel	240
Figure 79 The Replay Xcessory Debugger Window	243
Figure 80 The Replay Xcessory Debugger Window Record View	244
Figure 81 Ask Dialog	245
Figure 82 Replay Xcessory Tcl Debugger Showing Line Numbers	253
Figure 83 The Debugger Popup Menu	255
Figure 84 Changing one line of Tcl Code	256
Figure 85 Changing Multiple Lines of Tcl Code	257
Figure 86 Popup Menu	257
Figure 87 The Tcl Editor	258

List of Tables

Table 1 Property Filenames 102

Table 2 Debug Play Control Panel Entities 241

Table 3 The Replay Xcessory Debugger Window Buttons 245



Getting Started

Overview

After a brief overview of Replay Xcessory's capabilities, short tutorials in this chapter will take you step-by-step through record and play and programmatic testing sessions.

Introduction

Developing automated tests for Graphical User Interface (GUI) applications is both complex and challenging. One reason for this complexity is that—unlike command-line applications—the user, rather than the application, controls the interaction. In addition, many changes in an evolving user interface, such as fonts and color, may not logically affect results but may complicate result verification.

Replay Xcessory is part of ICS's product family of tools that accelerate Motif development. Replay Xcessory supports the automated testing of X Toolkit applications, including applications that use OSF/Motif. Replay Xcessory systematically lessens the difficulty of the automation process by removing the need for manual tests, while avoiding the pitfalls of other automated testing technologies which use attributes that are not portable, for example, image dumps and absolute screen coordinates.

License File Notes

Replay Xcessory binaries expect the license file to be placed in the HOME directory of the user that uses the application, with a name of form ``uname -n`.lic`. However, the name of the license file only has a logical meaning and is intended to be used to determine the license file station easily.

In the case that the license file is placed in some other place, the `ICS_LICENSE_FILE` environment variable can be used to specify the license file location as:

```
export ICS_LICENSE_FILE=/work/this_station_name.lic
```

for bash.

Replay Xcessory Capabilities

Replay Xcessory can be used to:

- facilitate the generation and organization of automated tests within an integrated test environment
- enable the tester to work with high-level GUI elements, such as dialog boxes and push buttons, rather than with bitmaps and coordinates
- provide for flexible verification of screen results; for example, tests can ignore physical rearrangement of GUI elements, changes in background color, and changes in font size or style of text. This flexibility enables the tester to focus on relevant elements of the interface specific to the current test application, rather than the underlying windowing interface
- support both record and playback as well as programmatic testing via a non-proprietary scripting language called Tcl
- provide test portability for applications that will be released on multiple platforms
- provide for ease of test modification and extensibility, which enables Replay Xcessory tests to remain useful over the entire life cycle of an application, and throughout the extensive revisions typical of most software products

Replay Xcessory Roadmap

Descriptions of the following topics are written at the same level and assumed experience of the person who will be using the following tutorial. Refer to Chapters 2-5 for a thorough explanation of the same topics. The following sections provide an overview of:

- the Replay Xcessory usage model
- Replay Xcessory Test Manager
- command line interface

Following these descriptions are tutorials that provide step-by-step instructions for:

- setting up and starting the Test Manager
- running a record and play session
- programmatic testing

Screen shots of Replay Xcessory windows, panes, menus, and icons, as well as program samples, provide a quick and easy way to familiarize yourself with the Replay Xcessory product.

Usage Model

As an application evolves, retesting can be performed periodically using Replay Xcessory. The new version of the application is replayed using the original input actions that have been recorded in a script. The results of the new tests are programmatically compared with previously verified results (known as *baselines*), and the success of the new tests is reported in a Subtest Report.

Capturing Input Actions—Scripts

Replay Xcessory operates by capturing input actions as the user operates a mouse and keyboard to interact with application objects. Actions and objects on the application's processes are captured by Replay Xcessory commands using Tcl (the Replay Xcessory scripting language). Specifically, command names represent input actions; for example:

- **click** (a single click of the mouse button)
- **dblclick** (a double click of the mouse button)
- **drag** (drag motion of the mouse button)
- **text** (for text entry)

Command objects are the widgets of the application's interface—an **OK** push button, a **Question** dialog box, an **Open ...** menu item.

Note: Most applications are implemented as single UNIX/Linux system processes. However, some complex applications may be implemented as a set of cooperating processes. For example, some applications launch a help browser when the Help button is selected. Testing such an application involves multi-process testing.

Capturing Application Results— Snapshots

Just as input actions are captured by the script, an application's results are captured by *snapshots*. At *verification points* during the application's execution, the tester must verify the correctness of results displayed on the screen. Replay Xcessory snapshots provide one basis for result verification.

The snapshot is a “picture” of the screen, although not necessarily a literal image. Replay Xcessory supports two types of snapshots: image and widget snapshots.

- An *image snapshot* is a literal picture of the relevant screen area using the standard **xwd** format to capture the screen.
- A *widget snapshot* is a logical picture; that is, it provides the values of selected widget attributes as opposed to an actual screen image.

The **Snapshot** button is pressed to request a snapshot. This process designates verification points that control where results will be verified during a play session. By editing a control file (the **.Vrdump** file), the granularity of the snapshots can be controlled by specifying whether a specific resource should be dumped for each widget class or widget instance.

Verifying Results— Baselines

When the collection of snapshots from a record session have been determined to be correct, they can be used as *baseline* snapshots. Baseline snapshots serve as a standard against which snapshots in future regression tests will be judged. Other methods used to verify correctness include checking for expected results programmatically and querying the process itself (see Chapter 2 for more information on verification methods).

Updating Baselines

If a mismatch is found when comparing baselines and results, it is necessary to determine whether the mismatch points to an error in the new results—which would require a correction in the process under test—or whether the mismatch resulted from the baseline being obsolete or incorrect, in which case it is necessary to update the baseline.

Replay Xcessory Test Manager

The Replay Xcessory Test Manager provides a graphical user interface that can be used to organize and conduct tests. The Test Manager enables you to:

- create a *test case*—an individual testing instance, usually a single record and play session, several of which make up a test package
- create a *test package*—a directory that organizes the files and directories of a set of test cases
- navigate among *test suites*—a directory that contains a number of related test packages
- select and open a test package within a test suite
- start a record or play session
- customize Replay Xcessory options
- view multiple, related elements of a particular test case—its script, baseline and result snapshots, and related information

Although the Test Manager provides a convenient facility for managing test suites, the core Replay Xcessory functionality can be accessed without using the Test Manager; this makes it possible to use Replay Xcessory in conjunction with custom or third-party test harnesses, or in batch mode.

Command-line Interface

In addition to running under the Test Manager, record and play sessions may also be utilized using a command-line interface. The command-line interface is especially important for its support of unattended Replay Xcessory sessions.

Tutorials

The following tutorials provide a hands-on introduction to Replay Xcessory without the need to understand the descriptions provided in the rest of the user's guide.

Setting Up and Starting the Test Manager

Some environmental variables need to be set or added permanently to your **.profile**, **.bash_profile**, or **.cshrc** file before running the Replay Xcessory user command interface. The examples in this section are designed for **sh** or **ksh** users. Users of **csh** should make the appropriate adjustment.

To use Replay Xcessory you must make sure that your application links to the version of the Xt library supplied by ICS for use with Replay Xcessory, in case it is linked statically.

Tutorial 1—A Record/Play Session

The tutorial leads you through record and play sessions of a calculator program, covering the following topics:

- setting up and starting the Replay Xcessory Test Manager
- creating a test package
- conducting a record session (creating a test case)
- conducting a play session
- examining the report file and other output

Starting the Replay Xcessory Test Manager

Start the Replay Xcessory Test Manager in the background by entering the following command:

```
replaytm
```

The **replaytm** command is in the **\$REPLAYHOME/bin** directory, which must be in your path for you to start Replay Xcessory Test Manager. **replaytm**, allows the name of a test package to be specified on the command line (**-p** option) so that the specified test package is already open when the Test Manager starts.

Creating a Test Package

The main window of the Test Manager appears and displays a directory listing similar to the following illustration:

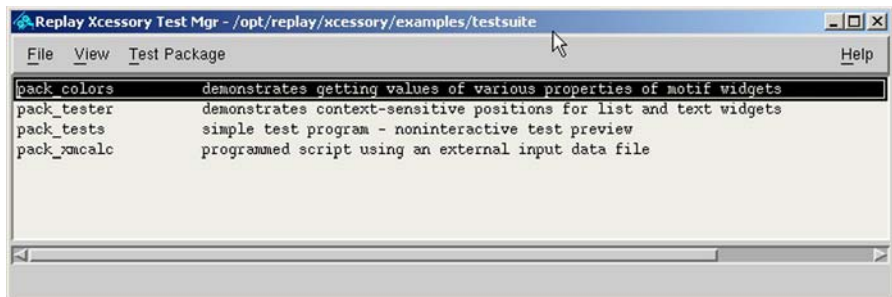


Figure 1 Test Manager Window: Suite Directory Listing

GETTING STARTED

Tutorials

Each line in the listing represents a test package, which is a directory that holds all the files and directories associated with a set of test cases. Two or more test packages grouped by some criteria make up a test suite.

Before running a record session, for example with the calculator, you need to create a test package to hold session files and directories. To create a test package for the tutorial sessions:

1. Select **New...** on the **Test Pkg** menu.

You will see a Test Package dialog box, as shown:

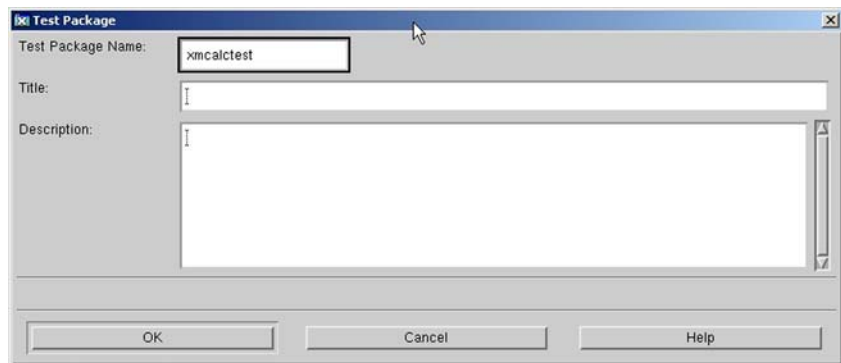


Figure 2 Test Package Dialog Box

The dialog box accepts information about the test package to be created.

2. For this tutorial enter the following—or something similar—in the dialog box:
 - In the **Test Package Name:** area, enter **xmcalctest** as the name of the directory for the new test package.
 - In the **Title:** area, enter a short annotation. The single-line description entered here appears as one of the lines in a directory listing.
 - In the **Description:** area, describe the test package in as much detail as necessary. Replay Xcessory places this information, along with the **Title:** information, in a testpackagename.tpd file in the new test package.
3. Press **OK** to complete test package creation.
A Test Package window appears, as shown:

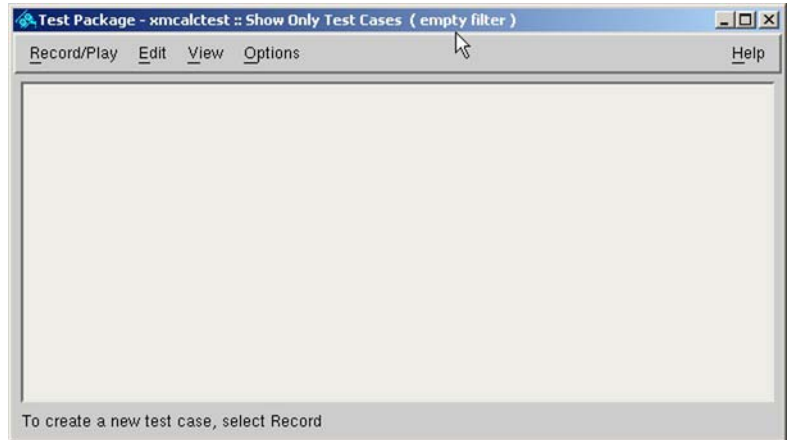


Figure 3 A New Test Package

The Test Package window is empty.

Deleting a Test Package

You can delete a whole test package if you do not need it.

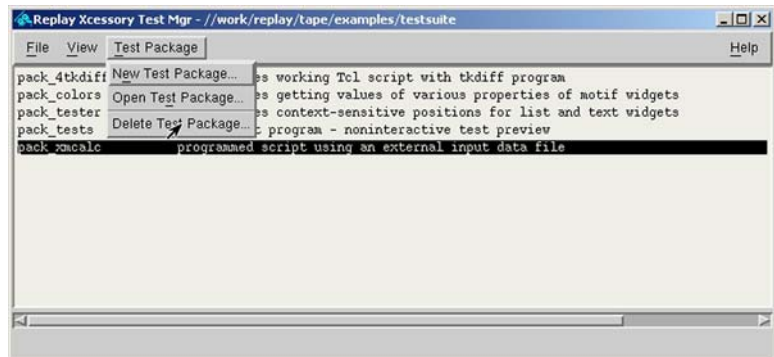


Figure 4 Deleting a Test Package

To do this, you should go to the “Test Package” menu and select “Delete Test Package Item.” Be sure that you properly confirm the deletion of the test package- you will not have the ability to restore it!

Conducting a Record Session

You are now ready to start a record session with the calculator.

1. Select **Record ...** from the **Record/Play** menu.

The Record Control Panel consists of three pages: Descriptions, Settings, and Record. The first page (Figure 5) contains general information about the current test case, such as test case Name, Title, and Description. **Test Name** is the only required field, and after you have filled it out, you can select the second tab (Figure 6) and fill out the needed settings.

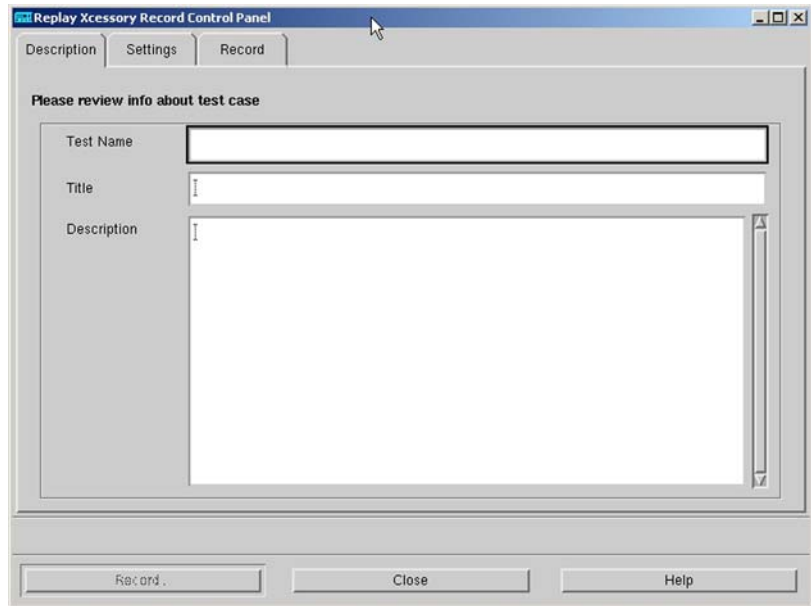


Figure 5 Record Control Panel, Description Tab

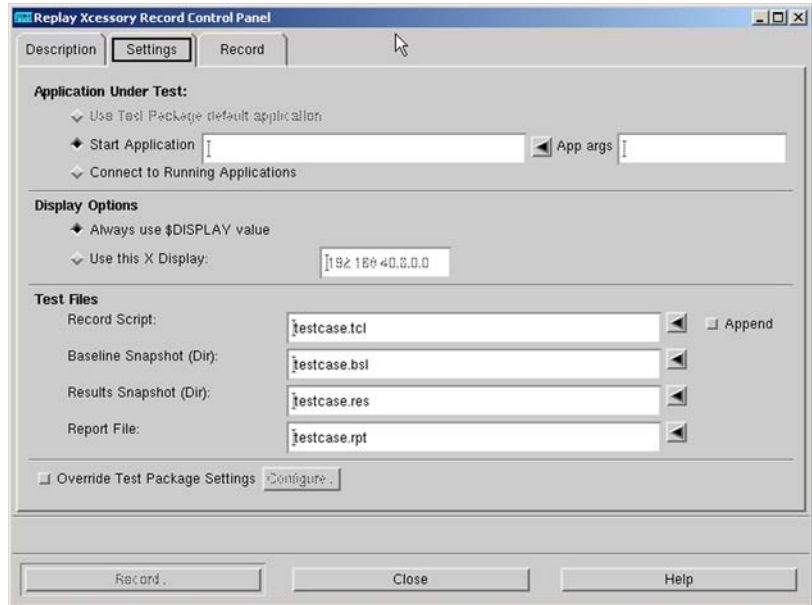


Figure 6 Record Control Panel, Settings Tab

1. For this tutorial, enter the following information in the panel:
 - In the **Application Under Test** area, place the cursor in the **Start Application** box and enter **xmcalc**, the name of the calculator test application.
 - The **Application Display** area should indicate the location of the display being used.

All fields about files and paths which Replay Xcessory will use will be filled automatically once the test case name is entered. The user can change any of these options as desired, or leave them as is.

2. Press **Record** to complete this part of the record session.

Several things will occur:

- The buttons along the bottom of the record control panel change to a new set of buttons that allow you to control the record session itself; for example, by marking the beginning of a subtest, taking a snapshot, pause, and so on.
- The main window of the calculator appears, as shown:

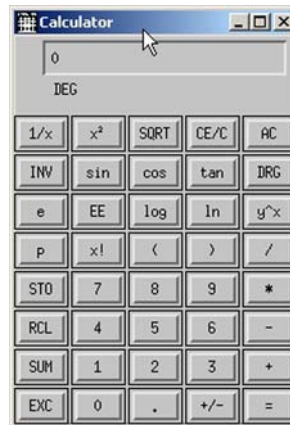


Figure 7 The Calculator

- The replay switches to the third page and disables Description and Settings pages.

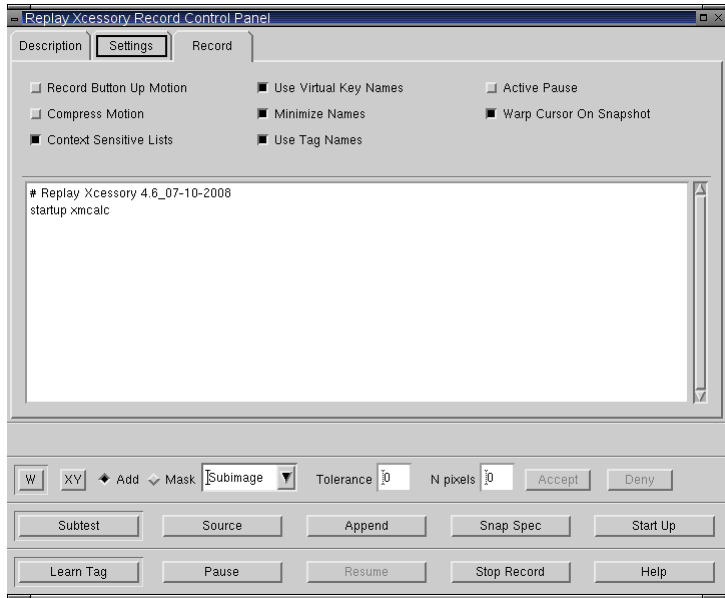


Figure 8 Record Control Panel, Record Tab

3. Enter the first subtest as follows:
 - Press the **Subtest** button to indicate the start of a new subtest. Enter the name of the subtest (1).
 - Press **AC** (clear) and then **4 * 5 * 6** and **=**.
 - Press the **Snapshot** button, at the bottom of the Record Control Panel, to save the results of the first test.
 - Since the snapshot scope defaults to **Object**, the cursor changes to resemble a camera.
 - Move the cursor over the LCD area of the calculator and click.
 - Replay Xcessory takes a snapshot of the LCD widget only.

GETTING STARTED

Tutorials

4. Enter the second subtest as follows:
 - Press **Subtest**.
 - Press **AC**, then **7**, **/**, and **2** keys.
 - Press **Snapshot** (or use **Ctrl+i**) to save the screen results of the second test.
5. To exit from the calculator, either choose **Quit** from the window manager menu, or click the right mouse button on **AC**.
6. To end the record session, press the **Stop** button at the bottom of the Record Control Panel.
7. To dismiss the Record Control Panel, press the **Cancel** button

In the Test Package window there are new icons that represent the files and directories created by the record session. By default, the filter for only showing the test case icons is activated.

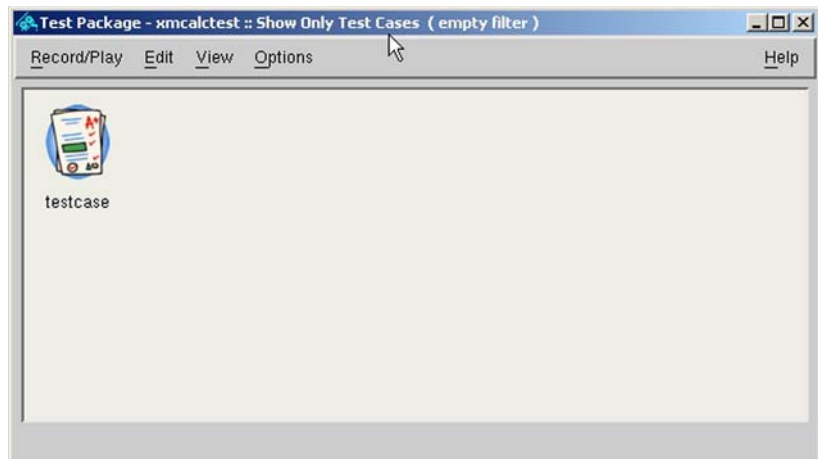


Figure 9 Test Package After Recording

As you can see, only the test case icon is now in the main replaytm window. You can change the current view mode by clicking the right mouse button on clear space and select "Show All"

Now you can see all icons generated by replay files and directories.

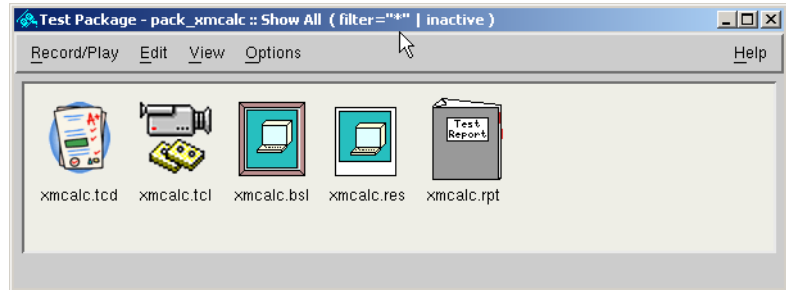


Figure 10 Test Package, “Show All”

Conducting a Play Session

In the record session, you created a script file, divided the script into subtests, and created baseline snapshots for subtest verification. You now have everything necessary for a play session.

To begin a play session:

1. In the test package window, select the **testname.tcd** icon and select **Play** from the **Record/Play** menu, or just double click on the icon.

A play control panel appears, as shown:

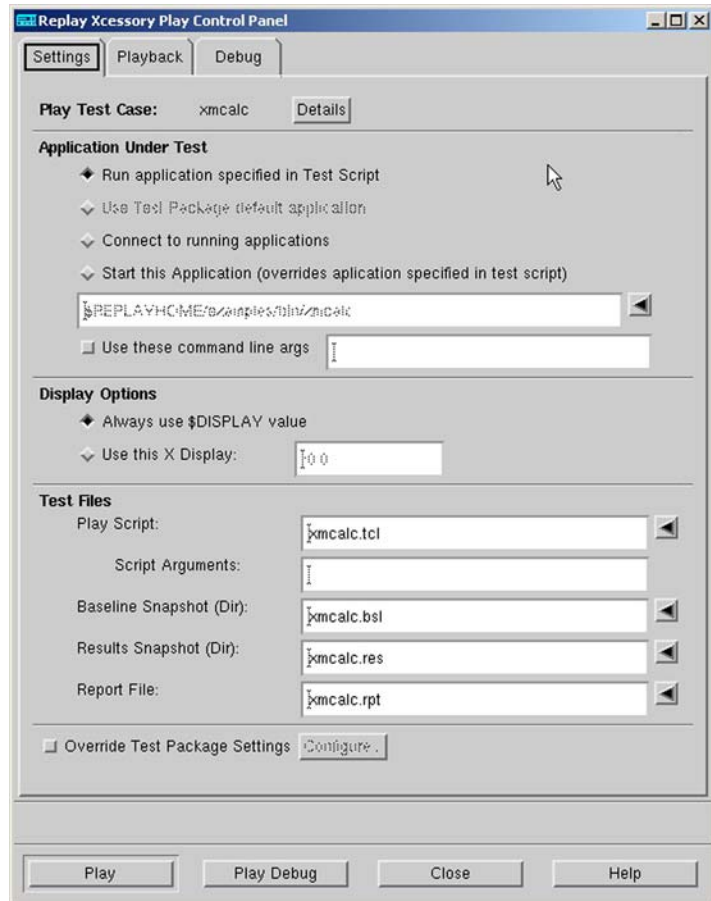


Figure 11 Play Control Panel

2. For this tutorial, enter the following in the panel:
 - Under **Application Under Test** leave the toggle “Run application specified in Test Script” on.
 - Leave the **Baseline Snapshot (Dir)**, **Results Snapshot (Dir)**, and **Report File** fields with their present values, or change them as desired. Note that if you change the Baseline Dir name, previously generated baseline snapshots will be inaccessible.
 - Select display, the graphical output to which Replay must send.

Press **Play** in the Play Control Panel. Replay will switch to the second page, making the first page disabled.

As the playback begins, the script scrolls through the script view area. The cursor moves over the calculator buttons being pressed.

The play speed will seem slow if there were pauses between entries. To rerun the play session at a faster rate, adjust the play speed scale on the play control panel. Drag the slider to the right towards 1.0 before pressing the **Play** button.

3. Exit from the Play Control Panel by selecting **Stop** and then **Cancel**.

Note: Be sure to exit the calculator by selecting **Quit** on the window manager menu; otherwise new calculators may appear stacked on top of each other.

Examining the Report File

Since you provided baseline snapshots and requested a report file, the report file should indicate—based on snapshot comparisons—whether the tests were successful. To look at the test results, double click on the report icon in the test package window.

A window containing the report file appears, and displays information similar to the following illustration:



Figure 12 Examining the Report File

In the current version of Replay Xcessory, the user can generate reports in HTML format. Such a report will be generated based on the xml report file for the test case/test package and placed in the test case/test package directory. If there is not an xml report file, an HTML report file will not be generated. To generate an HTML report, right click the desired test case or test package in the test suite tree and select the menu item “Full report...” Replay Xcessory will generate an HTML report for the selected test case or test package using the replayrep console utility. The HTML report will be automatically opened with user’s default browser. To change the browser where the report will be

displayed, click the File->Preferences menu item in the main Replay Xcessory window and fill in “HTML Browser” with the path to the correct browser application.

Examining the Script File

Double clicking on the script icon will open a script view panel:



Figure 13 Examining the Script File

The script is the same as the one you saw scrolling by during the record session, only the replay settings created at the record stage are added as comments at the beginning.

Examining the Baseline and Result Snapshots

Double clicking on the baseline and result icons opens view panes for these directories. Each pane initially shows a list of the files contained in the baseline or result directory. Each snapshot taken results in a separate ASCII file containing the logical snapshot.

The name of each snapshot file corresponds to an entry in the script file. These files have a **.snp** suffix. Double click on a snapshot entry to see the format of the snapshot file.

As you can see in the following illustration, the format of a logical snapshot is identical to that of an X resource file.

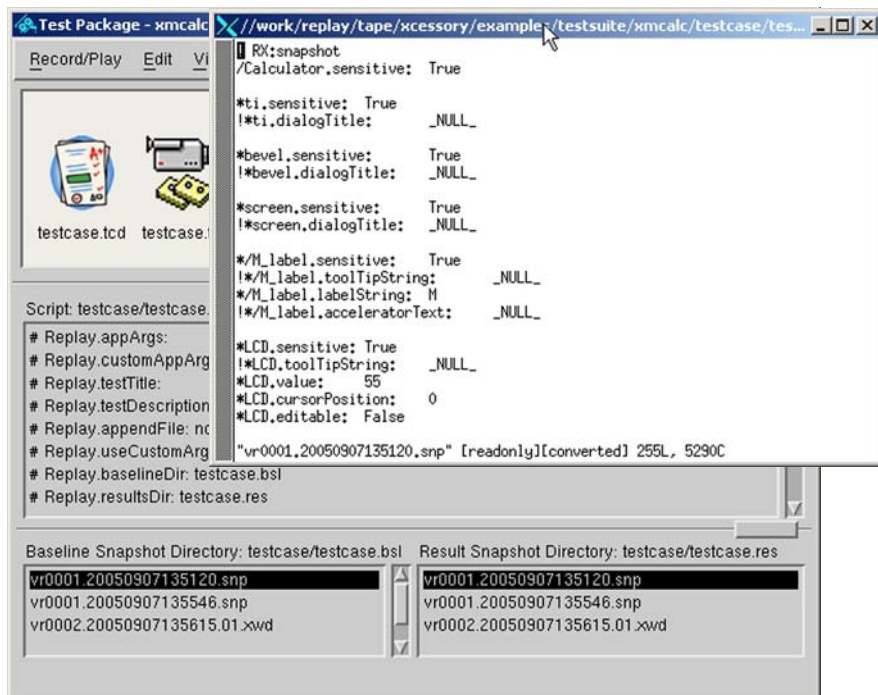


Figure 14 Examining the Baseline and Result Snapshots

If image snapshots had been taken, they would have also appeared here. An image viewer comes up when double clicking on an image snapshot; image snapshots can be recognized by their **.xwd** suffixes.

Error Handling

Replay Xcessory reports the name of the script file and line number for script syntax and run time errors. However, syntax errors are not recoverable; the user will have to correct the script and restart the play session.

Tutorial 2—Programmatic Testing

Test scripts can be manually written to take advantage of Tcl's programming constructs. Such scripts can also be generated before the application's processes are ready for testing. Recorded scripts can be edited to include control flow statements.

The following sections describe a calculator test driver in which the input data is read from a file. The script converts each input line into clicks on the calculator buttons and compares the result against the expected result, which is given as the last number on each input line, such as:

```
10 + 2 = 5
1 + 1 = 2
5 - 2 = 3
3 * 7 = 21
```

The input file makes it easy to separate the input data from the script itself. It is also a straightforward way to generate large numbers of portable subtests because the data is reusable across different scripts or applications under test. The script and input file simulate the actions of the person doing the testing.

Step-by-step directions are included in the next section, followed by a description of the test results. Following the results is a listing and an analysis of the script itself.

Running the Calculator Test Driver

Here are the steps to run the calculator test driver:

1. Double click on the **xmcalc** test package in the **replaytm** main window.

The **xmcalc** test package is displayed in the Test Package window.

2. Select the **xmcalc.tcd** icon; then select **Play...** from the **Record/Play** menu.

The play control panel appears, with Play Script set to **xmcalc.tcl**.

3. Snapshots will not be used for result verification. Instead, direct comparison of actual and expected results will be used.
4. Press **Play** at the bottom of the Play Control Panel to start the session.

Note: An erroneous result of 5 has been assigned to the calculation $10 + 2$, in order to make one subtest fail.

A subtest report summary similar to the following is generated and saved (unless the report file is **null**) in an **xmcalc.rpt** file:

```
Subtest 1 (10 + 2 = 5)  FAILED
Subtest 2 (1 + 1 = 2)   Passed
Subtest 3 (5 - 2 = 3)   Passed
Subtest 4 (3 * 7 = 21)  Passed
Subtest 5 (10 / 2 = 5)  Passed
Subtest 6 (1 * 9 = 9)   Passed
Subtest 7 (1 + 1 + 1 = 2) Passed
Subtest 8 (5 * ( 1 + 4 ) = 25) Passed
Subtest 9 (1.1 + 2.3 = 3.4) Passed
Subtest 10 (4.4 / 2 = 2.2) Passed
```

```
Percent Subtests Passed: 90.0% (9/10)
```

Calculator Test Script

Here is a listing of the script that has just run; it begins with some support procedures, followed by the main loop. A description of the script logic follows immediately after the listing.

```
# This Tcl script read an input file, breaks each
input line
# into the elements of an arithmetic equation. All
elements
# except the last one are entered into the
calculator. The last
# element of each line is the expected result.
This script
# only handles number input data.

proc enter_digit { digit } {
    if { $digit == "." } {
        click /\. Button1
    } else {
```

```
        click "/$digit" Button1
    }
}

proc enter_operator { operator } {
    if { $operator == "*" } {
        click {/\}*} Button1
    } else {
        click "/$operator" Button1
    }
}

proc clear_lcd {} {
    click {/AC} Button1
}

# Procedure to extract digits out of number string

proc enter_number { number } {
    set digits [ split $number {} ]
    foreach i $digits {
        enter_digit $i
    }
}

# Start program and open input file

startup xmcabc
```

```
activate {xmcalc}
currentwin {/xmcalc}

set File [open test.data r]
set testnum 0
gets $File buff

while { [ eof $File ] != 1 } {
    incr testnum 1
    subtest "$testnum ($buff)"

    # Separate the elements of the buffer into a list
    set token_list [split $buff { }]

    # Get the expected result (last element in list)

    set last_element_no [expr { [ llength
$token_list ] -1} ]
    set expected [lindex $token_list
$last_element_no ]
    incr last_element_no -1

    # Enter each element in list

    for {set i 0} { $i <= $last_element_no } { incr i
1 } {

        set current_element [lindex $token_list $i]
        if { [regexp {^[0-9.]+$} $current_element ] }
        then {
```

```

        enter_number $current_element
    } else {
        enter_operator $current_element
    }
}
# Compare value in calculator's LCD against
expected result

set result [getvalue {*LCD} {label} ]
if { $expected == $result } then {
    pass "$expected == $result"
} else {
    fail "$expected != $result"
}

# Clear display prior to next line of input

clear_lcd
gets $File buff
}

close $File

```

Description of the Calculator Script

The following is a description of the general flow of the script previously shown. The main loop of the script is preceded by actions that:

- clear the calculator
- convert a number into its component digits
- convert a digit in the input to a **click** command on a calculator button
- convert an operator to a **click** command on a calculator button

Each time through the main loop the following occurs:

- The subtest commands mark the beginning of a new subtest. The subtest name can be any string; in this case, it is the concatenation of the test sequence number and the test data.
- The loop begins with a new calculation in **\$buff**. The read action actually occurs at the end of the loop, using the **gets \$File buff** command to read a line from the **test.data** file. Items of the line become list elements.

The following line is typical:

```
5 * ( 1 + 4 ) = 25
```

The first items, separated by white space, are calculations for the calculator. The last item provides the expected answer for subtest verification.

- The **lindex** list function picks up the last element in the list—the expected answer for results verification. The script loops through the elements of the list. Each element is either a number—integer digits and, optionally, a decimal point—or an operator.
- If the element is a number, the **enter_number** procedure converts its digits to **click** commands on digit keys. If the element is an operator, the **enter_operator** command converts the operator to a **click** command on the operator key.
- The script uses **getvalue** to get the answer posted by the calculator.
- The script compares the calculator’s answer with the correct answer and, depending on the comparison, issues either a **pass** or a **fail** command.

Concepts of Operation

Overview

This chapter describes the concepts, elements, and terminology of Replay Xcessory. Although some of the same material was presented briefly in Chapter 1, we recommend that this chapter be used as the primary reference for Replay Xcessory concepts and terminology.

Introduction

Following the tutorial introduction of Chapter 1, this chapter proceeds systematically through the features, concepts, and terminology of Replay Xcessory. The discussion is divided into the following major topics:

- Replay Xcessory architecture
- script language
- results verification
- event synchronization
- code coverage

Replay Xcessory architecture

Figure 15 illustrates the elements of the Replay Xcessory architecture.

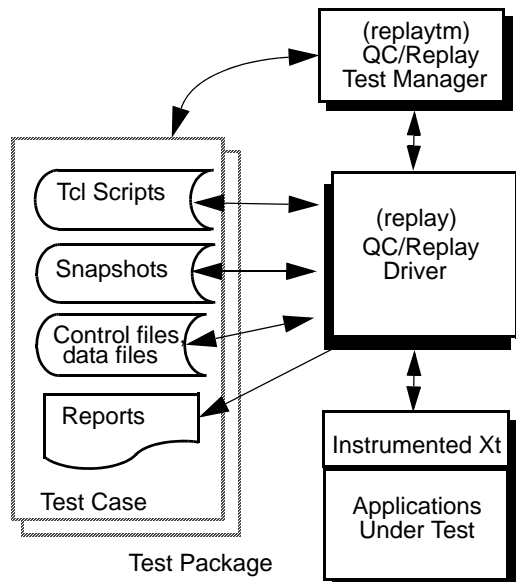


Figure 15 Replay Xcessory Architecture

As shown in Figure 15, the application processes under test communicate with the Replay Xcessory driver via hooks in an instrumented X Toolkit library provided by Replay Xcessory.

The Replay Xcessory driver controls all record and play sessions (start session, stop session, take snapshot, and so on) using a private protocol recognized by the instrumented X Toolkit library.

All files and directories created during the record or play session are saved to files in the current test case directory. These files and directories comprise the test case which is targeted to cover a single record session.

The Test Manager provides an optional GUI to the test package. The set of test packages constitute the test suite, which is targeted to cover one application's processes or subsystem.

Applications Under Test

Replay Xcessory operates by using an instrumented version of the X Toolkit library. The application under test can be linked with this library in two ways:

- If the application is dynamically linked with the X Toolkit library, then no additional rebuilding is necessary. The instrumented library is located by setting the library path environment variable directly or indirectly prior to starting the processes of the application under test.

Exceptions to this instrumentation include those dynamically-linked applications which ignore the standard library path environment variable, for example, **setuid** programs and programs that use `LD_RUN_PATH`.

- If the application is normally linked statically with the X Toolkit library, it must be relinked if it was compiled using the same header files as they use now (compiled against the same Xt version Library). The rebuilt application can still be used normally, independently from any capture or replay session. Otherwise the application must be recompiled too. If the application is sensitive to the speed of the Xt library, the user can specify any value for the `RX_NATIVE_XT` environment variable to disable processing of additional events.

Multi-Process Testing

Replay Xcessory supports regression testing of multiple X Toolkit application processes. There are three ways to select a new application for testing:

- Use the **Start Up** button to start a new application that is to be tested and recorded. The **Start Up** button prompts for the command line of the new application. A **startup** command is recorded and inserted within the script for each application selected with the **Start Up** button.
- Secondary processes can be started by the current application under test (using the **fork** or **exec** system calls).
- Use the **Connect** button to connect to all applications currently being executed that use the Replay Xcessory instrumented X Toolkit; a **connect** command will be recorded.

Regardless of how the processes are started, all Replay Xcessory actions, such as snapshots and resource merges, take place on the current active application; the current active application is displayed in the Replay Xcessory message line.

Replay Xcessory Driver

The Replay Xcessory driver provides the controls for the record or play session. It also records or interprets the scripting language and converts the commands into the low-level X events which must be received by the application's event queue.

The driver can be invoked directly by using the **replay** command. The driver must always be started in either the record or play mode. It can be invoked interactively or in batch. Batch mode is useful for running unattended sessions. The **replay** command is not explicitly invoked when running under the Test Manager. The driver can also be started with a Tcl debugger when in the play mode. The driver is described in detail in Chapter 5; the Tcl debugger is described in Chapter 9.

Replay Xcessory Test Manager

The Test Manager places all the features of Replay Xcessory under the control of a graphical user interface (GUI). See Chapters 3 and 4 for comprehensive information regarding the Test Manager's GUI. The test package and test suite concepts are basic to understanding the Test Manager.

The Test Manager:

- provides a directory listing that displays the test packages of a test suite and menus to change test suites and open test packages
- provides and organizes an iconic view of a test package as a collection of related test scripts, baseline and result snapshots, application data files, and reports
- starts record and play sessions and enables users to customize their interactions with Replay Xcessory

Use of the Test Manager is not required for operation of the core Replay Xcessory functionality. This enables Replay Xcessory to be easily integrated into existing test management facilities or shell scripts.

Test Suites

A *test suite* is a directory that contains a number of related test packages. The Test Manager displays the test packages of the current test suite and enables you to navigate from test suite to test suite.

Test Packages

A *test package* is a directory that holds and organizes the files and directories associated with a record or play session. We recommend that a test package be restricted to the files associated with a single record or play session.

Test Cases

A *test case* is a xml file that contains a custom *test case definition* - the set of the settings (which applied only to this test). Several test case definitions may have the same settings (for example, they might have the same script path or baseline directory) and differ in others.

Subtests

A *subtest* is a single script file in a test package that can contain more than one test. By using a **subtest** command, the script can be partitioned into multiple subtests. The results in a **Reports** file are summarized subtest by subtest, as shown in the following illustration:

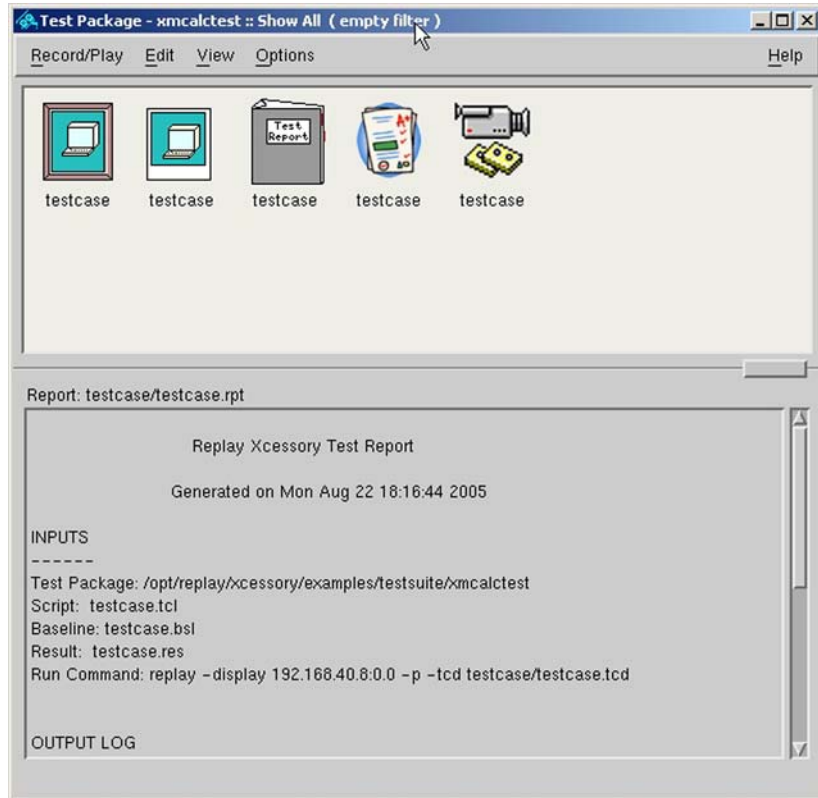


Figure 16 Test Package Report for Calculator Example

Replay Xcessory Script Language

The Replay Xcessory script language is based on the nonproprietary Tool Command Language (Tcl). Tcl was initially developed by John Ousterhout of the University of California at Berkeley and has quickly gained acceptance in the technical community, especially in testing organizations. Tcl is a powerful shell-like interpretive programming language that combines simplicity and flexibility.

A basic script consists of commands that specify actions on objects, plus additional command-specific information. The script is used to simulate input actions. In the case of a recorded script, the script enables a session to be played back. In the case of a handwritten script, the script represents what a user would do.

The following sections explain the various aspects of scripts and script commands:

- command actions and objects
- names of command objects (widget names)
- command delay
- programmability of scripts
- editability of recorded scripts
- portability of scripts

Script Command Actions and Objects

The command actions of the script are those actions that occur while using a particular application. Each user action is represented by a command. The following are the script commands, summarized by command category:

User Interaction	click dblclick multiclick
Commands	press release
	drag key text move resize
	iconify deiconify
	raise lower closewin popdown popup
	message
Test Management	subtest
Commands	pass fail
	snapshot
	mergespec loadspec

Widget	currentwin getcurrentwin
Information	getclass getchildren getpopups
Commands	ismanged ismapped iswidget getparent getfocuswidget widgetid widgetname widgettag getvalue setvalue getproperty windowid alias
Session	echo echoreport
Management	narrative activate
Commands	delay pause termsync startup connect system exec

Command objects represent the widgets with which the tester interacts, for example the menu items and push buttons.

Replay Xcessory also provides a set of extended commands that can be used for programmatic testing. The extended commands allow for more compact representation of interactions involving OSF/Motif. Additional information about extended commands can be found in Chapter 8.

Widget Names and Tags

Note: The **editres** program, available under X11R6 and later systems, can be used on X11-based applications operating under Replay Xcessory. This program provides a graphical view of the widget hierarchy and displays the widget names. The **editres** program is a useful tool for understanding the widget hierarchy in X11R6 (and later) applications. If **editres** is not available on your system, **ftp** the source for building it from **ftp.x.org** in the following directory: **/pub/R5**

Also useful is the program in OpenMotif demos, "getsubres", that shows a list of most of the general widget resources and their values.

Widget Names

In the script, a widget is identified by its *name* as assigned by the developer of the application's user interface. An easy way to determine widget names is to start a record session. As you click on a widget (for example, the **Help** button), its name scrolls through the script view area, and would look similar to the following:

```
click    {*help}Button1@26,15 2695
```

Widget Tags

An alternate way to identify a widget is to list the widget's *tag*. A widget tag is always preceded by a forward slash (/). A tag can be extracted automatically by Replay Xcessory from widgets that have obvious and logical names—for example, specific push button labels, or shell titles—where the tag is the button name or a window title.

An exception is made for label widgets. Their default is equal to the label string plus a **_label** suffix. With the **Record Tag Names** option toggled on, clicking on the **Help** button during a record session lists a widget's tag (in this case **{/Help}**), if the widget had a unique tag.

```
click {/Help}Button1@31,19 2080
```

A widget tag can be explicitly assigned using the **Learn Tag** facility (described in Chapter 5) which can be used to assign tags to widgets that do not have an easily identifiable label, for example, text fields or scroll bars. It is also possible to use the names of the parent widgets as part of the full widget tap name, if it is necessary: `.xmcalc.*./0` or `.xmclac.ti./9`. Using widgets in record mode is also described in the “Recording Widget Tags” section on page 142.

Fully-Qualified Versus Minimized Widget Names

The style of names that Replay Xcessory uses in scripts and snapshots, as well as many other properties, can be modified or defined. (Refer to Chapter 5 for detailed information on customizing record/play properties.) Two styles of widget names are present in scripts: fully-qualified and minimized.

The *fully-qualified name* is based on the widget hierarchy. A portion of the widget hierarchy for the calculator program (see the tutorial in Chapter 1) is shown in Figure 17 .

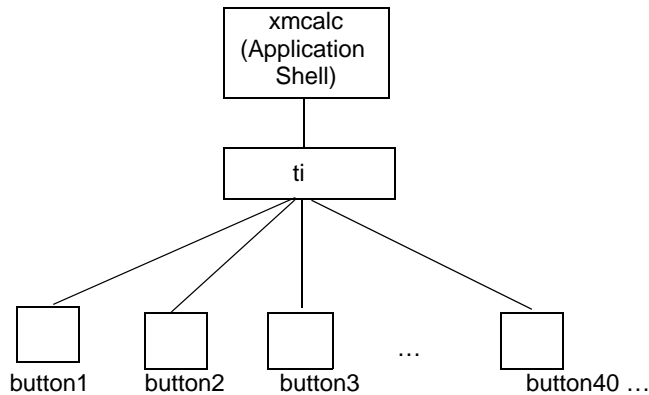


Figure 17 Widget Hierarchy

At the top of the hierarchy is **xmcalc**, the application shell: an ancestor of all the other widgets. The buttons are all children of **ti**. More complex applications can have multiple widget trees, each with its own application shell.

The fully-qualified name consists of the basic widget name qualified by each of its parents; for the first button: **xmcalc.ti.button1**.

The *minimized* name is the name of the widget prefixed by the minimum number of ancestor widgets needed to uniquely identify the widget. The asterisk (*) wildcard is used in the script to replace the omitted name components. The wildcard can only represent some number levels in the widget hierarchy, and cannot be a part of the widget name on a particular level. For example, **.xmcalc*button24** is equal to **xmcalc.button24**, **.xmcalc.level.button24**, **xmcalc.level.level2.button24**, and so on, but is not equal to **.xmcalc_widget.button24**. This limitation exists because of the X resources files standard that Replay Xcessory uses for tag representation. If the name of the widget is unique for the whole widget tree, no ancestor widget name will be prefixed to the widget name. The minimized name of the first button is ***button1**.

Recorded events may present a subwindow field. Each subwindow name is appended to the primary widget name. For details about widget subwindows, refer to the “Specifying the Target Widget” section on page 199.

Recording Name Components

The X Toolkit does not impose or enforce any restrictions on valid name components. Since Replay Xcessory relies on being able to uniquely identify a widget instance solely by its name, the following convention is used when recording widget names:

- Components that are duplicated at the same level are suffixed. The first instance is the component name. The second instance is suffixed by [2], the third by [3], and so forth. If a set of buttons had all been named “button,” the minimized names would have been ***button**, ***button[2]**, ***button[3]**, and so on.
The instance number always corresponds to the component’s initial position in the children list. Thus **instance [3]** will always be **instance [3]** even if **instance [2]** and **instance** are destroyed during the session.
- Components that are null are replaced by a single space.
- Certain characters with special significance to X, Tcl, or Replay Xcessory are escaped by the backslash character (\) if found in the widget name. These characters are: period (.), asterisk (*), dollar sign (\$), backslash (\), and left and right brackets ([]).
- If the application under test has multiple application shells, the application shell name is always recorded as part of the name, including minimized names.

Widget tags and names can be intermixed in the fully-qualified or minimized names. However, even with the **Record Tag Names** option toggled on, Replay Xcessory uses widget names if a widget tag cannot be found or is not unique enough to identify the widget instance.

Although Replay Xcessory can operate on any widget name, it is recommended that application developers give widgets unique, non-null, specific names in order to improve script readability.

Command Delay

The delay between most actions is recorded and used in play sessions to approximate the speed of the initial record session. The delay between the tcl commands can be forced to the default value (1 sec.) if the user sets the `Replay.omitDelay` resource to True in `REPLAYHOME/lib/appdefaults/Replayfile`. However, the play speed can be adjusted before starting a play session. The actual play speed depends on the desired play speed, system load, and whether widgets are in the proper state to accept the recorded action.

Note, however, that the tcl commands determine whether this option works and not. TCL commands that cannot take a variable amount of parameters often ignore the delay value when Replay Xcessory makes the script record as short as possible by considering default values.

For example, the full form of the click command is **click** *widget state [location [delay]]*, but only “*widget*” is obligatory when clicking on a widget other than `XmDrawingArea`, so using the default Replay Xcessory parameters will record the click command as `click “widget”`, which is equivalent to clicking “`widget`” `Button1 center_location 0`.

In such cases, to make the script more readable, Replay may omit the delay even if the `omitDelay` option is set to `False`.

On the other hand, the “`drag`” command always records a delay value because of especial handling of the command.

Each action is an atomic transaction. An action recorded as a double click will be played back as a double click regardless of the system or network load; two consecutive single clicks will not become a double click simply because the play back is done on a faster machine. (See Chapter 6 for more information.)

Command-Specific Information

In addition to specifying the action, widget, and time delay, most commands contain additional information. For example, a **text** command contains a string with the text entered, as well as the name of the widget where the text was entered. The **resize** command includes the new size of the window, as well as the widget name identifying the window.

Script Programmability

Tcl provides all the control flow constructs needed to support structured programming as well as many high-level built-in functions (string, array, and list processing, file input/output (I/O), regular expressions). Scripts can be written as modular units that contain user-defined procedures, and these scripts can be placed into reusable libraries. The file I/O functions add a powerful way to test, because input data can be read from external files rather than embedded in the script itself.

Script Editability

A recorded script file can be edited using any ASCII editor. The session can be programmed independently from the actual program if the specification for the user interface is known. Sensitivity to the actual widget tree can be minimized through the use of tags (see Chapter 5) and by using wild card notations in the name whenever possible.

Script Portability

The script file is portable across platforms and environments. For example, a session recorded on Red Hat Enterprise Linux 3.0 running the KDE window manager can be played back on a Sun SPARCStation™ running under an CDE window manager.

Results Verification

When regression tests run unattended, screen results must be programmatically verified. Comparing snapshot files is an important and straightforward programmatic method, as described in the following sections.

Snapshots

Snapshots provide a fast, convenient substitute for visual inspection by capturing the current screen when the snapshot is requested. During a record session, the Replay Xcessory tester selects snapshot as results are displayed on the screen. There are two types of snapshots: *widget* snapshots and *image* snapshots. (See Chapter 5 for more information about snapshots.)

Note: Subtests default to a **Pass** result if there are no snapshots taken and there are no explicit pass/fail commands invoked in the subtest. It is not possible to obtain a result of **Unknown**.

Widget Snapshots

A *widget snapshot* is an ASCII representation of the application's widget tree (or a portion of a tree) at a particular time. One can control which widgets, widget classes, and resource values for a widget class are included. This makes it possible to ensure that the string value of text fields are identical while ignoring differences in font, color, or other attributes. The resources are dumped in the standard X resource file format. The hierarchical relationships between the widgets is optionally preserved using pseudo resources. (See Chapter 5 for more information about pseudo resources.)

Widget snapshots are the recommended means of verifying results that occur as values (or other ASCII resources) within widgets. Comparisons of values that involve only ASCII comparisons are fast and reliable, and widget snapshot files are usually an order of magnitude smaller than image snapshots. Since values are referenced by the names of their widgets, such comparisons can be made insensitive to many changes in the application interface, such as the repositioning of a button or dialog box.

Image Snapshots

An *image snapshot* is a screen that has been captured as an **xwd** bitmap file. An application that displays its results as graphics, rather than as ASCII values, requires an image snapshot; for example, an application that draws or paints. Image snapshots can be optionally compressed to conserve disk storage.

Baselines

When the collection of snapshots from a record session have been verified as correct, they constitute a *baseline* snapshot. Baseline snapshots serve as a standard against which snapshots in future regression tests will be judged.

Replay Xcessory 4.6 supports a new feature - a set of valid baseline snapshots. If there is more than one correct snapshot for comparing tests and baselines or the snapshot content could be different for different time intervals, the user can replace the baseline snapshot file with a set of files. The user needs to create a directory named like a snapshot file and place all valid snapshots in that directory. Replay Xcessory will process the directory in special way. Each file will be compared with the target snapshots from the playback session. Directories with incorrect names will be ignored. Replay Xcessory will report mismatch errors only when all the snapshots are incorrect.

Users can create a set of valid baseline snapshots with the GUI version of Replay Xcessory in playback mode. Click the “Add” button on “mismatch error dialog” and Replay Xcessory will create the proper directory and place all available files in it. In case such a directory already exists, the files will be added to it automatically.

Regression Testing

As processes within an application evolve, they can be retested periodically using Replay Xcessory. The new version of the application is replayed using the original test actions, as recorded in the script. At each verification point during the play session, new snapshots are taken and compared with the baseline snapshots. If the snapshots are identical, the tests are successful. If not, the discrepancies can help determine the source of the problem.

Replay Xcessory compares new and baseline snapshots programmatically—depending on the type of snapshot—by using the **diff(1)** or **xwddiff(1)** utilities.

- The **diff(1)** utility compares widget snapshots.
- The **xwddiff(1)** utility compares image snapshots.

Replay Xcessory makes widget or image comparisons, or both, depending on what was requested during the test. The user can change the program for taking widget or image snapshots for the whole test package, by correcting the needed properties in the package settings dialog.

Determining Subtest Success

Replay Xcessory supports multiple subtests within a single test script. The success or failure of each subtest can be determined by any of the following mechanisms:

- Comparison of baseline and result snapshots provides the simplest method of determining success or failure. It is based on the assumption that there are *verification points* in a process execution where the screen contents can be checked to verify that they are correct. The snapshots can be widget snapshots, image snapshots, or both.
- Direct comparison of the widget contents with its expected value is more efficient than snapshot comparison. By using the **getvalue** and **setvalue** commands (which are analogous to **XtGetValues** and **XtSetValues**), it is possible to obtain the attribute of interest and compare the result directly against the expected value.

The **getvalue** command returns the requested widget resource; for example:

```
set result [ getvalue { *LCD } { label } ]  
sets the value of the label resource for the widget with the minimized  
name *LCD and places it in variable result.
```

Similarly,

```
setvalue {dialog_box_m} {background} "green"
```

sets the value resource to the color green of the widget named **dialog_box_m**.

- System commands and shell scripts can be invoked using the **system** or **exec** command. This allows the verification of states that are best verified externally.
- Custom resource type converters can be written to extract application data in a form that can be processed by the Tcl script (refer to “*Advanced Topics*” on page 261 for more information).
- Custom Tcl commands can be written to utilize existing libraries for result verification, for example, directly querying the database to verify that a database update took place.

Event Synchronization

With graphical user interfaces, events that occur out of the normal order can cause havoc. One example is a **click** command executed on a dialog box that has not yet appeared. Replay Xcessory addresses synchronization in two ways:

- automatic widget synchronization
- programmed synchronization

Automatic Widget Synchronization

Replay Xcessory uses widget status information to automatically:

- synchronize play events; the play speed can be adjusted to play at an accelerated rate to maximize usage of the available machine resources
- generate **popup** and **popdown** commands during a record session to force synchronization with pop up and pop down of shell widgets

Programmed Synchronization

Program synchronization points can be inserted into the script using any arbitrary state, described in terms of widget resource values. For example, it is straightforward to issue a **press** command and wait in a loop to issue the **release** command—that is, hold down a button until a slider reaches a certain value, as determined by the **getvalue** command.

For example, the following fragment shows a script that presses on a scale scrollbar until the scale value reads 70.

```
# press on slider in loop
while { [getvalue {tester.rc.scale} {value}] < 70
} {
press {tester.rc.scale.scale_scrollbar} Button1
    12 10
}
release {tester.rc.scale.scale_scrollbar}
    Btnlndown+Button1 12 10
```

In the preceding example, the **getvalue** command is used to retrieve the value resource of the **slider** widget. When the value becomes less than or equal to 70, the loop terminates.

Note: Replay Xcessory provides automatic synchronization for all keyboard and mouse input events. However, widget information commands, such as **getvalue** and **getchildren**, are executed immediately.

To ensure that Replay Xcessory maps a widget being queried by a widget information command, force a synchronization programmatically.

For instance,

```
while { [ismapped somewidget] != "TRUE" {
...}
```


Replay Xcessory Test Manager

Overview

The Test Manager places all the features of Replay Xcessory under the control of a graphical user interface. The material presented in this chapter assumes that you have already read Chapter 2, which introduces the concepts and terminology used here.

Introduction

Use of the Test Manager is not required for operation of the core Replay Xcessory functionality; however, this enables Replay Xcessory to be easily integrated into existing test management facilities or shell scripts. The Test Manager is designed to:

- partition a test suite into test packages
- provide a user interface for managing test suites
- organize and display a test package as a collection of test case definitions, related test scripts, baseline and result snapshots, application data files, and reports
- start file utilities, and record and play sessions

After starting the Test Manager, the directory listing for the current test suite is available. From this list you can open a test package that provides an iconic view of test package elements. Several test packages can be opened simultaneously. A variety of commands are available through menus of the test package window.

The following sections of this chapter describe:

- setting up the environment
- starting the Test Manager
- test suite commands
- test package commands
- test case commands

Setting up the Environment

In order to use Replay Xcessory you must make sure that your application uses a Replay Xcessory version of the necessary Xt library, instead of the regular version. You must also make sure that your path is correctly set to find the required Replay Xcessory executables and manpages.

Setting *REPLAYHOME*

You can set the *REPLAYHOME* environment variable in order to easily specify all other variables, or you can just use the path where you installed Replay Xcessory:

```
REPLAYHOME=/path_to_replay_root_dir
```

where *path_to* is the directory where Replay Xcessory is installed .

Consider placing the following appropriate environment specifications in your **.profile** or **.cshrc** file:

For **sh** or **ksh**, enter:

```
REPLAYHOME=/path_to_repalay_root_dir/  
export REPLAYHOME
```

For **csh**, enter:

```
setenv REPLAYHOME  
/path_to_replay_root_dir/Setting PATH
```

To set *PATH*, include the appropriate commands, below, in your **.profile** or **.cshrc** file:

Setting *MANPATH*

In order to access the on-line Replay Xcessory reference manual pages (manpages), include the following in your environment.

For **sh** or **ksh**:

```
MANPATH=$MANPATH:$REPLAYHOME/man; export MANPATH
```

For **csh**:

```
setenv MANPATH /$REPLAYHOME/man:$MANPATH
```

Using *Whatlib*

If you are unsure of whether you are linked with the instrumented Xt library, use the **whatlib**. For example, type the following:

```
whatlib $REPLAYHOME/examples/bin/tester
```

1. If the application is linked against the correct instrumented libXt, the user will see the message: Application linked properly. You can run Replay now.

2. If the application is linked against the wrong libXt, the user will see the message: Application linked statically against the wrong version libXt. You should relink it against the instrumented libXt and try again. If the application is linked with a shared libXt, this script will check the environment and report whether the Replay instrumented libXt is being used.

Multiple Library Paths

Dynamically-linked multiple processes usually have the same library path requirements for Replay Xcessory. However, one application may be based on Motif and another based on non-Motif widgets. If this situation occurs, potential problems can be avoided by statically linking one application with the appropriate Replay Xcessory instrumented Xt library and setting the library path to the appropriate library for the second application.

Setting Test Suite Properties

Test suite-wide properties can be placed in a single directory by setting the `REPLAY_TESTSUITE_PROPS` environment variable to the name of the directory that will be used to hold the test suite properties. The procedures pertaining to test suite-wide properties files, relative to similar files specified in local or home directories, or system defaults are described in detail in “*Record and Play Sessions*” on page 97.

Setting the Automatic Evaluation of Environment Variables

This is the procedure for automatic evaluation environment variables on the startup of the Replay Xcessory Driver and Replay Xcessory Test Manager. The main idea is that the user will not change the directory structure after the Replay Xcessory install.

1. When binary starts, it is trying to get the full path to itself.
2. It trims from the path binary file name and “/bin”.
3. Put to the environment `REPLAYHOME` variable - it is the place where Replay was installed.
4. Add “/lib/Xt” and export `LD_LIBRARY_PATH` with the additional value of the already existing variable.
5. Add “/xcessory/examples/bin/” and “/bin” and the export `PATH` with the additional value of the already existing variable.

Starting the Test Manager

To start the Replay Xcessory Test Manager, enter the following command:

```
replaytm &
```

The main window of the Test Manager appears with a test suite directory listing, as shown:

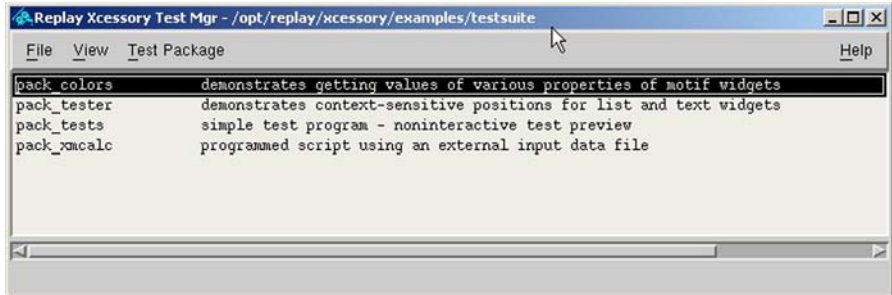


Figure 18 Test Suite Window

The directory listing displays the test packages in the current test suite. Each directory entry lists the name and a one-line description of a test package.

Test suites can be nested. If the line in the main replaytm window represents a test suite, it will be bold.

The user can move between nested test suites by clicking on the bold line and pressing the Backspace keyboard key (for entering one level higher).

Test Suite Commands

The following paragraphs describe how to use menus of the test suite window to:

- change the test suite directory
- create a test package
- open a test package
- customize utility commands

Changing Test Suite Directory

To display a different test suite, select **Change...** on the **Test Suite** menu. A dialog box appears, as shown:

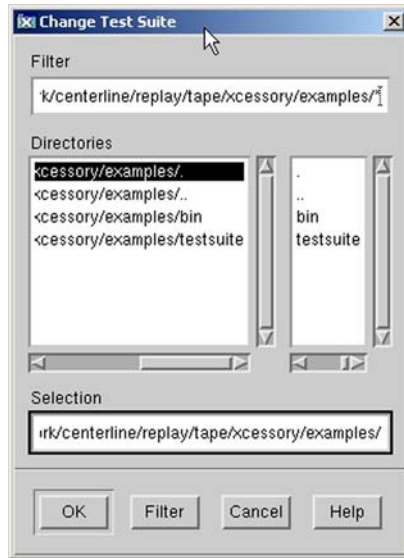


Figure 19 Change Test Suite Directory

To display a new, unrelated set of directories, enter the test suite directory name in the **Selection** field and click on **OK**; or, when you see the test suite directory of your choice in the **Directories** list box, click on it and then click on **OK**. Double clicking on a list-box entry also selects a directory. When you have finished, the Test Manager main window displays the test suite of your choice.

Creating a Test Package

To create a new test package, select **New...** on the **Test Package** menu. A Test Package dialog box appears, as shown:

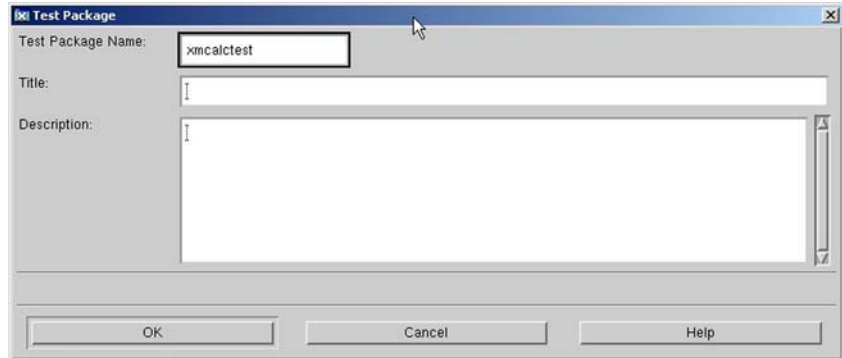


Figure 20 Test Package Dialog Box

Enter the following information in the dialog box:

- In the **Test Package Name:** area, enter a directory name for the new test package.
- In the **Title:** area, write a one-line description of the test package. The contents of **Title:** are displayed beside the test package name in a directory listing.
- In the **Description:** area, type in as much information as you like about the test package. This description, along with the title information, available through the View menu in the test package window.

When you press **OK**, the new package is created and opened. Refer to the next section for a description of an open test package.

The files and directories of a test package are created at various times:

- The physical name of the file is *testpackagename.tpd*, and it is created when the test package is created.
- The script file and the baseline directory are created at record time.
- The report file and the results directory are created at play time.

Opening a Test Package

To open a test package:

1. Find the test package in the directory listing of the appropriate test suite.
2. Click on it.
3. Select **Open** on the **Test Package** menu.

You can also open the test package by double clicking on an entry in the directory listing. When a test package opens, you see a test package window similar to the following illustration:

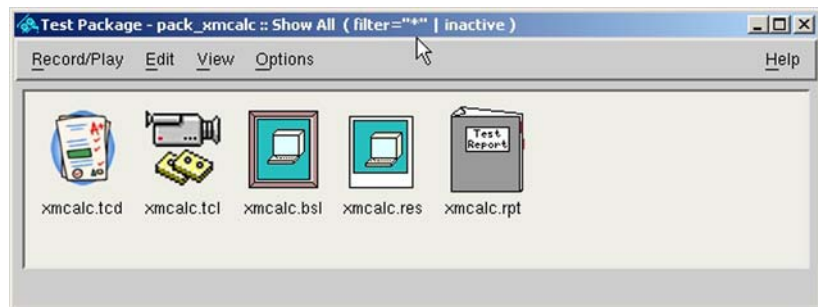


Figure 21 Open Test Package and its Icons

Here is a description of the icons shown in this Test Package:



Test case definition file



The script file is represented by a cartridge tape icon.

Suffix: **.tcl**



The baseline snapshots directory is represented by a framed picture icon.

Suffix: **.bsl**



The results snapshots directory is represented by an instant picture icon.

Suffix: **.res**



The report file is represented by a report binder icon.

Suffix: **.rpt**



Any nondirectory file that does not have a recognized suffix is given the file icon (a sheet of paper).



Any directory that does not have a recognized suffix is given the directory icon (a file folder).



File that contains the definition of widget tags of a custom application

Customizing Utility Commands

The **Edit**, **View**, and **Print** commands that are used in the test package window can be customized. To change one or more of these commands, select **Utilities...** on the **Properties** menu. A Utility Preferences dialog box appears, as shown:

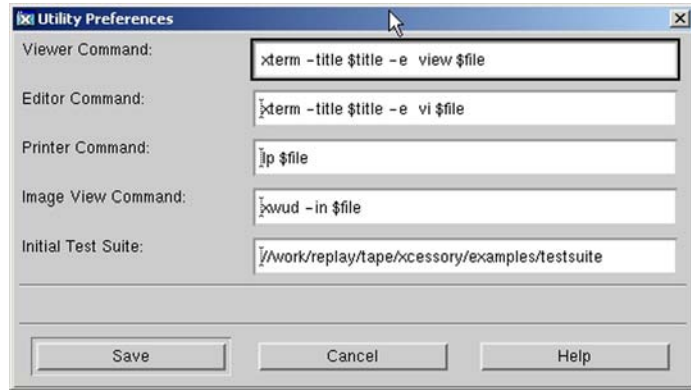


Figure 22 Utility Preferences

Press **Save** to use the new commands in the current session, and to save the values in the **.Replaytm** file for subsequent sessions.

Here are the commands that you can change and their default settings:

- **Edit** command. The default command to bring up an edit window with **vi** is:

```
xterm -title $title -e vi $file
```

- **View** command for ASCII files. The default command to bring up a window using the **view** utility is:

```
xterm -title $title -e view $file
```

Note: Avoid using viewers that exit automatically when they reach the end of the file, such as versions of **more** on some platforms.

- **View** command for image (**xwd**) files. The default command for viewing an image file is:

```
xwud -in $file
```

- **Print** command. The default command for printing is:
`lp $file`
- **Initial test suite:** indicates the directory to be used as the test suite when the test manager is initiated. This eliminates having to **cd** to the test suite directory before starting the test manager. If this field is empty, the Replay Xcessory Test Manager will save the current testsuite at the end of the session and restore it at the beginning of the next one.

Special tokens **\$title** and **\$file** will be replaced by Replay Xcessory prior to the actual command invocation—**\$title**, by a descriptive title; **\$file**, by the name of the file.

The editor and ASCII viewer commands should run in the X environment. If the command is not X-based, as with **vi**, you must initiate the command within an X terminal emulator, as done in the default commands previously shown.

Test Package Commands

The following paragraphs describe the commands available from the Package window:

- viewing test package elements—shows how to view various elements of the test package simultaneously
- using menu bar commands—explains the menu commands available from the menu bar
- using additional menu commands—explains the menu commands available from additional panes of the test package window

Viewing Test Package Elements

The following illustration shows a Test Package window, with views opened of the script file and the snapshot directories.

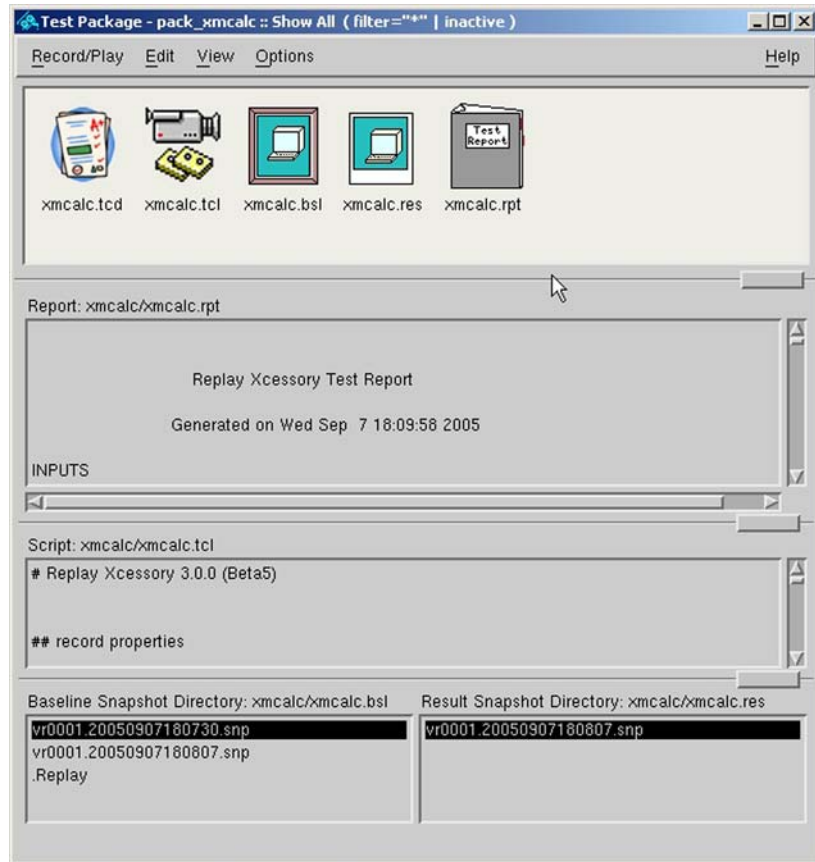


Figure 23 Test Package Window for **xmcalc**

Test Package Menu Bar

The following sections describe commands invoked through the menus in the menu bar of the Test Package window. These menus govern only the pane with test package icons. See “*Test Package View Panes*” on page 61 for information about menus for the other panes.

Note: Some menu options require that some test package icons be selected (or deselected). To select an icon and deselect all others, click on the icon with the left mouse button. To add other icons to the currently selected set, use **Shift+Button1**.

Record/Play Menu

The following commands are available from the **Record/Play** menu in the Test Package Window menu bar:

- | | |
|------------------|---------------------------------|
| Record... | Starts a record session. |
| Play... | Starts a play session. |
| Close | Closes the Test Package window. |

For detailed information regarding the **Record** and **Play** commands, refer to Chapter 5. For information about running with the Tcl debugger, see Chapter 9.

Edit Menu

The following commands are available on the **Edit** menu of the Test Package window.

- | | |
|--------------------|---|
| Edit Object | When the script or a file icon is selected, Edit Object creates an independent edit window. Use this window to edit the file and save the changes. |
| Link/Copy | This items allows the user to copy, move, or create a link on the file system to the appropriate test case into a test package other than the current one. This is active only when the test case icon is selected. |
| Print | When you select any directory element representing an ASCII file, Print will print the file. |
| Delete | Deletes the files represented by selected icons. For example, if you select the baseline directory icon and then select Delete , the baseline directory and all its included files are deleted. |

View Menu

The following choices are available from the **View** menu in the Test Package window menu bar:

- | | |
|-------------------------------|--|
| View Object | Creates a view-only display of a file or a snapshot directory. If the object selected is a Tcl script, a Replay Xcessory report, or a snapshot directory, the view is opened as a new pane within the test package window. Otherwise, an independent window is brought up. |
| Package Description... | Displays the description of the test package in a dialog box like the one used to create the test package. |
| Rescan | Causes the Test Manager to redisplay the test package. This is useful when new icons have been created since the test package was opened. |
| Show only Test Cases | Filters main icon area to show only test case definitions. |
| Show All | Shows all files related to the test cases. |

For detailed information regarding the **Session Props** and **Snapshot Props** commands, refer to Chapter 4.

Options Menu

- | | |
|--------------------------|---|
| Session Props... | Brings up the Session Properties form which allows you to change various parameters for the record or play session. |
| Snapshot Props... | Creates a text editor window for the snapshot dump control file (.Vrdump) for the current test package. If the file does not exist, it is copied from \$HOME/.Vrdump or from /usr/lib/X11/app-defaults . |
| Batch Creation... | Allows the user to choose which test cases inside the current test package to save into the xml batch file, which can be used later for unattended sessions. |



Figure 24 Batch File Creation Dialog

Test Package View Panes

The following paragraphs describe the panes of the Test Package window and what actions can be taken from each of the following panes:

- report view pane
- script view pane
- baseline and result snapshot view pane

Aside from the menus in the menu bar of the Test Package window, menus are available through the menu button of the mouse. These menus are available only when you press the right mouse button in one of the additional panes.

See “*Customizing Utility Commands*” on page 55 for information on customizing the operation of these menu commands.

Script View Pane

The Script View pane enables you to scroll through the script. You can also view each pair of snapshot files (baseline and results) in side-by-side views:

1. Open the script view pane and both snapshots view panes.
2. Select the three icons and then the **View** command or double click on each icon.
3. In the Script View pane, scroll until a snapshot command—shown in italics—is visible.
4. Double click on the **snapshot** command in the script view window.

The baseline and results files that correspond to this **snapshot** command are displayed in the baseline and results snapshot view panes, respectively. This feature enables you to proceed through all the **snapshot** commands in the script and directly view the corresponding snapshot files, as shown:

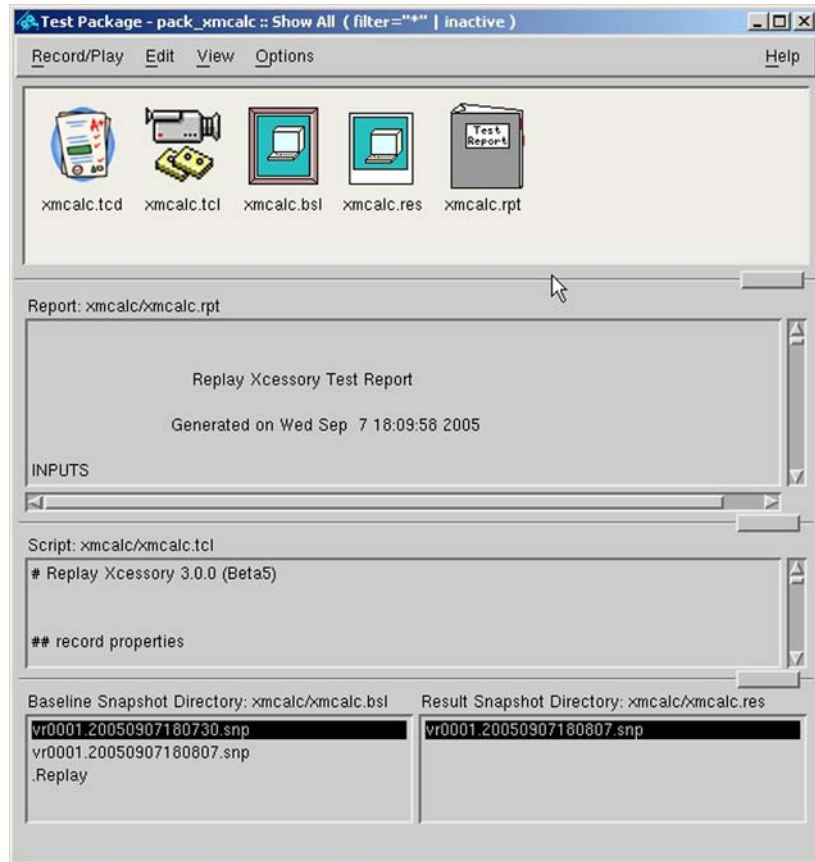


Figure 25 Script and Snapshot View Panes

Baseline and Result Snapshot View Panes

The *baseline snapshot view pane* lists the widget and image snapshot files in the baseline snapshots directory. Alternatively, double clicking on one of these file entries displays the contents of the widget or image file in the view pane.

Similarly, the *result snapshot view pane* lists the widget and image snapshot files in the baseline snapshots directory. Alternatively, if you double click on one of these file entries, the view pane displays the contents of the widget or image file.

Double clicking on an ASCII file in the baseline or results snapshot view panes is equivalent to selecting the **view** option from the popup menus. Double clicking on an image file (**.xwd** extension) causes the image viewer to be called.

The size of the view panes for result and baseline snapshots varies. If you view only one directory, the view pane occupies the full width of the test package window; if you view both directories, the view panes co-exist side by side.

The following commands are available through the right button of the mouse and apply to the highlighted directory element:

- view** For a widget snapshot, an image snapshot, or any ASCII file, the **view** command brings up an independent viewing window; double clicking on a directory element also brings up the viewing window.
- edit** For any ASCII file, **edit** brings up an independent edit window (with the ASCII editor of your choice).
- print** For widget snapshot files, or other ASCII files, the file is printed, using the **print** commands of your choice.
- close** Closes this view pane.

Report View Pane

The report view pane enables you to scroll through any report file. This option is especially useful when reviewing the output from batch runs.

Here is an illustration of a Report View pane:

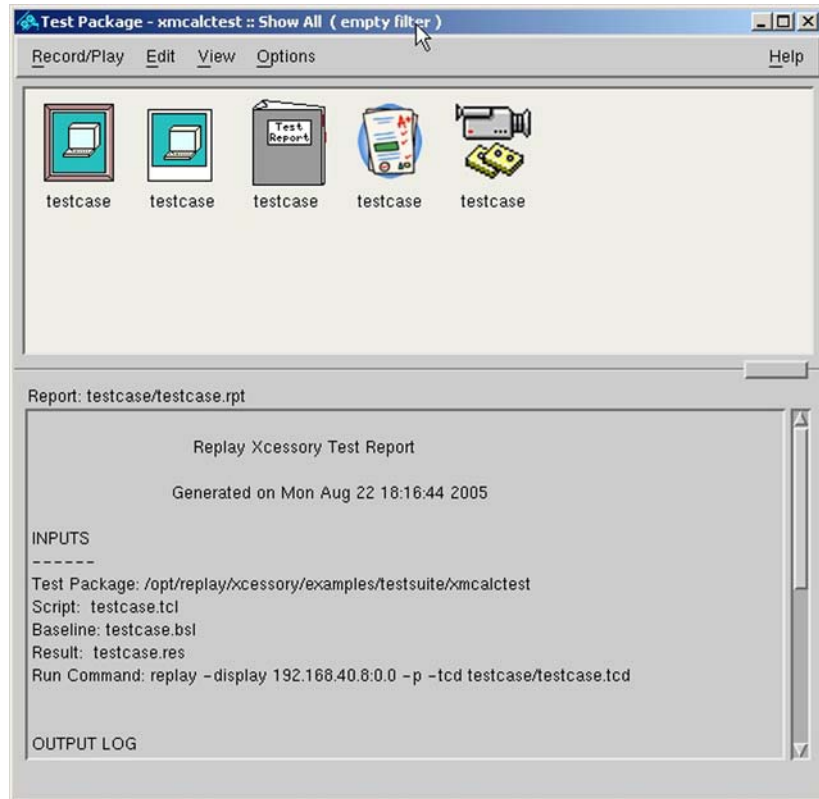


Figure 26 Report View Pane

Note that:

- Differences that have been found in the widget snapshots are included in the report.
- Differences that have been found in the image snapshots are referenced in the report. Double clicking on this line displays a composite (**XOR**) image of the differences.

The **close** command, on the menu of the right mouse button, closes this view pane.

Test Case Commands

The key icon in the main replaytm's work area is the test case definition icon.

It represents the custom test case defined by the user, and can be invoked to start record or playback session by selecting the icon and the appropriate item in the popup menu (which shows in result of a right click) or choosing Record or Playback item in Record/Play menu.



Figure 27 Test Case Popup Menu

The playback session also can be running by double-clicking on the test case icon.

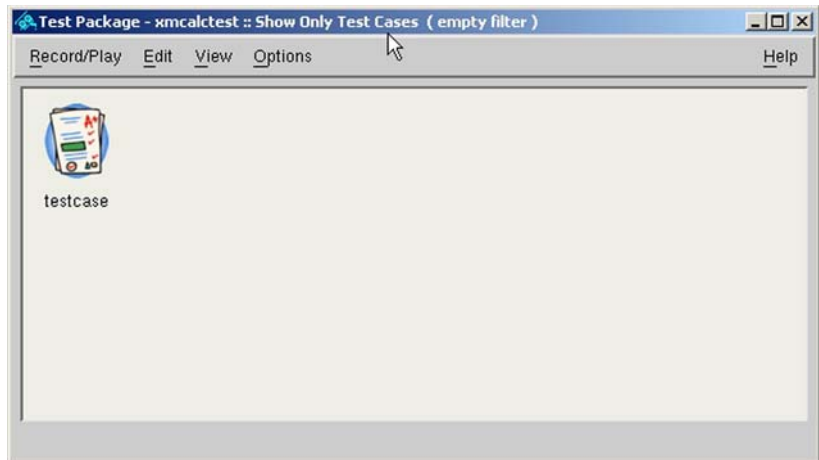


Figure 28 Test Case Icon

You are able to choose which view you wants to see at the current moment.

Three alternates are available:

1. Clicking on the tcd icon brings up a popup menu which has as a list item button “Show by test case” - by selecting this item, Replay will fill the test package view only with files and directories which are related to the tcd file on which right click was occurred.
2. Clicking on a empty space will bring up a popup menu which has an option to “Show only tcd files” - this will fill the package view only with icons which represent a particular test case - this can be useful if the package has a lot of tests.
3. Another option available is to “Show All”. It will fill the view with all the files which are related to all test cases, under the condition that Replay test Manager knows what that file is.

Each Test Case has its own line and sequence of showed icons. The first icon is always test case definition, the second is tcl script, the third is the baselines dir, the fourth is the snapshots dir, and the fifth is the report file.

The New Replay Xcessory Test Manager

4

Overview

This chapter discusses the new Replay Xcessory Test Manager. If you are using an older version of Replay, you should refer to Chapter 3.

Introduction

The following sections of this chapter describe:

- starting the new Replay Xcessory test manager
- preferences
- Replay Xcessory tag manager
- Vrdump editor window

Starting the New Replay Xcessory Test Manager (rtm)

To start the Replay Xcessory Test Manager, enter the following command:

```
rtm
```

The main window of the new Test Manager appears as shown:

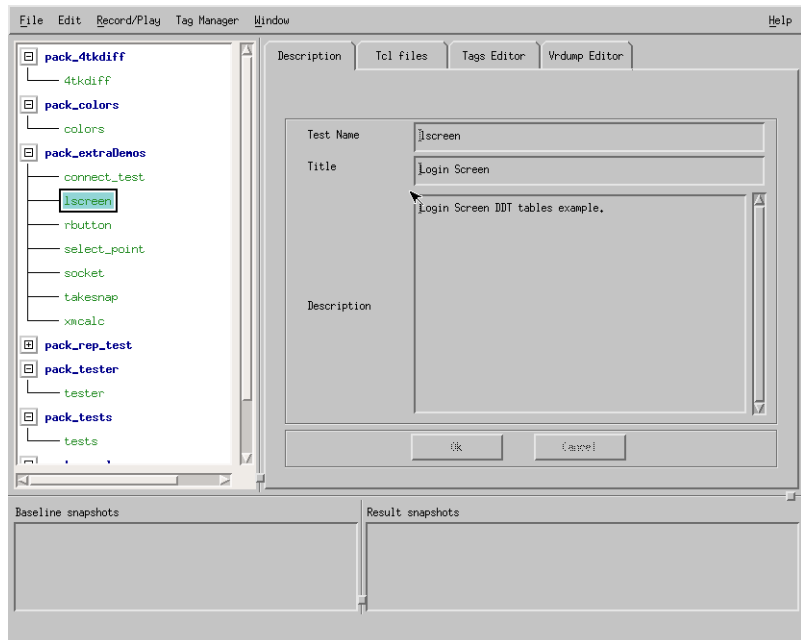


Figure 29 The New Test Manager

The test suite tree on the left hand side shows all the test packages and their test cases of the currently opened test suite in the tree hierarchy. Each first level node of the tree represents a test package and has an appropriate directory on the disk that contains the tpd file describing the test package. Each second level node of the tree represents the test case and has an appropriate directory on the disk that contains tcd file describing the test case.

The following sections describe how to use rtm main window menus to:

- manage test suites
- create a test package
- customize utility commands
- run playback/record sessions
- edit vrdump file
- work with tag files

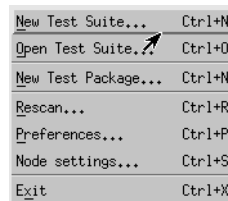


Figure 30 Test Manager File Menu

Creating the Test Suite Directory

To create another test suite, select “Create Test Suite” from the File menu. A dialog box appears, as shown:

THE NEW REPLAY XCESSORY TEST MANAGER

Starting the New Replay Xcessory Test Manager (rtm)

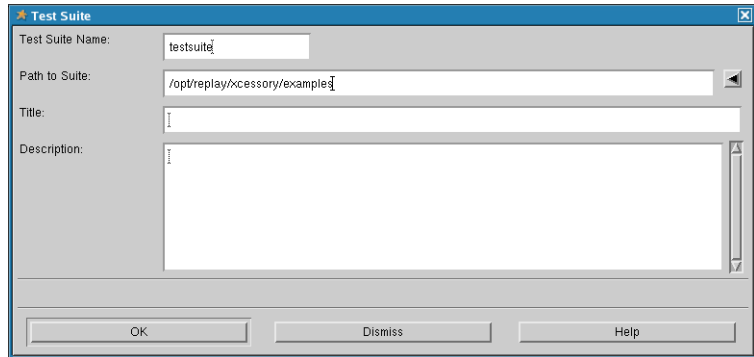


Figure 31 New Test Suite Dialog Box

Enter the test suite name and the desired directory in the path field and click on **OK**; When you have finished, a new test suite will be created and the hierarchy tree will be changed according to your input.

Changing the Test Suite Directory

To display a different test suite, select “Open Test Suite” from the File menu. A dialog box appears, as shown:

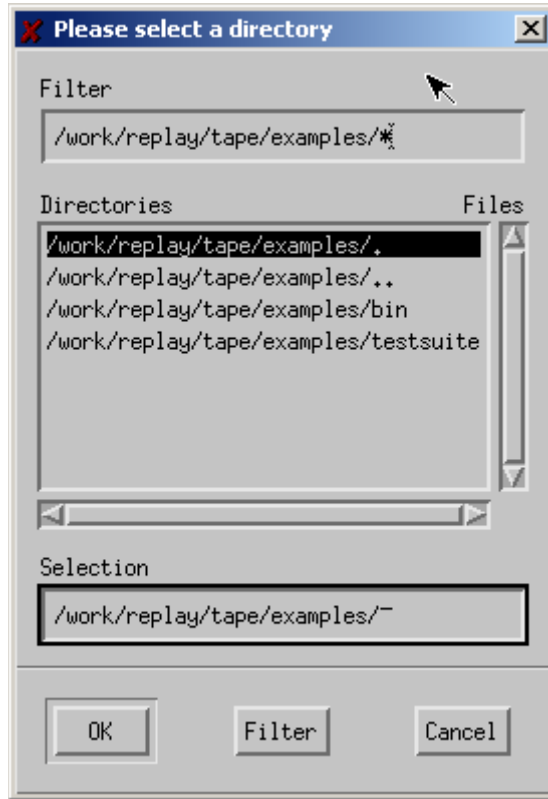


Figure 32 Open Test Suite Dialog Box

To display a new, unrelated set of directories, enter the test suite directory name in the **Selection** field and click on **OK**; or, when you see the test suite directory of your choice in the **Directories** list box, click on it and then click on **OK**. Double clicking on a list-box entry also selects a directory. When you have finished, the hierarchy tree will be changed according to your selection.

Creating a Test Package

To create a new test package, select **New Test Package...** from the **File** menu. A Test Package dialog box appears, as shown:

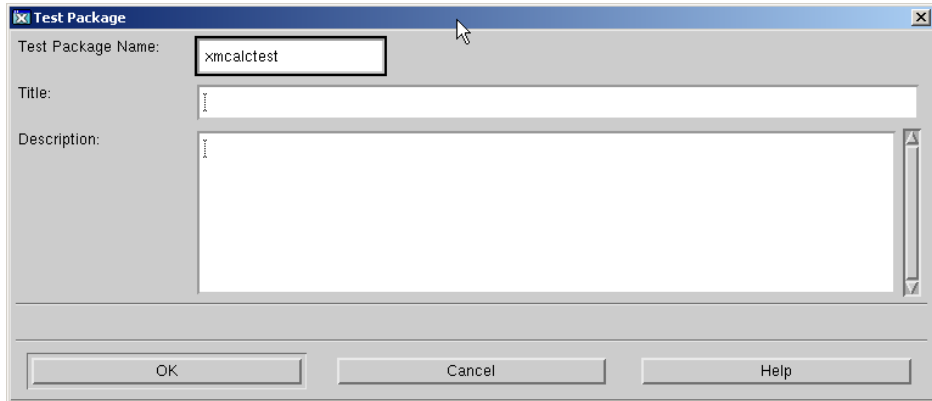


Figure 33 New Test Package Dialog Box

Enter the following information in the dialog box:

- In the **Test Package Name:** area, enter a directory name for the new test package.
- In the **Title:** area, write a one-line description of the test package. The contents of **Title:** are displayed beside the test package name in a directory listing.
- In the **Description:** area, type in as much information as you like about the test package. This description, along with the title information, is available through the View menu in the test package window.

When you press **OK**, the new package is created and opened. Refer to the next section for a description of an open test package.

The files and directories of a test package are created at various times:

- The physical name of the file is *testpackagename.tpd*, and it is created when the test package is created.
- The script file and the baseline directory are created at record time.
- The report file and the results directory are created at play time.

To update the hierarchy tree explicitly, the user can select “Rescan” from the File menu to force an update.

Preferences

The **Edit**, **View**, and **Print** commands in the rtm window can be customized. To change one or more of these commands, select **Preferences...** from the File menu. A Utility Preferences dialog box appears, as shown:

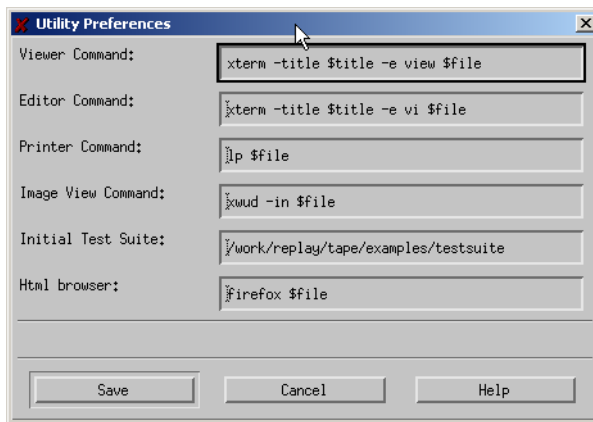


Figure 34 Utility Preferences Dialog Box

Press **Save** to use the new commands in the current session, and to save the values in the **.Replaytm** file for subsequent sessions.

Below are the commands that you can change and their default settings:

edit The default command to bring up the edit window with **vi** is:

```
xterm -title $title -e vi $file
```

view for ASCII files	<p>The default command to bring up a window using the view utility is:</p> <pre>xterm -title \$title -e view \$file</pre> <p>Note: Avoid using viewers that exit automatically when they reach the end of the file, such as versions of more on some platforms.</p>
view for image (xwd) files	<p>The default command for viewing an image file is:</p> <pre>xwud -in \$file</pre>
Print	<p>The default command for printing is:</p> <pre>lp \$file</pre>
Initial test suite	<p>Indicates the directory to be used as the test suite when the test manager is initiated. This eliminates having to cd to the test suite directory before starting the test manager. If this field is empty, the Replay</p> <p>Xcessory Test Manager will save the current test suite at the end of the session and restore it at the beginning of the next one.</p>
Html browser command	<p>When selected to generate xml reports rather than plain txt this command will be used to open html representation of xml report.</p>

Special tokens **\$title** and **\$file** will be replaced by Replay Xcessory prior to the actual command invocation—**\$title**, by a descriptive title, and **\$file**, by the name of the file.

The editor and ASCII viewer commands should run in the X environment. If the command is not X-based, as with **vi**, the user must initiate the command within an X terminal emulator, as done in the default commands previously shown.

Node settings	Please refer to the “ <i>Controlling Session Properties</i> ” on page 103.
Exit	Close the rtm window to stop working with the test suite.

Edit Menu

The edit menu contains two items that allow printing and deletion of the test suite tree items.



Figure 35 Edit Menu

If a **Print** menu item is selected, Replay Xcessory's test manager will try to print the definition file for the selected node item (tpd file for a test package and a tcd file for test cases). The print command can be adjusted in the preferences menu.

If a **Delete** menu item is selected, then after the confirmation dialog the Replay test manager will try to delete the entire test package/test case that was selected. Please note that it will be deleted using general system utilities, e.g. as though the `rm -r` directory was invoked. It can potentially be difficult to restore any deleted data.

Record/Play Menu

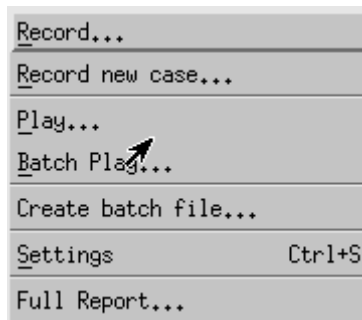


Figure 36 Record/Play Menu

The **Record/Play** menu allows the user to run the actual recording or playback sessions. The behavior of the menu items depends on what the current tree item is when a menu item is selected. The same menu is accessible as a popup (context) menu by right clicking on the tree node.

- Full report** If the generating xml reports option was selected, this option will bring up the html browser specified in the Preferences dialog to show an html representation of the generated xml report.
- Record...** Runs recording session. When selected on the test package node, it runs a recording of the new test case. When selected on the test case node, it runs a record session using the settings stored in the appropriate tcd file (for the current test case node).
- Record new case...** Runs a record session for new test case, regardless of the node selected in the hierarchy tree.
- Play...** Runs a playback session. When selected on the test package node, it runs all the test cases that belong to the selected test package sequentially, as if the `-tpd` option was specified for the Replay driver. When selected on the test case node, it runs a general playback session.
- Batch play...** Runs a batch playback session. Its behavior is the same as for the “Play...” item, but with the batch mode also enabled.
- Create batch file...** Allows the user to choose which test cases inside the current test package to save into the xml batch file, which can be used later for unattended sessions, using the `-tbf` option for the Replay driver.

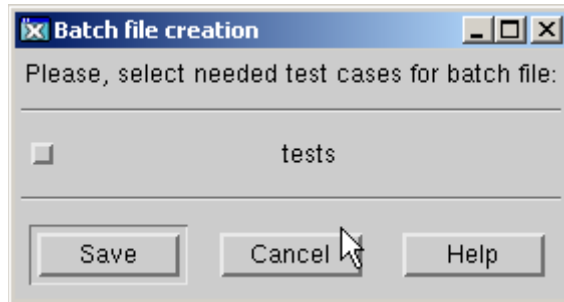


Figure 37 Batch File Dialog Box

Settings.. This item shows a dialog with session properties for a test package/test case depending on the hierarchy tree node selected. Please refer to the section “*Controlling Session Properties*” on page 103 for further information.

Window Menu

Toggle tree pane This item toggles the left tree pane visibility.

Toggle info pane This item toggles the center work area pane visibility.

Toggle snapshots pane This item toggles the bottom snapshots pane visibility.

Snapshots Pane

The snapshot pane at the bottom of the rtm window is divided into two sections - one for baselines and another for the resulting snapshots.

Selecting some test case in the tree hierarchy will cause the snapshot lists to be updated accordingly. There is a set of actions that can be performed on the snapshot items. The user can open the popup context menu by right clicking on the item in the list.

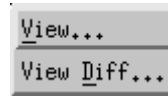


Figure 38 Snapshot Pane

The “**View**” item brings up an xwd viewer for image snapshots and the text viewer that was specified in Preferences dialog to view the snapshot content. This action can also be activated by double clicking on the list item.

The “**View diff**” item forces Replay Xcessory to generate a diff of snapshots with a given name by comparing the appropriate files from the baseline and result directory and opening the correct viewer for the diff result (xwd for image files and a text editor for logical snapshots).

Description Tab

When the user selects a node in the hierarchy tree, relevant information about the selected node is shown in the center work area (description tab). This information is stored on the disk in the tpd/tcd file of the selected node. The user can update this information by changing the title and description fields. After any change, the buttons Ok and Cancel at the bottom of the window became accessible and can be pressed to commit to or revert changes. An appropriate confirmation dialog will be shown in each case.

Tcl Files Tab

This tab contains a list of all the tcl files from the test case directory and the list widget that can display the content of a particular tcl file in highlight mode.

Clicking any tcl file that is specified in the top list of the files listing widget will update the content of the tcl file. If users want to edit the selected tcl file in some external editor, they can double click the list item to bring it onto an editor.

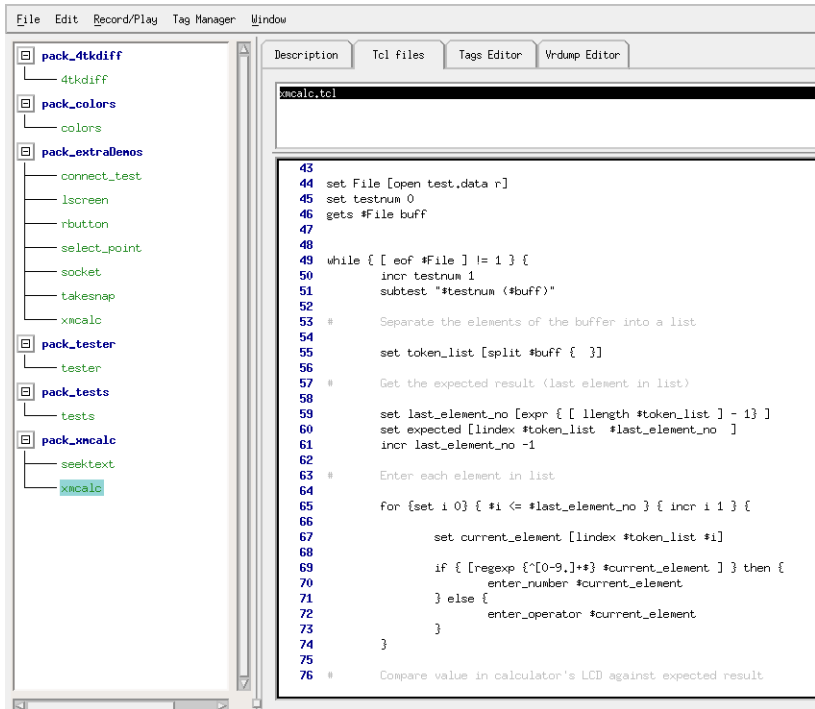


Figure 39 Tcl File Editor

Replay Xcessory Tag Manager

The Tag Manager is a part of the Replay Xcessory that helps users and test case creators create and manage tags for the widgets used in applications under test. Each application under test has its own tag file with proper keys such as:

full_widget_name.vrTag: widget_tag

Currently, users can easily create such tag files by clicking the “Learn Tags” button in the “Tag Manager” menu. For more information on this, see the description below.

THE NEW REPLAY XCESSORY TEST MANAGER

Replay Xcessory Tag Manager

It is very important for the X server (which will load and treat tag records) to have the tag file in the proper X resource format. Tag files can also be edited in hand - mode, so it is possible when a tag file has malformed records that it will not act as expected.

The tag manager can be used for creating, managing, and validating tag records in vrTag files.

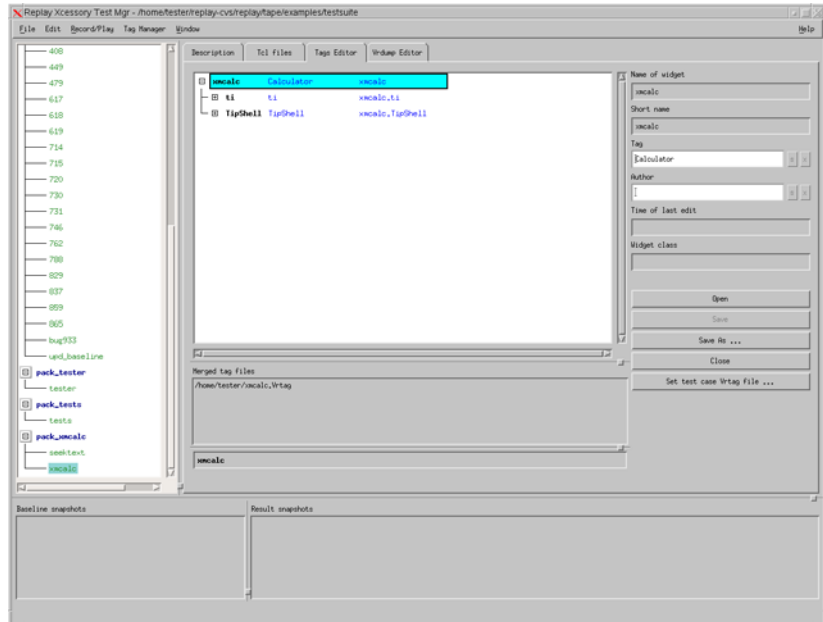


Figure 40 Tags Editor Window

There is a widget tree on the left part of the “Tags Editor” tab that represents the application’s widget hierarchy. Please note that this hierarchy can be changed if the code of application is changed. In such a case, the widget tree should be updated to include all the proper widget names and tags for the widgets that will be used while conducting record or playback sessions.

When starting the application, Replay loads and merges tags from several places. The “Merged tag files” area shows the list of tag files that will be used for the current Test Case. For more information, see **Understanding the Replay Xcessory Property Files** on page 101.

There are fields with information available for users on the right of the tree widget.

Name of the widget	The full name of the widget that contains concatenated names of each level in the hierarchy separated by a dot.
Short name of the widget	Name of the widget itself.
Tag	The alias or pseudonym of the particular widget that is currently selected in the widget tree. When a particular item is selected, this field will contain the information of the item's tag. If the user changes the tag information button to the right of the field, the field also changes. See Figure 41 .

There are two buttons with “s” and “x” letters on them. The first “s” button allows the user to confirm that the change is correct and store new tag name in the memory representation of the widget tree. When clicked, the second button with the “x” reverts the changed tag name to the what it was before the last load or confirmed change.

After confirmation of the change, the color of the changed tag will be red, so that it will be visually separated from the rest of the tags. See **Figure 42** .

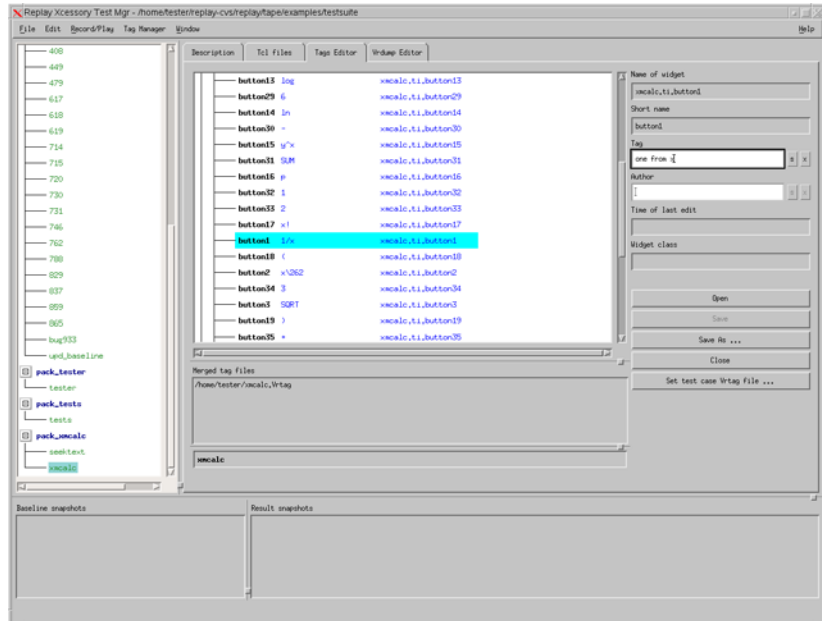


Figure 41 Tag Button After Modification

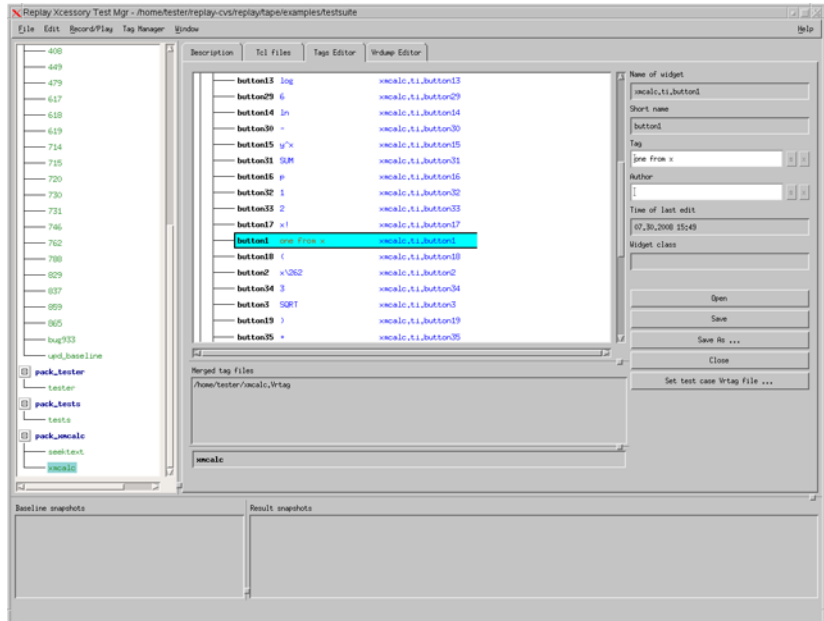


Figure 42 Tag Button After Modification Confirmation

Author

This field can be filled with any string that is to be considered the author name.

Time of Last Edit

This field is automatically filled with the time of the last change of the widget tag or author.

Widget Class

When the application is run using “Start original app” or “Start custom app” from the Tag Manager menu, this field shows the string representation of the widget class that the current widget belongs to. If the application is not accessible or the current widget cannot be found on runtime, this field will contain the message “Widget is not accessible”.

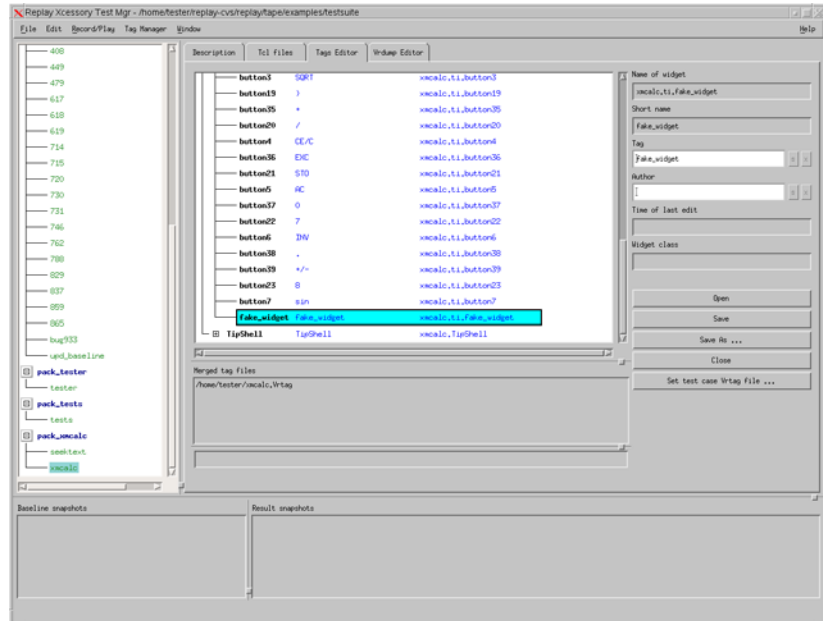


Figure 43 Information for “fake_widget”

The following buttons are at the bottom on the right side of the tree widget:

- Open** Opens the file-selection box to load tags from a file.
- Save** Saves changes in a file.
- Save As...** Opens the file-selection box to save the current tags tree in a custom file.
- Close** Closes the tags tree. If the tree was modified after it was saved, a warning message will appear. A user can choose to save changes, cancel changes, or cancel. See **Figure 53**
- Set test case Vrtag file...** Opens the file-selection box to set default tags file for a current test case. This button is sensitive only when any test case is selected in the tree on the left of Tags Editor window.

At the bottom of the tree widget there is a list called 'Merged tag files'. It shows the list of tag files to be merged for a selected test case. See "Understanding the Replay Xcessory Property Files".

You can see the "Tag Manager" sub-menu in the menu area:

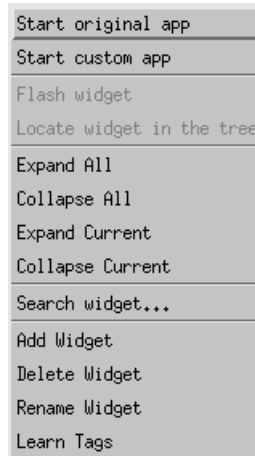


Figure 44 Tag Manager Sub-Menu

Start original app

During vrTag file generation, Replay Xcessory saves the name of the current application with the tag file parameters. This file can be easily invoked from the Tag Manager by clicking "Start original app". This allows users to see how the widget currently present on the application widget tree and the tree hierarchy built using the generated tag file in the tree widget match.

Once the application has successfully run, the Tag Manager connects to the application and allow users to perform some actions (see description below).

Sometimes it is necessary to run another application that was stored in the tag file. (For example, a user might want to give another set of parameters to the application.) The "Start custom app" item of the "Tag Manager" sub-menu can be used for that purpose. That will bring up the following dialog:

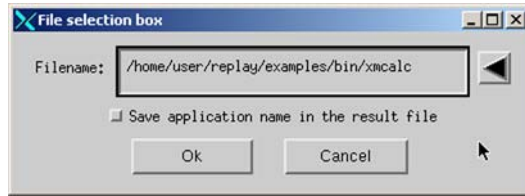


Figure 45 Start Custom App File Selection Box

This dialog box contains a text field filled in by default with the application name retrieved from the tag file. The toggle button allows the user to store the new application name (if provided) into the result tag file after the user makes changes and selects the “Save” or “Save as” button or any of the “File” sub-menu items.

After the application is started, the menu changes:

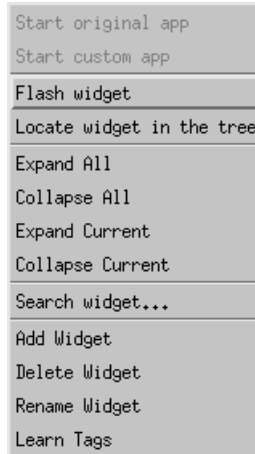


Figure 46 Tag Manager Sub-Menu After the Application Starts

Note that the “Start...” items appear to be disabled and will remain disabled until the application stops.

Flash widget

This item allows the user to employ a very helpful functionality that can be used to see the actual widget is for a particular widget in the tree. Selecting it will force the Tag Manager to search for the appropriate widget in the application widget tree and flash it three times. Each time a widget in the tree is selected, the Tag Manager will flash the widget if possible.

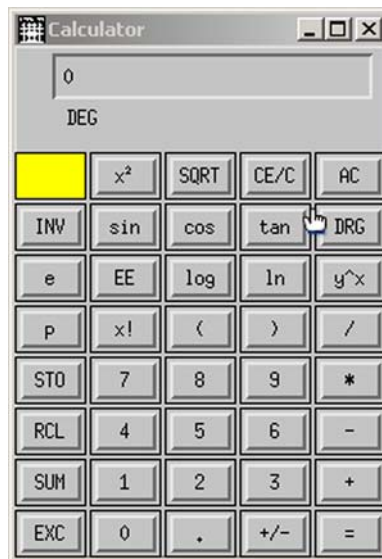


Figure 47 xmcac Flash Widget

Locate Widget in the Tree

This option is used when users know what widget they want to use, but they do not know its tag and full widget name.

After selecting this option, the status bar will show a message instructing the user what to do. The Tag Manager grabs the pointer and waits until the user clicks on some widget in the real application. If the application is accessible, it will attempt to locate this widget by full name in the widget tree currently built in the tree widget. If the searched item exists, it becomes the current item, and the correct information is filled into the fields and the scroll view is positioned to allow the user to see if the item was hidden (below or high on the tree). If the widget does not exist in the tree, the user will see that the widget can not be found. See **Figure 48**.

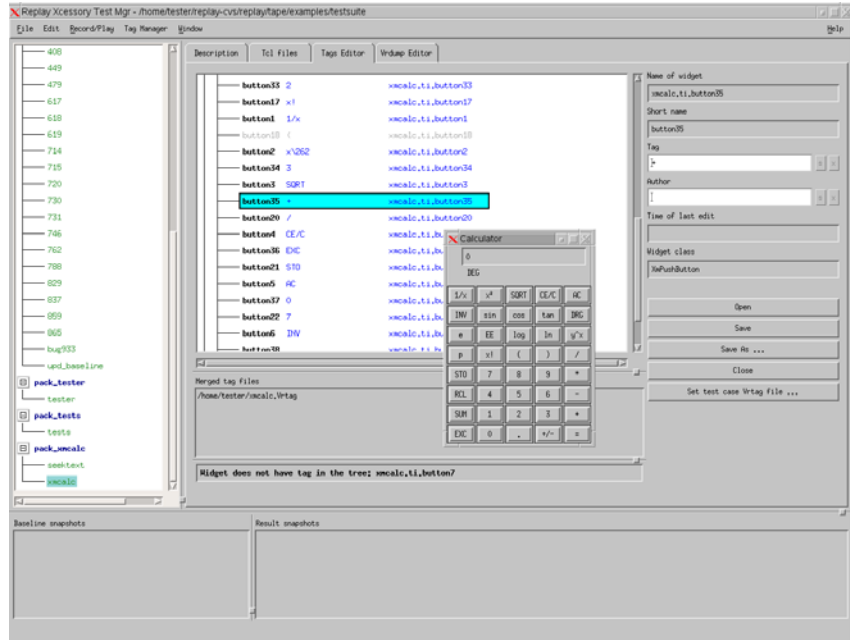


Figure 48 Failed Widget Search

Expand All	Forces the tree to expand each item on the tree.
Collapse All	Forces the tree to collapse each item on the tree
Expand Current	Expands the current item on the tree. This is only accessible if an item is selected.
Collapse Current	Collapses the current item on the tree. This is only accessible if an item is selected.
Search Widget	Searches for a widget using the widget's short name.
Add Widget	Allows the user to add a widget on a particular level of the widget hierarchy. To do this, the user has to select the widget that will be a parent for the newly created widget and select "Add Widget" from the "Tag Manager" sub-menu. The dialog shown in Figure 49 appears.

After the name fields will be filled and user click on "Ok" button Tag Manager will look for the existing children of the selected tree item and will try to check whether wanted item already exist - this should prevent users from creating duplicate items. If it is so the warning message will be shown to user with this information and propose to make the wanted item selected. Otherwise the new item will be added to the tree. Note that you can add item which has no appropriate real widget on the application side - this is not an error and could be done if such widget going to be added to the application later.



Figure 49 Add Widget Dialog Box

**Delete
Widget**

In some stages of application development, the widget hierarchy can be changed and some widgets can become obsolete. To clean the widget tree of such widgets, select the “Delete widget” menu item. After selecting this item, a confirmation widget will appear. See **Figure 50** . The currently selected widget will be marked as deleted, as symbolized by its color changing to gray. See **Figure 51** .

Note that deleted items are not really deleted from the tree and can easily be restored by editing the tag or author information, which automatically switches the item status to existing. When deleting the widget, remember that the resulting tag file after saving will not contain the string that represents the deleted widget. All of the deleted widget’s ancestors and descendants will exist and have records on the tag file. The same result can be achieved by cleaning out the tag field for a widget item.

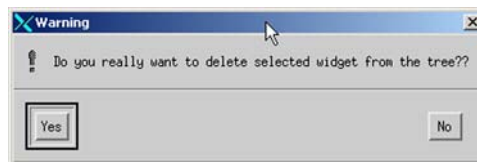


Figure 50 Widget Deletion Warning Box

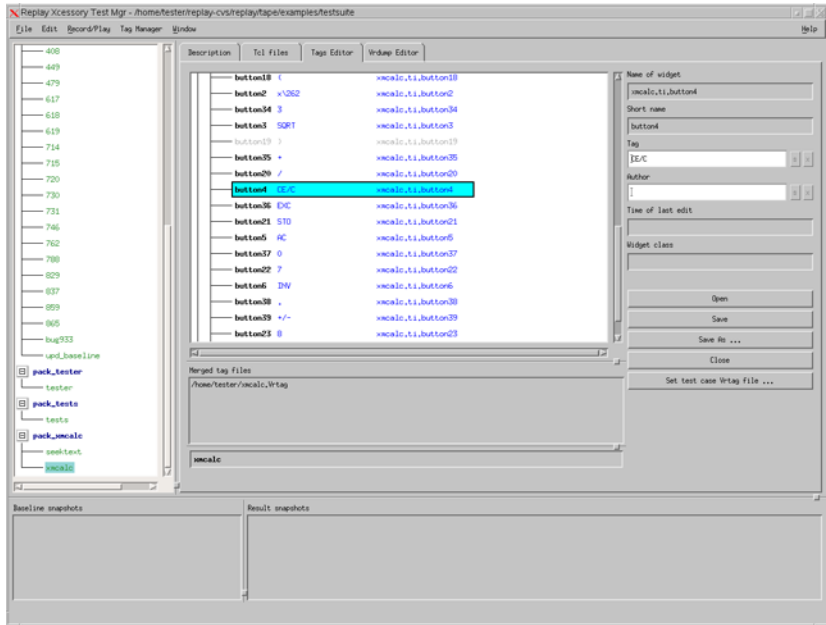


Figure 51 Widget After Deletion

Rename Widget

In case a widget name was added incorrectly or something happens to the real widget name on the application side, the user can use the “Rename Widget” functionality. See [Figure 52](#) .

This option changes the actual (short) name of the selected widget. Note that if the user changes a widget’s name, ALL of its descendants will also have their full widget names changed on the appropriate hierarchy level to contain the new parent name instead of the old name.

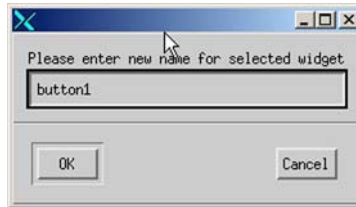


Figure 52 Rename Widget Dialog Box

Learn Tags

This menu item is only active when a client is started from TM or any test case selected in the RTM test suite tree. In the first case, the Tag Manager learns the widget hierarchy of a client, generates default tags, and loads them. In the second case, the Tag Manager starts the application related to the selected test case first.

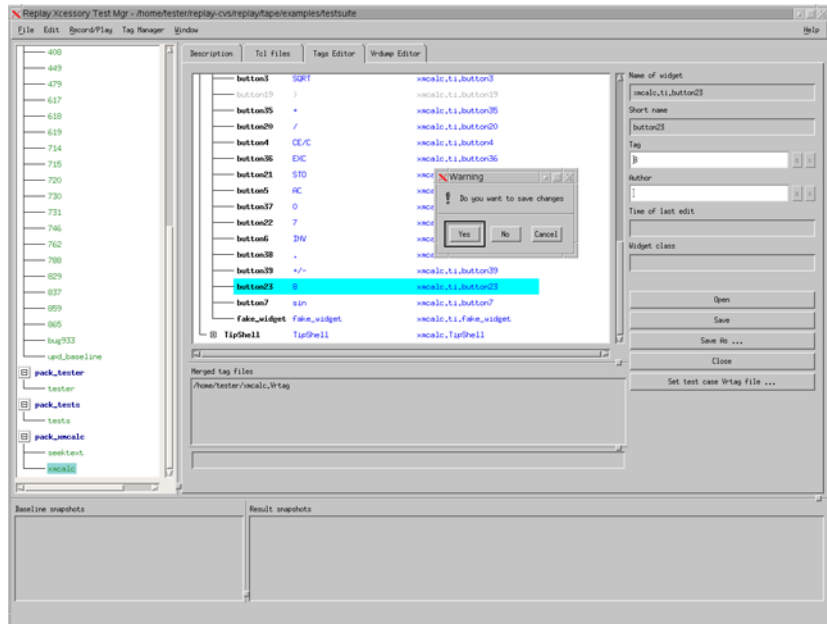


Figure 53 Save Changes Warning Box

Before using the Tag Manager, please note that there are some peculiarities in using certain widget names that can influence proper handling by the Xt library:

- Widget names should not contain spaces or tabs at the beginning and/or end of the name.
- Widget name should not be empty or consist only of spaces.
- Widget names should not have semicolon “:” in it - this symbol is used by the Xt library to separate a widget property name from its value.
- Widget names should not contain dots - this symbol is used for separating different levels of the widget hierarchy.

Although in most cases the Tag Manager can properly handle incorrect names, it is strongly recommended that users name widgets in accordance with the rules described above.

If there is an incorrect widget property / value pair, the Tag Manager prints out a warning message, but still tries to use the incorrect property/value in the widget tree.

Vrdump Editor Window

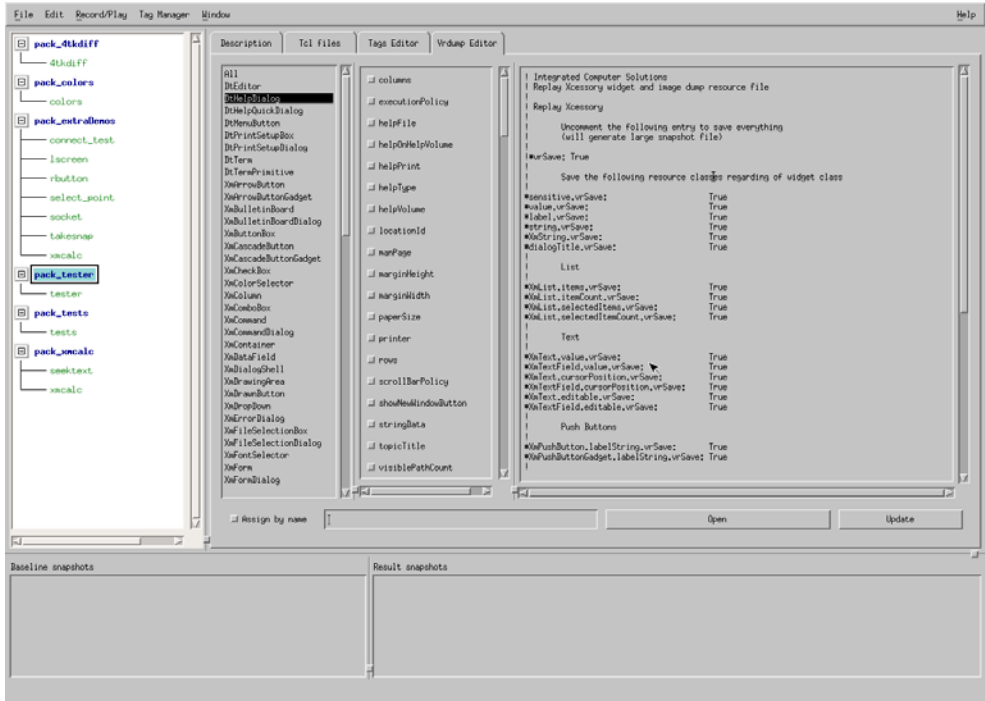


Figure 54 Vrdump Editor Window

With the Vrdump editor, the user can easily modify Vrdump files and control the granularity of snapshots (whether a specific resource should be dumped for each widget class or widget instance) in Replay Xcessory. Users can edit the ASCII control file Vrdump manually in the text editor.

By default, the Vrdump editor will load Vrdump from the `$(REPLAYHOME)/lib/app-defaults` file. Users will be able to edit this file or to select another Vrdump file from a different location. To select a new

Vrdump file, press the “Open” button in the Vrdump editor and navigate to the desired file. The UI of the Vrdump editor in Replay Xcessory consists of three parts: widget list, resource list, and Vrdump file content.

The left list will show all available widgets for the current version of the Motif toolkit. The middle list of toggles will show all available resources for the selected widget in the widget list. Vrdump file content is displayed in the text field.

To update an open Vrdump file choose the widget in the widget list and look for the resource list in the middle. All resources allowed for dump will be marked (the toggle button will be in set state). Replay Xcessory automatically searches for resources in a Vrdump file and will set all needed toggle to True. To include more resources, users can press on the needed toggles. To delete resources, users must unselect the needed toggles. Users can operate with a group of resources, but only one widget can be selected and updated at a time. The “Update” button will write all changes to the last selected widget to a file. After applying changes, the content of the rdump control file will be refreshed in the right text field.

If there is a need to dump an exact widget, users can select the widget by name. Select the widget type in the widget list and modify the set of needed resources. Enable toggle “Assign by name” and fill in the widget name in the text field. Press “Update” to apply changes. While updating, the Vrdump file widget type will be substituted with the widget name. The content of the Vrdump file will be refreshed in the right text field.

Record and Play Sessions

5

Overview

This chapter provides information for preparing a test application and conducting record and play sessions. Before starting a record or play session, refer to Chapter 3 for detailed information about the Replay Xcessory Test Manager.

Introduction

This chapter provides the detailed information needed to prepare a test application and conduct record and play sessions on the processes that make up the application. This information includes how to:

- prepare applications for Replay Xcessory
- control session properties
- control snapshot granularity
- establish tags for widget names
- conduct a record session
- conduct a play session
- use the command-line interface

Note: Before starting a record or play session, refer to Chapter 3 for information on the Replay Xcessory Test Manager, setting up the user environment, and creating and opening a test package.

Preparing Applications for Replay Xcessory

To prepare an application for Replay Xcessory, you should arrange for the application to use the instrumented Xt library, rather than the standard Xt library. To do so, either link the application with the Replay Xcessory Xt library statically, when the application is created, or link it dynamically, as part of the execution process, as explained in the following sections.

Linking Library Routines Dynamically

The most flexible approach to prepare applications is to link the Xt library dynamically. This lets you switch your application from running stand-alone to running with Replay Xcessory by changing your environment variable as necessary.

However, **setuid** programs must be statically linked with the Replay Xcessory Xt library because these programs typically ignore the library path environment variable.

Programs spawned by the **setuid** program must also be statically-linked with the Replay Xcessory Xt library as they do not inherit the library path

The name of the library path environment variable varies depending on the platform that is being used. In the following examples, we assume that the application is built with X11R6. Note that you have to specify these variables only if the case that the application is run outside of Replay Xcessory, and that it uses the “connect” method to work with the application.

- Solaris 7+ (SPARC):

```
LD_LIBRARY_PATH=$REPLAYHOME/  
lib/Xt:$OPENWINHOME/Xt:$LD_LIBRARY_PATH  
export LD_LIBRARY_PATH
```

- HP-UX

```
SHLIB_PATH=$REPLAYHOME/lib/Xt:$SHLIB_PATH  
export SHLIB_PATH
```

- Linux

```
LD_LIBRARY_PATH=$REPLAYHOME/lib/Xt:$LD_LIBRARY_PA  
TH  
export LD_LIBRARY_PATH
```

To ensure proper operation of Replay Xcessory, verify that:

- The X Toolkit, X library, and widget (such as Motif or Athena) libraries are all dynamically linked. Failure to do so can result in unresolved symbol references at run time.
- The library path environment variable will be used at run time. Linkers on some platforms have options that disable the use of the environment variable and force the use of the library that is specified at link time. HP-UX users should use the **-Wl,+s** *compiler* option, or the **chatr** command to enable the “look-up” of the environment variable, which is disabled by default. SVR4 systems (for example, Unixware or Solaris) should make sure that the LD_RUN_PATH environment variable is not used. Other linker options and environment variables may apply to other specific platforms.

Multiple Library Paths

Dynamically linked multiple application processes usually have the same library path requirements for Replay Xcessory. However, one application may be based on Motif and another based on non-Motif widgets. If this situation occurs, potential problems can be avoided by statically linking one application with the appropriate Replay Xcessory instrumented Xt library.

Linking Library Routines Statically

If your application is statically linked with the Xt library, you need to relink with the Replay Xcessory Xt library. To do so, update your Makefile to find the Replay Xcessory static Xt library, for example, the `$REPLAYHOME/lib/Xt` directory.

Checking Dynamic Dependencies

Under certain platforms, commands are available to verify that an executable was dynamically linked with Xt. On Solaris, the `ldd` command shows which one will be dynamically linked with the Replay Xcessory Xt shared library. `ldd` lists dynamic dependencies, using the same algorithm as the dynamic linker to locate shared objects.

For example, the command to check library dependencies for the `xmcalc` program shows the following:

```
ldd ./xmcalc

libXmu.so.6 => /usr/X11R6/lib/libXmu.so.6
(0x4008c000)
libXt.so.6 => /opt/Replay/lib/Xt/libXt.so.4
(0x400a2000)
libXext.so.6 => /usr/X11R6/lib/libXext.so.6
(0x40163000)
libX11.so.6 => /usr/X11R6/lib/libX11.so.6
(0x40171000)
libXpm.so.4 => /usr/X11R6/lib/libXpm.so.4
(0x403ca000)
libXm.so.3 => /usr/X11R6/lib/libXm.so.3
(0x403d9000)
```

The third line indicates that this `xmcalc` uses the Replay Xcessory Xt library.

Note: If you are using HP-UX, use the `chatr` command. Contact HP for specific information regarding this command. Unlike `ldd`, `chatr` lists the dependent libraries specified at link time, not how they would be resolved given the current value of the library path environment variable.

Excluding Applications

There may be applications that should not be included for recording, even though the application is running with the Replay Xcessory Xt library. By adding an application's name to the **.Replay.excludeApps** resource, any application with that name is excluded from the record session. For example:

```
Replay.excludeApps: mwm xclock replay replaytm
```

ignores actions on all instances of the **mwm** window and the **xclock** application, even if they are linked with the Replay Xcessory Xt library. The exclusion facility affects applications started by Replay Xcessory as well as applications which are connected to it after they have been started. Replay Xcessory must be in the list of excluded applications, or it might cause Replay Xcessory to hang up.

Understanding the Replay Xcessory Property Files

Replay Xcessory properties can be customized to suit specific testing requirements and environment preferences. These properties are stored as ASCII files that can be modified using the Replay Xcessory graphical user interface (GUI) or standard text editors.

The Replay Xcessory properties include:

- session properties — **.Replay**
- snapshot properties — **.Vrdump**
- widget tag file — **.Vrtag**

Properties can be defined as:

- one generic set that is appropriate to any applications under test
- a second set that is application-specific

The two sets have the same syntax and differ only in the file name; the application-specific set uses the application name as a prefix. For example:

Table 1: Property Filenames

Property	Generic	Application-specific
Session Property	.Replay	None
Snapshot Property	.Vrdump	<i>appname.Vrdump</i>
Tag File	.Vrtag	<i>appname.Vrtag</i>

These property files can be located in a number of directories. The effective set of properties is the merging of the effective application-specific file with the effective generic property file. The active file is the first property file found in the following search sequence:

1. -tcd level (if override is selected).
2. local directory—When using the Test Manager, the open test package is *always* considered the local directory.
3. directory pointed to by the \$REPLAY_TESTSUITE_PROPS environment variable.
4. directory pointed to by the \$REPLAY_PROPS_DIR environment variable.
5. directory pointed to by the \$HOME environment variable.
6. \$REPLAYHOME/lib/app-defaults if the \$REPLAYHOME environment variable is set.

\$REPLAY_TESTSUITE_PROPS is an environment variable that should be set to the directory containing the test suite-wide property files; it should be made into a special test package in the test suite. It is also recommended that, whenever possible, set up procedures should specify all properties in the test suite-wide property files. Scripts that require special properties should be placed in the local test package or directory in order to override the test suite-wide property.

To search the sequence for specific Vrtag files:

1. \$REPLAYHOME/lib/app-defaults/Vrtag.
2. appVrtagFile option in TCD-file.

3. *Test Case dir/appname.Vrtag.*
4. `$REPLAY_TESTSUITE_PROPS/appname.Vrtag` or
`$REPLAY_PROPS_DIR/appname.Vrtag.`
5. `$HOME/appname.Vrtag.`
6. `$REPLAYHOME/lib/app-defaults/appname.Vrtag.`

Controlling Session Properties

The properties window provides the controls to specify session properties that Replay Xcessory saves in a **.Replay** file for a test package, and in a **.Replay_Appname** file for a test case. This window can be accessed through the Test Manager by selecting **Open Test Package** under the **Test Package** menu, and then **Test Package** and **Session Props** from the Test Package window.

This window can also be accessed during a record or playback session on stage just before recording and playback starts when the user switches on the toggle “Override package settings” dialog.

The property window consists of three pages:

RECORD AND PLAY SESSIONS*Controlling Session Properties*

Here are three illustrations of the properties window:

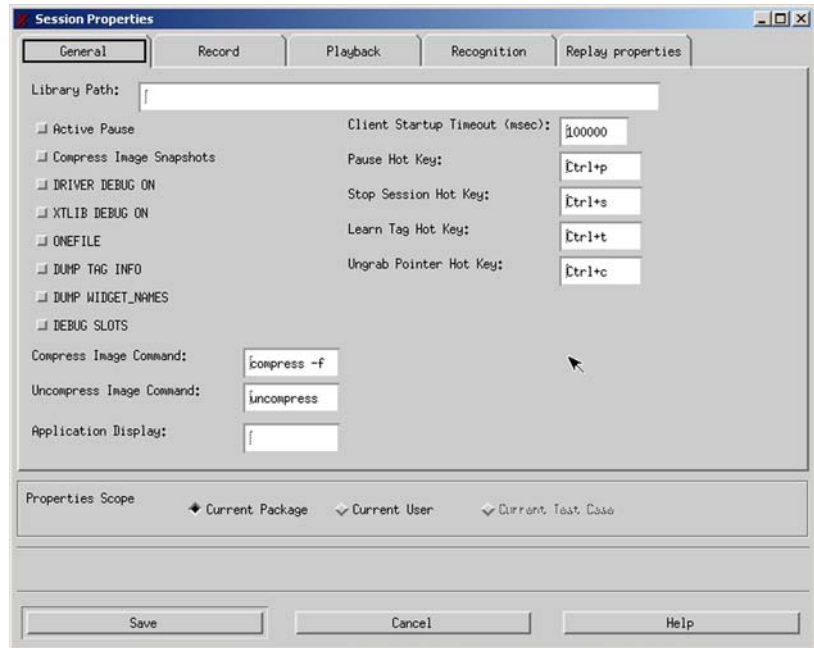


Figure 55 Session Properties Window, General Tab

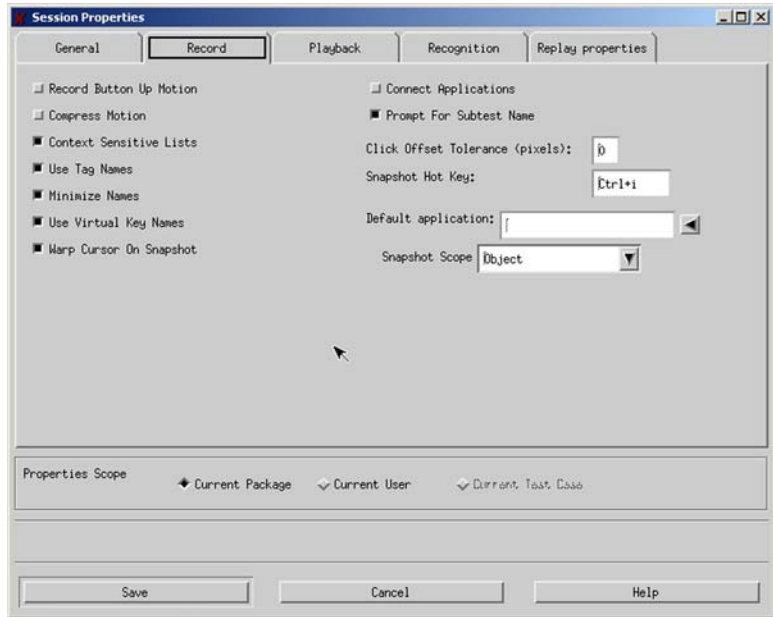


Figure 56 Session Properties Window, Record Tab

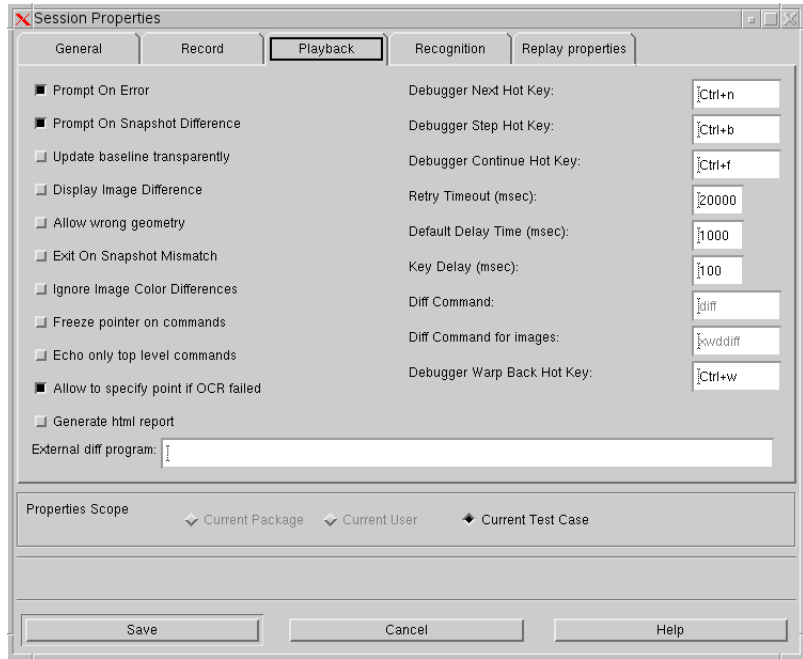


Figure 57 Session Properties Window, Playback Tab

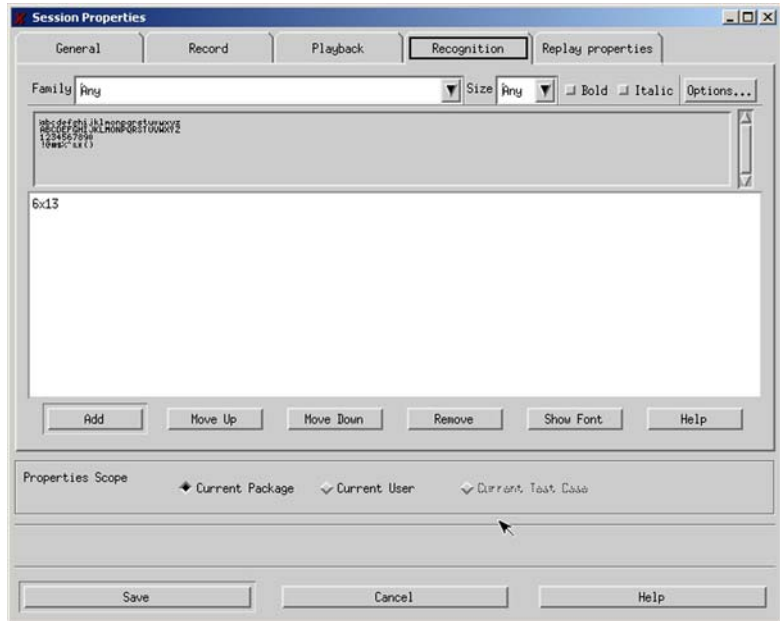


Figure 58 Session Properties Window, Recognition Tab

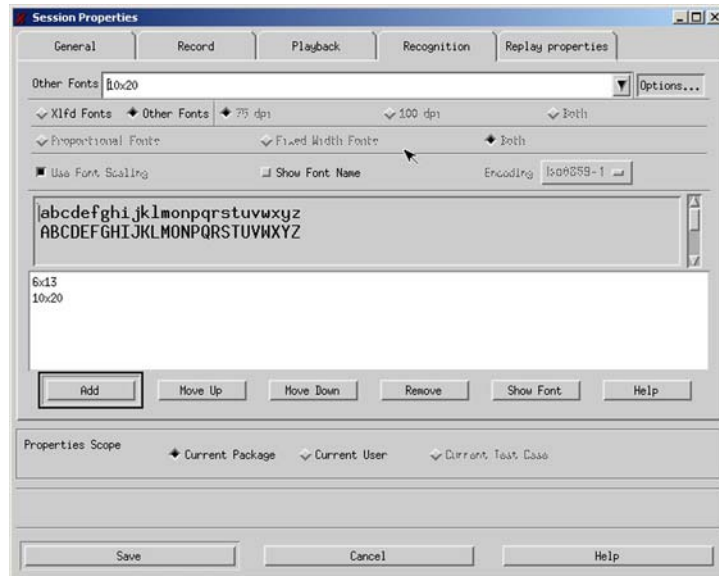


Figure 59 Session Properties Window, Recognition Tab (additional fonts)

During a record or play session, default properties determine interface behavior; for example, a system-default property establishes **Ctrl+i** as the hot key for a snapshot request during a record session. In addition, default properties determine whether record or play checkboxes are initially set on or off when the record or play control panel appears. For example, the **Record Button Up Motion** button typically comes up unchecked when a record session is started.

Many of the properties specified here override system default properties, and in turn can be overridden by properties specified when a session is begun.

Note: You have the ability to see what a particular property is. Simply mouse over any option and you will see popup information about what the option is intended for.

Invoking the Session Properties Window

Detailed explanations of the default options and property controls are described in the following sections.

Specifying Hot Keys

Hot keys are keyboard shortcuts that can be entered for some common actions, such as taking a snapshot, toggling the pause mode, etc. Hot keys are important because they allow key actions to be invoked without having to bring the Record or Play Control Panel to the front when it may be currently obscured by the process under test or even iconified in order to save on screen space.

Hot keys are specified in the form of a list of tokens separated by plus signs. Thus, **Ctrl+i** indicates that the hot key is composed of the **Ctrl** key and the letter **i**. Multiple modifiers, such as **Shift+Ctrl+i** are also allowed.

Hot keys are entered while the cursor is on the application under test. For this reason, make sure that a hot key sequence does not conflict with a sequence that the process uses.

Modifying Properties

The Test Manager creates a **.Replay** file that saves the properties from one session to the next. By clicking on **Current Test Package** or **Current User** you specify the directory where Replay Xcessory is to save the **.Replay** file. Choose **Current Test Package** if these options should apply to this test package only; choose **Current User** if the options should apply to all your Replay sessions. These toggles are only accessible if the property window is invoked from the package menu. If it is invoked from the test case definition dialog, the only available option is to store modified settings in the test case's properties file. When you have finished making modifications, press the **OK** button. The Test Manager creates the **.Replay** file and saves it in the specified directory.

Properties Controls

The descriptions for the properties controls are contained in the following sections according to the following categories:

- general properties
- record session properties
- play session properties
- the scope of properties
- properties window buttons

Additional information about the arguments used to set these properties can be found in “*Accessing Run Time Parameters*” on page 219.

General Properties

The following entries apply to both record and play sessions:

Library Path This directory contains the Replay Xcessory (instrumented) Xt library. When looking for Xt library routines to load dynamically, the system begins its search at the directory you specify here. This entry takes precedence over the current setting of the library path environment variable. (Replay Xcessory resets the library path environment variable, or its equivalent, to start with this entry.)

Active Pause Allows processes under test to be operated during pause mode while recording a play session. Mouse and key interactions with the application under test will not be recorded.

Use this option carefully, as it is possible that the nonrecorded interactions may substantially change the application state and result in invalidating a play session; for example, popping down a dialog box while in **Active Pause** could be dangerous because the script is unaware that the pop down has occurred. The inconsistent state could result in a false snapshot comparison failure. If the application state is changed, the original state should be restored before continuing record or playback.

Use Tag Names Requests that widgets be identified by their tags whenever possible. If a tag name cannot be found or would result in a duplication, the internal widget name is used. This option affects both the script and the logical snapshots.

Compress Image Snapshots Requests that image snapshots be stored in a compressed format to save disk space.

Compress Image Command This command compresses image snapshots (system default: **compress -f**).

Uncompress Image Command This command uncompresses image snapshots (system default: **uncompress**).

Application Display	The display on which the application is running.
Client Startup Timeout	Maximum application start up time, in milliseconds. If the application fails to start up within the specified time, the replay driver exits (system default: 100000 msecs).
Pause Hot Key	Key combination to be used to request a pause in a session (the default hot key is Ctrl+p).
Stop Session Hot Key	Key combination to be used to terminate a session (the default hot key is Ctrl+s).
Learn Tag Hot Key	Key combination to be used to start the Learn Tag mode.
Ungrab Pointer Hot Key	Key combinations used to ungrab the cursor if it is currently grabbed by Replay Xcessory or by the process under test.
DRIVER_DEBUG_ON	Switches on debugging of the Replay Xcessory driver. This variable forces Replay Xcessory to produce the log file for processing TCL commands on the Replay Xcessory side (parsing TCL script and going to execute commands). The log file that will be created will have name like driverlog.4122, where 4122 is the pid of the process. The log file is created in /tmp directory. See also the ONEFILE variable description since it affects the log file name.
XTLIB_DEBUG_ON	Switches on debugging of the customized Xt driver. This variable forces customized libXt to produce the log file for the actual processing of the test script commands on the client side. The log file that will be created will have name like xtliblog.4123 where 4123 is the pid of the process that has the current libXt instance in its memory map. The log file is created in /tmp directory. See also the ONEFILE variable description since it affects the log file name.

- ONEFILE** Switches the log reporting manner. It forces all logs to be dumped in one log file: /tmp/xtliblog for the customized libXt and /tmp/driverlog for the driver itself. When this variable is specified, Replay Xcessory will not create a log file for each session, each instance of Replay Xcessory, and each application under test. It rather will dump all the information in one file for libXt and one file for Replay Xcessory. Note that files generated before specifying this variable will still remain in the temp directory but will mostly be obsoleted. Also note that log files could be quite big, thus making it harder for the support team to perform the analysis when investigating a certain issue. Please remove old log files from the /tmp/ directory before you create new ones when reproducing an issue.
- DUMP_TAG
_INFO** Helps to debug issues with tag handling. This variable forces Replay Xcessory to dump additional information about tag name generation and processing into the xtliblog file. It should be turned on in case Replay Xcessory can not find some widget in the application being tested but a user is sure that the TCL script file is correct and widget exists.
- DUMP_WID
GET_NAME
S** Helps to debug issues with snapshots. This variable can be very useful when taking logical (resources) widget snapshots. Its main purpose is to dump the names of the widgets being processed to standard output. It can help in case some widget property contains a value for which there is custom type converter registered in Xt by the application being tested. So, if this application crashes while taking a widget snapshot (most likely because of the issue in type converter) a user could specify this variable and get the list of widgets while Replay Xcessory processes them. The last name in the list before the application crashed will most likely identify the problem widget.

DEBUG_SL OTS	Switches additional logging of the name generation process. If a user has problems with using tags in the applications under test, this variable can force Replay Xcessory to generate additional log statements into the xtlblog file, which is very useful when debugging tag issues. It should be used in the case that either the assigned or the default tag suggested by Replay Xcessory has a [#] suffix at the end of the tag name, but the user thinks that it is incorrect or excessive.
Xnest/Xvfb depth	Specifies the size of the nested/virtual X server that will be used for the playback session if the appropriate option was selected by the user.
Xnest/Xvfb dimension	Specifies the color depth of the nested/virtual X server that will be used for the playback session if the appropriate option was selected by the user.
Xnest/Xvfb window manager	Specifies the window manager for the nested/virtual X server that will be used for the playback session if the appropriate option was selected by the user.

Record Session Properties

The following controls apply only to record sessions.

Record Button Up Motion	Records all button-up movements. For most processes, leave this property off; mouse movements when mouse buttons are up are typically meaningless. Applications with certain modal interfaces are exceptions. For example, turn this on for drawing tools that draw with mouse buttons up (system default: off).
Compress Motion	When off, this option records all intermediate coordinates reported by the X server when tracking mouse movements. When on, it records only the ending coordinate, which shortens the script (system default: off).
Context Sensitive Lists	Specifies that the Motif list item being acted on be recorded using the string listed in the item itself. This allows the same item to be selected on playback even if the font or item position changes. Actions on OSF/Motif list widgets use the context sensitive recording by default.
Use Tag Names	Requests that widgets be identified by their tags whenever possible. If a tag name cannot be found or would result in a duplication, the internal widget name is used. This option affects both the script and the logical snapshots.
Minimize Names	Requests minimized widget names in scripts and widget snapshots; otherwise, fully-qualified names are used. See Chapter 2 for further information on widget names (system default: on).
Use Virtual Key Names	Specifies that key symbol (X Keysyms) be recorded using the OSF/Motif key symbol names for maximum script portability across displays that use different key bindings (for example, backspacing makes use of the Backspace key on some displays, but uses the Delete key on others). Recording using the virtual key name (in this case, osfBackSpace) ensures that the script is portable to both display types. This option is valid only for Motif applications. This is turned on by default.

Warp Cursor on Snapshot	Replay Xcessory normally moves the cursor out of the way before taking a snapshot. This precaution generally results in more reliable snapshot comparisons (fewer false negatives). Turn this option off only if the cursor warping generates an undesirable side effect.
Connect Application	Connect to all existing applications linked to the Replay Xcessory X Toolkit library.
Prompt For Subtest Name	If turned on, Replay Xcessory will prompt for the name of the subtest whenever the subtest button is pressed in the Record Control Panel. If turned off, Replay Xcessory numbers the subtests automatically, starting from 1 (system default: off).
Snapshot Scope	Specifies the scope of the snapshot to be taken whenever a snapshot is requested using the snapshot button or hot key. The possible values are Object , Window , Viewable , and Full . Refer to “ <i>Controlling Snapshot Scope and Granularity</i> ” on page 122 for additional information.
Click Offset Tolerance	Maximum number of pixels that the pointer can move between the press and release of a mouse button and still be considered a click or a multclick.
Snapshot Hot Key	Key combination to be used to request snapshots; the default is Ctrl+i . When you press this key combination during a record session, the Test Manager takes a snapshot.
Default Application	Path to the application that Replay Xcessory will use as the default application under test when composing a new record session.

Play Session Properties

The following controls apply only to play sessions.

Prompt on Error	<p>If set to true, a prompt dialog will appear whenever an error occurs. The dialog provides two options: The errors can be ignored, or execution of the command can be suspended.</p> <p>If set to false, the prompt dialog does not appear and execution of the command continues; the command being executed returns an error return code when appropriate.</p>
Prompt on Snapshot Difference	<p>Requests to be prompted whether to continue or to exit following a snapshot comparison mismatch (system default: on).</p>
Update Baseline Transparently	<p>In the case that the snapshot comparison fails, Replay Xcessory automatically updates the baseline file despite what the value of the "Prompt On Snapshot Difference" setting (system default: off).</p>
Display Image Differences	<p>Requests that image differences be displayed in an independent window (system default: on).</p>
Allow Wrong Geometry	<p>Causes Replay Xcessory to compare the results and the baseline by matching the largest possible geometry area of both snapshots and returns the appropriate result based on whether those areas match.</p>
Exit On Snapshot Mismatch	<p>Requests that a play session be stopped when a snapshot difference is encountered (system default: off).</p>
Ignore Image Color Differences	<p>Ignores color differences when comparing images (system default: off).</p>
Freeze Pointer on Commands	<p>Enabling this option causes the cursor to be returned to its starting position after Replay executes a command. Without this option Replay will move the cursor to the application under test to execute a command but not return the cursor to its original location.</p>

Echo Only Top Level Commands	If selected, it only allows notes about commands that are not contained inside nested TCL bodies (e.g. procedures, loops, and condition statements) to be echoed into Replay Xcessory's history window.
Allow to Specify Point if OCR Failed	If selected and the "seektext" TCL command is unable to locate a needed string, this allows the user to choose a point on the application surface by clicking the left mouse button to be returned as the result coordinates of the seektext command. Replay will wait ten seconds for input. The status bar will blink once a second.
Generate HTML Report	If set to true, Replay Xcessory will generate an additional HTML report file in the test case directory. If set to false, no additional HTML report file will be generated. This option does not affect the generation of text report files.
Debugger Next Hot Key	Hot key for the "Next" command in the playback debugger. The hot key combination is Shift+n by default.
Debugger Step Hot Key	Hot key for the "Step" command in the playback debugger. The hot key combination is Shift+s by default.
Debugger Continue Hot Key	Hot key for the "Continue" command in the playback debugger. The hot key combination is Shift+c by default.
<hr/>	
	Note: You can use hotkey combinations now both on the driver side and on the application under test side.
<hr/>	
Retry Timeout	Maximum delay time, in milliseconds, to allow for widgets to map (system default: 20000 msec). If the expected widget does not map within the timeout period, a prompt dialog will be displayed in interactive playback if Prompt On Error is set to true.
Default Delay Time	Time to be used when the delay time is not explicitly included in a script command (system default: 1 sec.).
Key Delay	Time, in milliseconds, to simulate typical lapse between entries of characters (system default: 100 msec).

Diff Command	Command to compare the ASCII logical snapshot files (system default: diff).
Debugger Warp Back Hot Key	Hot key to return the pointer to the debugger control panel. The hot key combination to control warping is Ctrl+w .
External diff program	Diff-like application. Click the “ Ext diff ” button in the Snapshot Mismatch dialog to view the text differences in the external diff viewer.

Properties Recognition

On this page, you can select any of the fonts currently installed in your system that are available for Motif applications. Each selected font will be added to the list of fonts that will be used in each “seektext” invocation for searching the text strings on the widget surface.

The text field under the dropdown combo box shows symbols from the currently selected font. The string list below that shows the list of currently selected fonts. The user can apply each of these fonts to the upper text field by double clicking to see what the current font is.

The current functionality provides tremendous flexibility in XLFD (X Logical Font Description) choices. Clicking the Options button displays an additional panel of controls. This allows access to non-XLFD fonts, control of the resolutions of the chosen fonts, choice of fixed or proportional fonts only, removal or use of font scaling, selection of different font encoding, and dynamic display of the XLFD name of the font which is constructing.

By choosing the “Other Fonts” toggle from the option panel, the Family and Size lists, as well as the Bold and Italic toggles, are replaced with a combination box containing all non-XLFD fonts available on your system. This feature allows users to select non-XLFD fonts. A string entered in the text field of the combination box is interpreted as a font name. You can also enter XLFD names manually. The combination of this feature and the other options allows you to display any font on the entire system.

As a default, the standard resolution that is closest to the current display is selected. Choose a font of a different resolution or display both resolutions to allow a wider range of choices.

In most cases, the fact that a font is a fixed width or proportional is not important to the user. However, some applications require a fixed width font, such as terminal emulators, and many users prefer proportional fonts for appearance purposes. The font selector allows users to limit the font choices to fixed width or proportional or to allow both.

The font scaling technology uses bitmap scaling which, although useful in some cases, generally results in very ugly fonts. Users often want to know which fonts are scaled and which ones exist as hand crafted bitmaps. To remove the scaled fonts from the list of choices, toggle off the “Use Font Scaling” button.

Clicking the Show toggle displays the current font's XLFD name at the bottom of the font selector.

Statistic Options

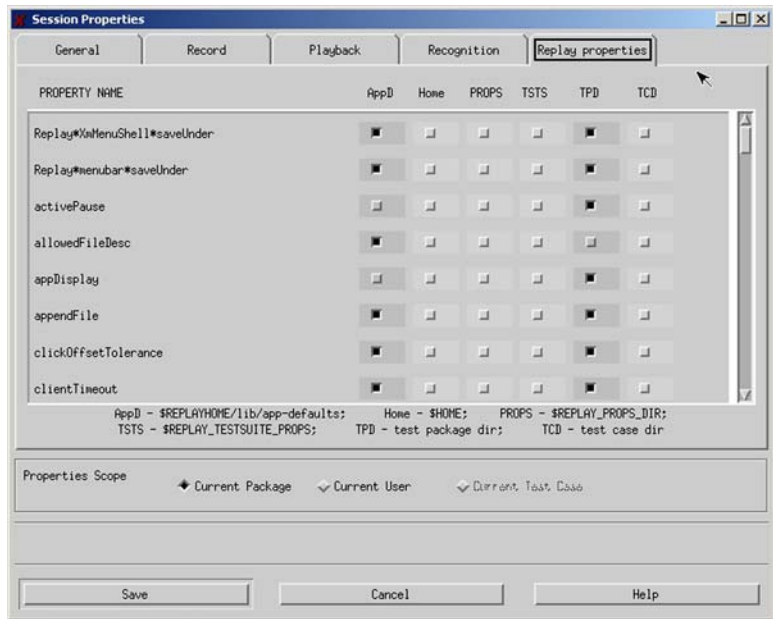


Figure 60 Replay Properties Page

The Replay properties page represents the combined statistical information of all the Replay settings stored in the configuration files visible by the Replay Xcessory driver. Those settings could be stored either in “Replay” (for

app-defaults) or “.Replay” files for all other directories. This page can be useful to trace down exactly which setting works at the current moment for a particular package session or test case session.

The order of how settings apply is the following:

1. \$REPLAYHOME/lib/app-defaults if the \$REPLAYHOME environment variable is set (it is set automatically set by Replay, but Replay Xcessory do not override it if it already exists when the Replay Xcessory driver starts, so be sure that it is either not set or set to the proper location).
2. Directory pointed to by the \$HOME environment variable. If this directory contains a .Replay file, it will be sourced and all the settings merged to an existing X resource database.
3. Directory pointed to by the \$REPLAY_PROPS_DIR environment variable. If this directory contains a .Replay file, it will be sourced and all the settings merged to an existing X resource database.
4. Directory pointed to by the \$REPLAY_TESTSUITE_PROPS environment variable. If this directory contains a .Replay file, it will be sourced and all the settings merged to an existing X resource database.
5. Test package directory. Settings that can be customized using the Replay Xcessory Test Manager.
6. Tcd level. If the “Override Test Package Settings” button is selected on the Replay Xcessory driver dialog, the sixth setting is most weighty.

The left column of the statistic table holds a list of all the settings that exist and are understandable by Replay Xcessory that could act on the recording and playing back of saved test cases.

The titles of the following columns are short variants of the places where the appropriate settings were found. The bottom of the table contain matches for short and long variants.

AppD = \$REPLAYHOME/lib/app-defaults/Replay

Home = \$HOME/.Replay

PROPS = \$REPLAY_PROPS_DIR/.Replay

TSTS = \$REPLAY_TESTSUITE_PROPS/.Replay

TPD = test_package_dir/.Replay

TCD = test_case_dir/.Replay_test_case_name

TCD = test_case_dir/.Replay_test_case_name

If the appropriate setting is set in a particular file, then in the cell that corresponds to the property name row and column there will be a toggle button with the status of “set”. Otherwise, the status will be “unset”.

Properties Scope

The following check buttons control the scope of properties:

- | | |
|------------------------------|--|
| Current Test Packages | Stores the .Replay file containing these properties in the current directory, rather than the home directory (system default: on). |
| Current User | Stores the .Replay file containing these properties in the home directory, rather than the current directory (system default: off). |
| Current Test Case | Stores the .Replay_appname file containing these properties in the test case directory (system default: on when invoked from test case dialog). |

Properties Window Buttons

The following push buttons affect the properties window only:

- | | |
|--------------|---|
| OK | Terminates properties modification and saves these properties in the .Replay file. |
| Close | Dismisses the properties window, discarding these changes. |
| Help | Brings up an explanation of these options. |

There are several additional resources that are not accessible through GUI but can be modified in **[.]Replay** file directly.

- | | |
|--|---|
| Replay.enforceNames | If True, enforces the file naming conventions (system default:true). |
| Replay.omitDelay | If True, omits delays between user actions on the record stage and on playback uses the default value of 1 sec. (system default:false). |
| Replay.useCurrentWindow | If True, always saves the current window and checks if it is active before further commands (system default:True). |
| Replay.omitCoordinatesWidgetClasses (list of widget classes) | Omits the recording of window coordinates on actions on widgets of listed classes. |
| Replay.omitButtonUpMotionWidgetClasses (list of widget classes) | Forces Replay to not save the movement of the mouse pointer on the area of widgets of the listed widget classes. |

Replay.maxDelay Value passed to the embedded Tcl interpreter, and represents the maximum delay for it.

Controlling Snapshot Scope and Granularity

Replay Xcessory provides maximum flexibility in the contents of each snapshot; two concepts are involved, snapshot *scope* and snapshot *granularity*.

Snapshot Scope

Snapshot *scope* refers to the portion of the application widgets that should be stored in the snapshot. The following snapshot scope levels are supported:

- A *full* snapshot includes all of the current widgets in the application. This type of snapshot can be large and may include widgets (such as unmapped widgets) which are not visible to the end user.
- A *viewable* snapshot is similar to a full snapshot except that only those widgets that are visible to the end user are saved.
- A *window* snapshot is similar to a viewable snapshot except that the widget tree starts from one specific window which must be identified at the time the snapshot is taken. Menus, if currently popped up, are included in the snapshot. (The term “window” here corresponds to a shell widget in the X Toolkit terminology, not a menu shell.)
- An *object* snapshot is similar to a viewable snapshot except that the widget tree starts from one specific widget which must be identified at the time the snapshot is taken. The widget tree is more often than not a single widget. Object snapshots are the most efficient because they only save the information about the widgets that are relevant to the test.

Snapshot Granularity

Snapshot *granularity* refers to the ability to control which widget resources to include and whether the snapshot will include image snapshots. Snapshot granularity is typically specified by widget class or resource class, for example, saving the label resource of all push button widgets or saving all resources of type string.

In some cases it may be necessary to override the class-based specification because specific widget instances have unique requirements, for example, the text in all labels is normally desired but one specific label widget that displays the current time might not be. The final contents of the snapshot specification depends on the intersection of the snapshot scope and the snapshot granularity.

Controlling Widget Resources

An ASCII control file, **.Vrdump**, controls which widget resources to include in a snapshot. Replay Xcessory provides a default **.Vrdump** file. You can edit a copy of **.Vrdump** to customize which resources are included, as appropriate for your application.

In most cases the system default **Vrdump** file (from **app-defaults**) can be used as is. This **.Vrdump** file dumps those resources that are most likely to change or represent a widget state and have significance from a verification point of view for most Motif widgets; these include most text strings.

If customized **.Vrdump** files are run for all tests, then this version should be kept in a location pointed to by the **REPLAY_TESTSUITE_PROPS** environment variable; otherwise, a dump specification that must be specific to a particular test script should be kept in the test package directory.

The **.Vrdump** file defines a snapshot granularity that is to be used generally throughout the record or play session. In some cases, you may find it useful to modify the snapshot granularity in the middle of the session. Modifying granularity can be accomplished using the **Snapshot Spec** button in the **Record Control Panel**.

Invoking the Snapshot Properties Editor

To change snapshot properties from the Test Manager, select **Snapshot Props...** from the **Options** menu in the test package window. A text editor containing the **.Vrdump** file for the current test package will be started. If there is no current **.Vrdump** file, one will be copied from one of the standard locations.

A portion of the snapshot properties window is shown in the following illustration:

```

//work/replay/tape/xcessory/examples/testsuite/pack_xmcalc/.Vrdump
! Integrated Computer Solutions
! Replay Xcessory widget and image dump resource file
!
! Replay Xcessory
!
!       Uncomment the following entry to save everything
!       (will generate large snapshot file)
!
!*vrSave: True
!
!       Save the following resource classes regarding of widget class
!
!*sensitive.vrSave:                True
!*value.vrSave:                    True
!*label.vrSave:                    True
!*string.vrSave:                   True
!*XmString.vrSave:                 True
!*dialogTitle.vrSave:              True
!
!       List
!
!*XmList.items.vrSave:              True
!*XmList.itemCount.vrSave:         True
".Vrdump" 66L, 1471C

```

Figure 61 Snapshot Properties Window for **xmcalc1** Snapshot

Note: An error message displays if the Snapshot Props... menu is requested and no snapshot properties are found.

Dump Specification Format

The **.Vrdump** file has the same format and rules used for the X resource files (such as **.Xdefaults**). However, the **.Vrdump** file is not a resource file and so standard X resources should not be placed here.

- Each line is an attribute/value pair separated by a colon (:) and terminated by a new line.
- Comment entries should start with an exclamation mark (!) and end with a new line.
- A period (.) separates adjacent components.
- A question mark (?) can be placed between two periods to represent a single component; this notation is not available for applications linked with X11R4 or earlier.
- The asterisk (*) represents zero or more components. Unlike the question mark, however, periods are not needed around the asterisk.

Snapshot Types

There are two types of snapshots:

- *Widget* snapshots are ASCII dumps of the widget tree of an application under test.
- *Image* snapshots are window dumps of the specified widget window in the standard **xwd** format.

Both widget and image snapshots can be specified by widget class or by widget instance. (See “*Snapshots*” on page 41.)

Controlling Widget Snapshots

Widgets can be controlled in several ways. The following sections explain widget control by:

- widget class-based specification
- widget instance-based specification
- resource type-based specification

The following topics are also discussed:

- Replay Xcessory pseudo resources
- snapshot file format

Widget Class-Based Specification

The **.vrSave** boolean resource controls which resource names will be dumped for each widget class or widget instance. A widget class specification takes the form

```
application_class.widget_class.resource_name
    .vrSave: [True|False]
```

The following establishes which Motif push button resources to include in widget snapshots:

```
*XmPushButton.labelString.vrSave: True
*XmPushButton.armPixmap.vrSave: True
*XmPushButton.sensitive.vrSave: True
```

The application class is usually left out and replaced by the wild card asterisk. The widget class can be omitted if the given resource name should be dumped for all widget classes; for example:

```
*sensitive.vrSave: True
```

RECORD AND PLAY SESSIONS

Controlling Snapshot Scope and Granularity

To dump all resources for a Motif push button widget class, the following specification could be used:

```
*XmPushButton.?.vrSave: True
```

For an uncommon situation that requires saving all resources for all widgets, use:

```
*vrSave: True
```

Such a specification should be used sparingly since it will generate large snapshot files. They can be very useful, however, in debugging applications and as a way to quickly get information on the complete widget tree.

Widget Instance-Based Specification

A widget class-based specification is easiest to use because it allows one to quickly control the dump granularity for most widgets in the application. However, there might be cases where a particular widget instance needs to have a dump granularity that is different from the widget class; for example, if you want all text label contents dumped but not the contents of the label that contains the time field because it changes continuously and therefore cannot be used for verification purposes.

A widget instance specification takes the following form:

```
application_class.instance_name.resource_name.vrI  
nstanceSave: [True | False]
```

Example:

```
*topshell.form.timeLabel.labelString.vrInstanceSa  
ve: False
```

The instance name can be a widget name or a widget tag.

The asterisk and question mark wild card notations can be used to match more than one widget instance; for example:

```
appname*sensitive.vrInstanceSave: True  
appname.topshell.?.sensitive.vrInstanceSave: True
```

The first entry dumps the sensitive resource for all descendants of **appname** while the second entry dumps the same resource for all children of **topshell**. The instance-based specification always takes precedence over the widget class specification.

Another advantage of the ? notation, is its usefulness for dumping all resources for a composite widget.

Example:

```
*topshell.?.vrInstanceSave: True
```

Resource Type-Based Specification

The *resource_name* field in the widget class- and instance-based specification can also be replaced by a resource type. Such a specification is useful when all resources of certain types should or should not be dumped, regardless of the widget class or instance in which they occur. Resource type specifications have a lower precedence than specifications based on resource name.

Examples:

```
*XmString.vrSave: True
*XmText.Boolean.vrSave: True
*myTextEdit.Boolean.vrInstanceSave: True
```

The first entry dumps all resources of the type **XmString**; the second entry dumps all boolean resources (**XtRBoolean**) that occur in **XmText** widgets; the third entry dumps all boolean resources in widget instance **myTextEdit**.

Replay Xcessory Pseudo Resources

In addition to dumping the resources defined by the widget classes, Replay Xcessory also defines a number of additional pseudo resources that provide additional information about the widget. The dumping of these pseudo resources can be controlled in the same way as the regular widget resources by using the **.vrSave** or **.vrInstanceSave** specification.

The value of pseudo resources cannot be retrieved using the **getvalue** command; use the **getclass**, **ismanaged**, **getchildren**, and **getpops** commands instead. The pseudo resources are:

wcClassName	Identifies the widget class.
wcManaged	Identifies whether or not the widget is managed.
wcChildren	Identifies the names of the widget's regular children.
wcPops	Identifies the names of the widget's pop up children.
vrImageSaveFile	Identifies the name of the image save file; this provides a way to correlate an image file in the snapshot directory with a particular widget instance.

**Widget Snapshot
File Format**

The pseudo resources with the **Wc** prefix have a resource type of **WcResources**, while **vrImageSaveFile** has a resource type of **VrResources**.

The widget snapshot is an ASCII file containing one entry for each widget resource dumped. All resources regardless of class are converted to an ASCII string. The conversion is performed via the X Toolkit resource type converters. For a resource to be correctly dumped, there must be a converter registered that converts the desired resource type to a string.

Replay Xcessory provides resource converters for most Motif resource types. If a resource converter is needed but one does not exist, Replay Xcessory still generates an entry in the widget snapshot file as an entry comment (a line that starts with an exclamation mark). Instead of the resource value, the resource type name is printed in parentheses. (Refer to Chapter 9 for instructions on how to write and register additional converters.)

Please note that at the beginning of each new record/playback session, the appropriate baseline/result snapshots will be removed.

Image Snapshots

Image snapshots in Replay Xcessory should generally be avoided if a logical snapshot can be taken. However, image snapshots are useful when you need to verify the visual representation of a portion of the application and there is no other convenient way to do it. This might be the case in some highly graphical applications consisting primarily of drawn areas and images.

Image Snapshot Scope

To obtain an image snapshot of any widget, including container widgets such as forms, do the following:

1. Set the snapshot scope to **Image** in the Snapshot Scope option menu and click on the **w** button of the Record Control Panel. The cursor changes to the shape of a camera. The Replay Xcessory main window will be iconified to free the visible desktop area and allow the user to select the appropriate widget.
2. Position the cursor so that the arrow points to the desired widget and click the left mouse button. To take an image snapshot of the complete window, press the Shift key while clicking on the desired window.
3. After clicking on the target widget, Replay Xcessory's main window will be restored to its previous state.

Replay Xcessory takes an image snapshot of the selected widget and widgets contained within it.

Alternately, use the snapshot hot key combination.

Image snapshots created in this manner will not generate a logical snapshot. Therefore, if you want to create a corresponding logical snapshot, you should take an object snapshot of the same widget.

Note: It is still possible to create a logical snapshot that includes image snapshots via the `.Vrdump` or `loadspec/mergespec` dump specifications.

Sub-Image Snapshot Scope

Sub-image mode allows you to construct complex masks indicating areas to be compared with the original image and masked areas that are skipped.

Here is the description of controls on the Snapshot control panel:

W	Select object region as with the Image snapshot scope.
XY	Select free rectangle region with arbitrary dimensions.
Add	Set unmasked region type. The sub-image selected with the region of this type is compared with the original one.
Mask	Set masked region type. The sub-image selected with the region of this type is not compared with the original one.
Scope	Select the snapshot scope (object, window, viewable, full, image, sub-image). Refer to “ <i>Controlling Snapshot Scope and Granularity</i> ” on page 122 for additional information.
Tolerance	Option for xwdiff program that makes image comparison. See man 1 xwdiff.
N pixels	Option for xwdiff program that makes image comparisons. See man 1 xwdiff.

Accept Create snapshot and save mask parameters in the snapshot properties file.

Deny Cancel the created mask.

Note: Coordinates in sub-image mode are always recorded relative to the window (shell) in which they appear, even if they are wholly contained within a specific widget. Therefore, they are inherently less portable, i.e. more susceptible to changes in font and other resources, than all other forms of snapshots, including image snapshots.

For additional information about image snapshots, see “*Controlling Snapshot Scope and Granularity*” on page 122.

Record Control Panel

The Record Control Panel provides the controls to conduct record sessions started by the Test Manager or with the command-line interface. When using the Replay Xcessory Test Manager, you can select an icon that represents the script and baseline snapshot, or just the test case definition file, before selecting the **Record** option. The names of the icons selected appear in the Record Control Panel. If a script, baseline icon, or test case icon is not selected, the user will have to enter at least the test name. All other fields will automatically fill in with the test name and the necessary suffix.

Here is an illustration of the Record Control Panel:

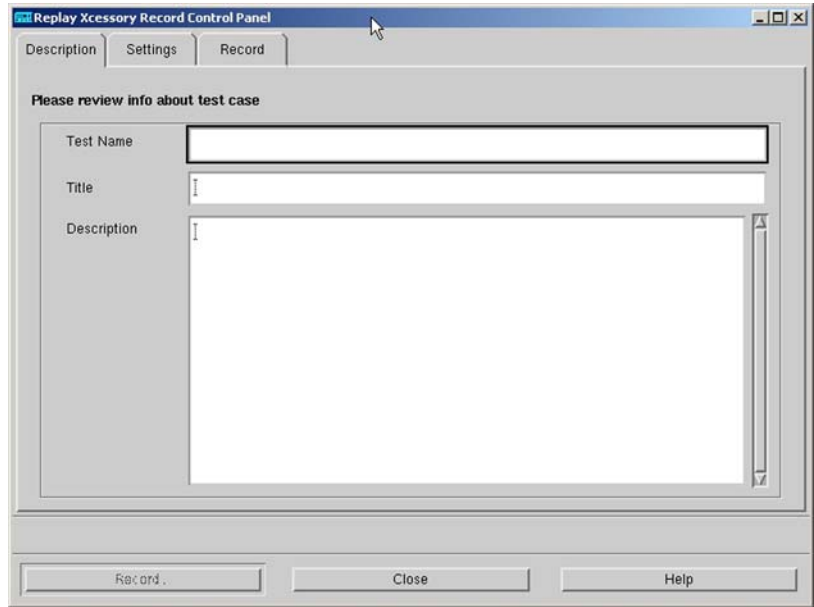
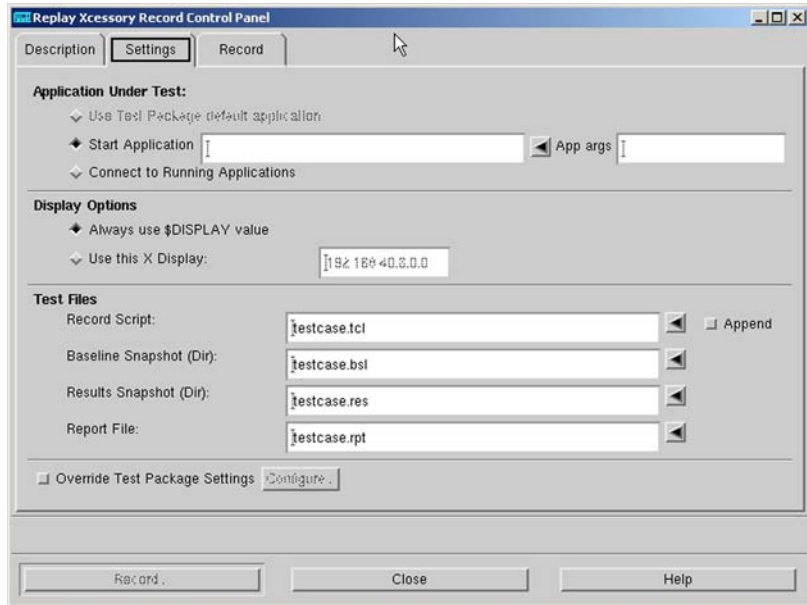


Figure 62 Record Control Panel, Description Tab

**Figure 63** Record Control Panel, Settings Tab

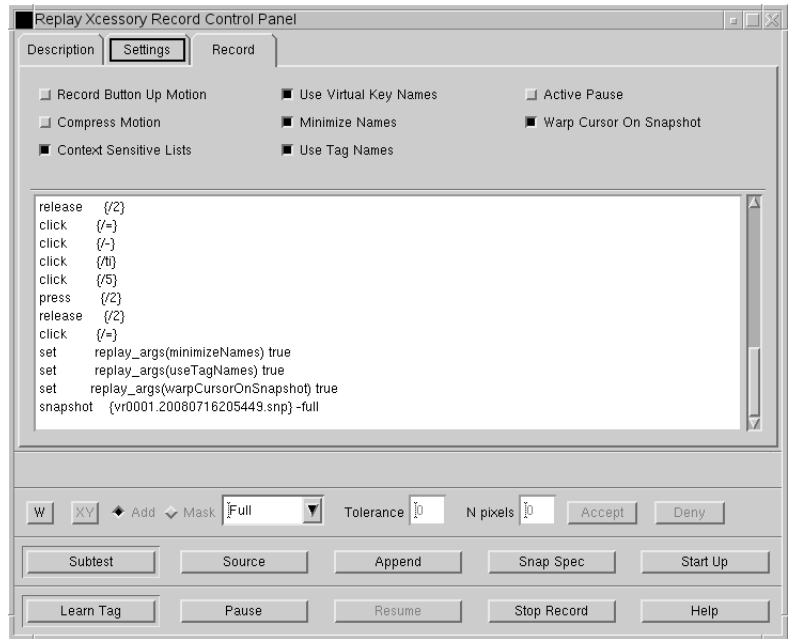


Figure 64 Record Control Panel, Record Tab

Note: A relative path for a file name is relative to the path of the open test package.

Record Controls

Here are detailed descriptions of the controls of the record control panel, divided up by the three tabs:

Description Page

Test name	A short test name which identifies one test name from another.
Title	The main idea about what the current test is for.
Description	Any text which might describe the details of the test and its purpose.

Settings Page

Application Under Test If the **Start Application** toggle button is selected, enter the command line for the application. If the application's location file name is in your **PATH** environment variable, you can omit the full path specification here.

If the **Connect to Running Applications** toggle button is selected, Replay Xcessory connects to all existing applications that are linked with the Replay Xcessory X Toolkit library and uses the properly configured environment (for shared linked AUTs) on the current display, or on the application display if one is specified. This allows recording of applications without launching them from Replay Xcessory; on playback, the same applications must be present and in the same state as when recorded.

For an explanation of excluding applications from record, refer to "*Excluding Applications*" on page 101.

Apps args The user can supply any command line arguments, which will be transferred to the application under test on startup. Note that you can use environment variables here. Replay will substitute them with their actual values at the startup stage.

- Display Options** Name of a display (for example **nodename:0**) in which to display the applications under test. The name of the last used display which was stored in the test case definition file is shown by default. All applications launched through Replay Xcessory will have their DISPLAY variable set to the application's display.
- This feature is useful when the applications under test occupy most or all of the available screen space, or to avoid side effects caused by interaction between Replay Xcessory and the applications under test. The remote display can also be a nested server (Xnest) in the same physical display.
- Record Script** File name with a **.tcl** extension for this session's script file. Note that you can use environment variables to set this field.
- Append** If the specified file already exists, this option appends the script to it rather than over-writing it. To use this option, ensure that the application state at the end of the existing script is consistent with the state from which the new record session will start. Only in this case will a play session be able to play the merged script seamlessly.
- Baseline Snapshot (Dir)** Directory name with a **.bsl** extension for the directory that will contain the snapshots for this session. In addition, certain control files in the test package (**.Replay**, **.Vrdump**, **~/Xdefaults**) are copied into the baseline as a historical record. Note that you can use environment variables to set this field.
- Results Snapshot (Dir)** Directory name with a **.res** extension for the directory that will contain the resulting snapshots which will generate on playback. Note that you can use environment variables to set this field.
- Report File** A file with an extension **.rpt**, which will hold the status of the playback sessions. Note that you can use environment variables to set this field.

Override Test Package Settings When the user switches this toggle on and presses the “**Configure**” button, the session properties window appears and the user is able to set the desired options which will override the test package options and system-wide options, but only for the current test case.

Record Page

Record Button Up Motion Requests the recording of mouse movements with mouse buttons up. Mouse movements when the mouse buttons are up are irrelevant for most processes. However, they are important with certain modal interfaces; for example, in testing drawing tools that draw with the mouse button in the up position (system default: off).

Compress Motion Shortens the recorded script by recording the coordinates of the end point only. Use **Compress Motion** to make your script files smaller if the intermediate points are irrelevant (system default: off).

Context Sensitive Lists Specifies that the Motif list item acted on be recorded using the string listed in the item itself. This allows the same item to be selected on playback even if the font or item position changes. This option should be turned off if the horizontal position in the selected item is important as is the case with many list items used in hypertext applications (system default: off).

Use Virtual Key Names Specifies that key symbols (**X Keysyms**) be recorded using the OSF/Motif key symbol names for maximum script portability across displays that use different key bindings (for example, backspacing makes use of the **Backspace** key on some displays, but uses the **Delete** key on others).

Recording using the virtual key name (**osfBackSpace**) ensures that the script is portable to both display types. This option is valid only for Motif applications.

Minimize Names Requests minimized widget names in scripts and widget snapshots; otherwise, fully-qualified names are used. See Chapter 2 for further information on widget names (system default: on).

Use Tag Names Records the widget tag rather than the internal widget name. See Chapter 2 for further information on widget names (system default: on).

Active Pause This option allows a process to be acted on during pause mode.

Note: Use this option with caution, as the nonrecorded interaction could invalidate a play action if the application state is modified during the pause.

Warp Cursor on Snapshot Replay Xcessory normally moves the cursor out of the way before taking a snapshot. This precaution generally results in more reliable snapshot comparisons (fewer false negatives). Turn this option off only if the cursor warping generates an undesirable side effect.

Note: The hot keys mentioned in the following enable you to control the record session without having to keep the Record Control panel visible at all times. This is important when the application under test occupies most of the screen.

Before **Record** is pressed, these buttons are located along the lower edge of the panel:

- | | |
|---------------|--|
| Record | Starts the actual recording process. |
| Close | Dismisses the record control panel if the Record button has not yet been pressed. |
| Help | Brings up an explanation of these record options. |

After **Record** is pressed, the Record Control Panel switches to the third page and disables the first and second pages. New buttons along the lower border appear, as shown:

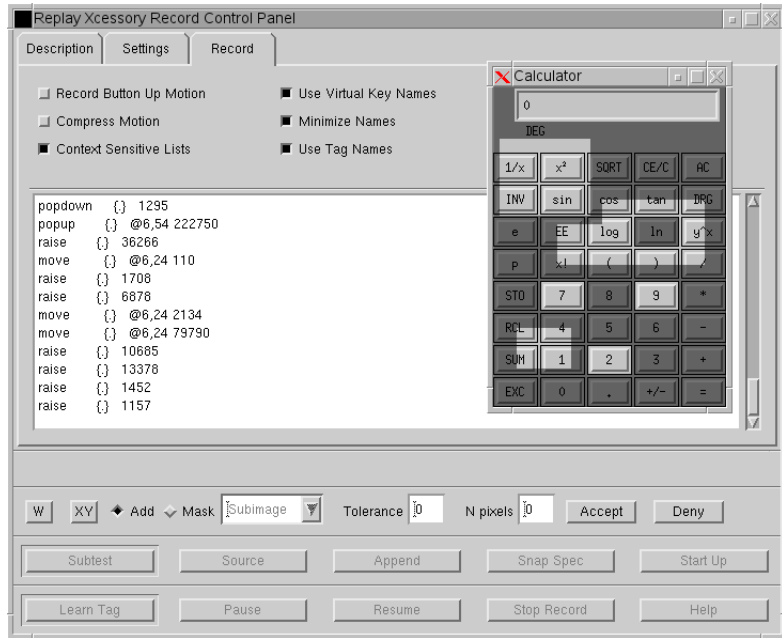


Figure 65 Record Control Panel While Recording

Here is a description of the view area:

Script View Area Scrollable area in which Replay Xcessory commands display as the script is created; these commands are saved in the script file.

After **Record** is pressed, these buttons appear along the lower edge of the panel:

Subtest Indicates the start of a new subtest. By default, subtests are numbered sequentially, starting from 1. If the subtest autonumbering is turned off, you are prompted for the subtest name.

w

Takes an object, window, viewable, or full snapshot, depending on the current state of the **Snapshot Scope** option menu. The default hot key for **w** is **Ctrl+i**.

A *full* snapshot potentially includes all widgets in the application depending on the current snapshot specification. If more than one process is being tested, the snapshot is only taken in the current application process. A bell sounds when the snapshot is complete.

A *viewable* snapshot is similar to a full snapshot, except that it only includes the currently visible widgets in the application.

When taking an *object* or *window* snapshot, the cursor changes to the shape of a camera. Position the cursor so the arrow points to the desired widget and click the left mouse button; the snapshot will contain the selected widget and the widgets contained within it, but does not include any pop up widgets or menus.

A *window* snapshot is similar to an object snapshot, except that the object is always an entire shell window.

To obtain a snapshot of a widget contained within a menu, leave the mouse in the popped-up state and press the snapshot hot key combination.

Source

Brings up a dialog box that prompts you for the name of a Tcl script to be sourced in. If you enter the file name and press **OK**, the script will be played before further recording occurs. There are also options to press **Cancel** or **Help**.

The **Source** button can be pushed at any time during the record session. This causes the specified file to be played and a source command to be entered in the record script. Possible uses include bringing the application to the desired state and invoking common cleanup procedures.

Append	Brings up a dialog box to insert a comment or command in the script. Control flow and other commands to be executed during playback may be inserted using this button, but they are not executed during the record session.
Snap Spec	Brings up a dialog box for entering new snapshot dump specifications.
Start Up	Brings up a dialog box that allows you to type in a new command line in order to begin a new application to test.
Learn Tag	Brings up a dialog box for learning widget tag names. The default hot key combination for Learn Tag is Ctrl+t .
Pause	Temporarily interrupts the record session; press Resume to continue with the session. Pause prevents unnecessarily long delays from being recorded when you must interrupt the record session to do something else. The default hot key combination Ctrl+p toggles Pause and Resume .
Resume	Causes the record session to continue, after a temporary interruption initiated by Pause . The default hot key combination Ctrl+p toggles Pause and Resume .
Stop Record	Ends the record session. The default hot key for Stop is Ctrl+s .
Help	Brings up an explanation of these record options.

Changing Snapshot Granularity During Record

The granularity of a snapshot is controlled by the dump specification file **.Vrdump**. In cases where it is necessary to modify the granularity of the snapshots during the record session, you can do so by activating the **Snapshot Spec button** in the **Record Control Panel**. Activating the **Snapshot Spec** causes a Snapshot Specification Entry form to pop up, as shown:

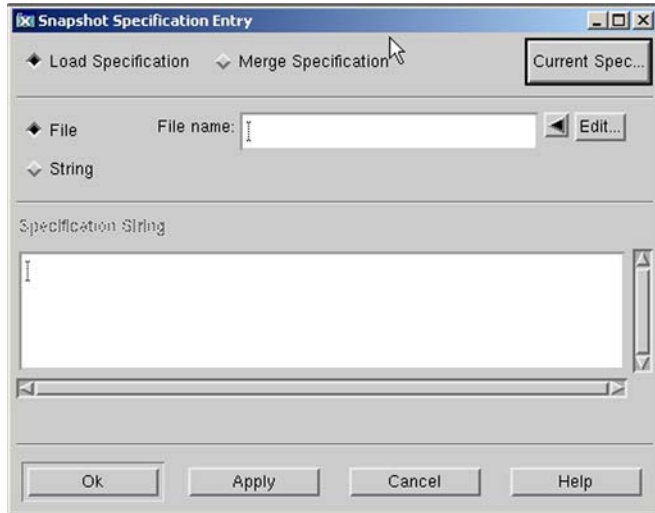


Figure 66 Snapshot Specification Entry

This dialog box loads or merges a new snapshot specification to change the snapshot granularity during the record session. Snapshot specifications may be *merged* or *loaded* (see the **xrdb(1)** reference manual page).

- loaded specifications replace previous specifications
- merged specifications indicate that the specification be merged with, instead of replacing, the current contents of the previous specification

An appropriate **loadspec** or **mergespec** command is added to the record script when **OK** is pressed. Snapshot specifications may be specified with a file or with a string.

The following table describes the buttons on the dialog box.

Load Specification	Snapshot specifications will be loaded.
Merge Specification	Snapshot specifications will be merged.
File	Snapshot specification is in a file.

String	Snapshot specification is in a string. The string is entered in the Specification String text window.
Edit...	Bring up an editor to edit a file snapshot specification. The Replay.editorCommand resource in the .Replay file can be used to modify the editor command.
Current Spec...	Show the current snapshot specification.
Apply	Load or merge a new snapshot specification; in addition, write out an appropriate loadspec or mergespec command to the record script.
Cancel	Cancel the session and close the dialog box.

Recording Widget Tags

Replay Xcessory records the names of the widgets using the widget *names* or *tags* (see “*Widget Names and Tags*” on page 36 for a description of the differences between widget names and tags). Because widget names are assigned by the developer of the application or, in some cases, by the GUI builders, the assigned widget names may not be particularly unique or meaningful. Widget tags are arbitrary strings associated with a widget and are typically identically assigned to visible strings that are associated with the widgets.

By default, widget tags are recorded if a name string is available. Replay Xcessory automatically generates a widget tag based on labels or titles associated with that particular widget; for example, push button labels or shell titles. Widget tags can be manually assigned to other widgets by using the **Learn Tag** feature which is available in record mode. The resulting tag associations are recorded in a tag file (**.Vrtag**), which can then be used during future record or play sessions.

Typically you should generate a complete tag file in a practice record session before proceeding with the actual record sessions. The script that is generated in the practice session can be discarded, and the tag file that has been generated can be copied to a central location, for example, the properties directory for the test suite.

Conducting a Learn Tag Session

Use the following steps to conduct a Learn Tag session from the Test Manager:

1. Select **Record...** from the **Record/Play** menu in the Test Package window.
2. Place the cursor in the appropriate fields in the Test Package window and specify the application to be tested. For this example, type the name of the application as **tester**.
3. Press the **Record** button to begin the record process.
4. The control panel changes and the application under test appears. The buttons along the bottom of the control panel change; included in this group is the **Learn Tag** button.
5. Use the application the way you normally would. The script echoed in the script view area reflects the names (either widget names or tags) being recorded.
6. To switch to **Learn Tag** mode, press the **Learn Tag** button.

Learning Tag Names

The **Learn Tag** button brings up the Learn Widget Tag Name window, which makes it possible to assign names to GUI objects or change default tags to unique reference names. Here is an illustration:

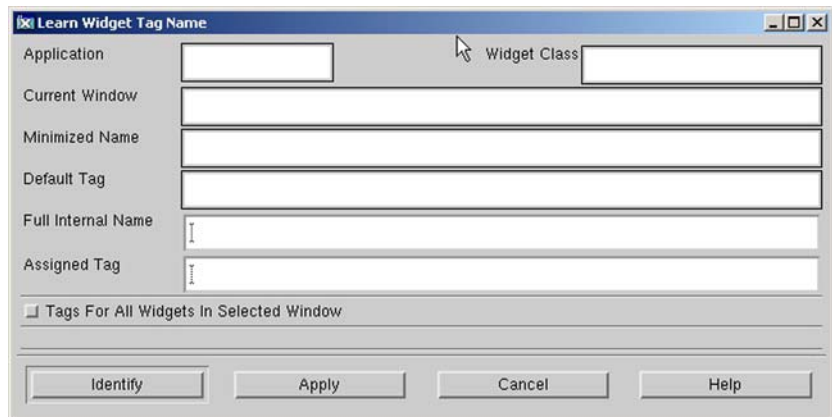


Figure 67 Learn Widget Tag Name

1. Click on the **Identify** button. The cursor changes to a question mark and you are requested to “Move pointer over widget of interest.”

RECORD AND PLAY SESSIONS

Recording Widget Tags

- As you move the cursor on the application process, widgets flash to let you know what is being selected. Notice that the fields in the **Learn Widget Tag Name** window change as the cursor is moved; for this example, the **Help** button is retagged.
- Click the left mouse button when the pointer is on the widget that will be assigned the tag.
- Click the right mouse button to exit from the **Identify** mode.
- Give the widget its own unique tag by typing the new tag name in the **Assigned Tag** field and press the **Apply** button; for this example, we are renaming the **Help** button to “Main Help.”

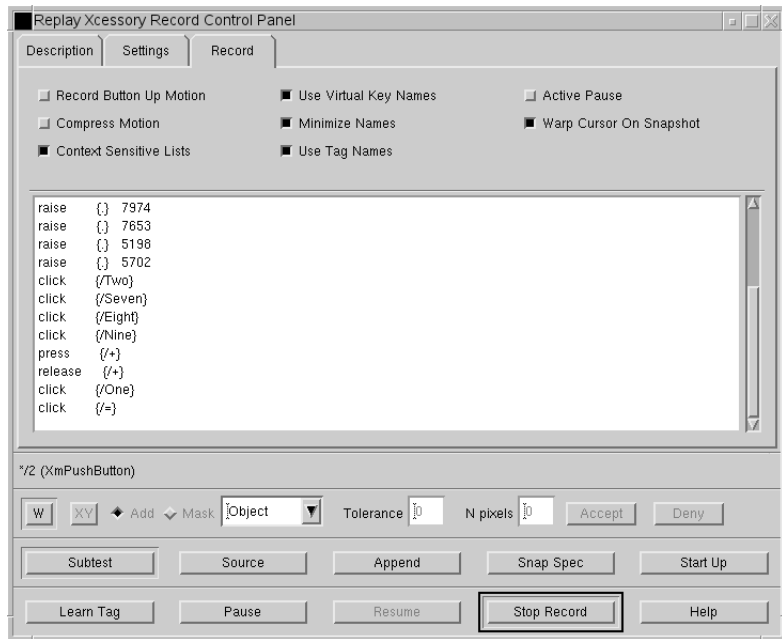


Figure 68 Displaying a Tagged Widget During a Record Session

Note: A quick way to assign tags when the desired tag is a visible label on a Motif 1.2 or later application is to drag the label into the **Assigned Tag** field. To use this feature, the **Active Pause** button should be selected in the Record Control panel before starting the **Learn Tag** session.

Deleting the Tag Name

To delete the new tag name:

1. Pop up the Learn Widget Tag Name window as previously described, select the widget, and place the cursor in the Assigned Tag field.
2. Erase the tag field, and click **Apply** to confirm the deletion. A dialog box pops up to ensure that this is the action you want to take:

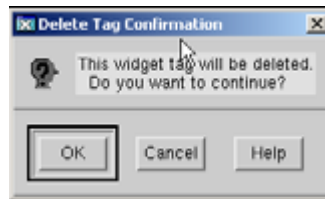


Figure 69 Delete Tag Dialog Box

3. Click **OK** to remove the new tag name.

Tag File Generation

Tag files associate arbitrary widget tags with internal widget names. This level of indirection allows all widgets to be named by any arbitrary name rather than the internal widget names created by the developers. Generation of complete tag files for an application is especially useful for internationalized applications because the default widget tags, if used, would be different for each locale.

The Learn Widget Tag Name form now provides the option of creating tag entries for all widgets in a selected window, including popup children. Tag entries generated in this manner contain the default names, if possible. Otherwise the tags are the same as the internal name of the widget.

If there are multiple occurrences of the same name, the widget tags have a “_#” suffix, where # is a number, appended to the tag name.

You can typically use mass generation of tag entries to accelerate the creation of tag files corresponding to a complete application. You can then edit the generated file to assign more meaningful names than those which were automatically generated.

To generate tags for a window, do the following:

1. Select the **Generate Tags for all Widgets in Selected Window** toggle from the Learn Widget Tag Name form.

2. Click the **Identify** button, move the pointer over the desired window, and click the left mouse button.
3. Click the **Apply** button. The tag file is not actually written until the application exits. The tag file is named *appname.Vrtag*.

To create a comprehensive tag file for the complete application, repeat these steps for each top level shell window in the application.

Note that after vrtag generation is complete, a new item appears in the test package window under the “Options” menu named “Edit vrtags”. It is in the corresponding pulldown menu list of all the vrtag files located in the current package. Each vrtag file generated in a particular test case will be copied to the test packed directory, which is usually parent to the test case, and will be used in all other test cases during the record or playback stages.

A partial sample generated tag file is shown below:

```
tester.rc.label2.vrTag: single line text_label
tester.rc.label1.vrTag: multi line text_label
tester.rc.menubar.button_0.vrTag: File
tester.rc.menubar.button_1.vrTag: Help
tester.rc.menubar.vrTag: menubar
tester.rc.label3.vrTag: scale_label
tester.rc.Gadget.vrTag: Gadget
tester.rc.scale.Title.vrTag: Title_label
tester.rc.scale.Scrollbar.vrTag: Scrollbar
tester.rc.scale.vrTag: scale
tester.rc.Widget.vrTag: Widget
tester.rc.Quit.vrTag: Quit
tester.rc.text[2].vrTag: text__2
tester.rc.draw.vrTag: draw
tester.rc.ListSW.VertScrollBar.vrTag:
    VertScrollBar
tester.rc.ListSW.List.vrTag: List
tester.rc.ListSW.vrTag: ListSW
tester.rc.FileSelection[2].vrTag:
    FileSelection__2
tester.rc.text.vrTag: text
tester.rc.FileSelection.vrTag: FileSelection
tester.rc.vrTag: rc
tester.vrTag: tester
```

For additional information about widget tags, see “Recording Widget Tags” on page 142.

Please note that sometimes during application creation, programmers do not specify names or specify empty names for implicit widgets that are not shown by the user or that the programmer is not planning to use later. This has the potential to cause the Xt library, and hence Replay Xcessory, to misbehave while interpreting X resource files (Vrtag, Vrdump) files. Several possible solutions to this problem are listed below.

Empty or space-filled name:

```
level1.level2..widget1.vrTag : widget1
```

or

```
level1.level2.      .widget1.vrTag : widget1
```

These examples indicate that there is an additional widget between level2 and widget1, but that it will be invisible to the Xt library and Replay Xcessory.

A less obvious and harder to figure out situation is when the user manually edits the resource (tag) file and enters spaces at the beginning or end of a widget/tag name.

```
level1.level2.widget1 .vrTag : widget1
```

It looks like widget1 has a space as part of its name and probably as part of the tag name, but in this case, this will not be seen by Xt. The proper solution is to enclose the spaces by back-slashes, such as:

```
level1.level2.widget1\ .vrTag : widget1\
```

However, it is strongly recommended that one does not use spaces at all.

Another example of an incorrect widget name is using a colon ":" inside of a widget name. In certain cases, this will prevent Replay from properly understanding the information contained in the X resource files (vrTag files for example) and should be avoided during programming.

Streamlined Recorded Scripts

The following resources are designed to help you generate simpler scripts with less superfluous information.

Omit Delay

Explicit delays are never needed for proper playback of recorded scripts. Replay Xcessory does not record delays if the **omitDelay** resource is set to **True** in the **.Replay** file. During a play session, the default delay between script actions will be controlled by the **defaultDelayTime** resource (default is one second).

Omit Coordinates by Widget Class

Widget coordinates are superfluous for most commonly used widget classes because the effect of a button press or click is identical regardless of where the click took place. Replay Xcessory now allows you to specify the widget classes where it is safe to ignore the coordinates. The result is a simpler script. If a widget coordinate is omitted, the delay for that transaction will be omitted too.

The new default **.Replay** file omits recording of most Motif widgets where coordinates are immaterial, such as push buttons, toggle buttons, and labels. The complete list of widget classes can be found in the following directory:

```
$REPLAYHOME/lib/app-defaults/Replay
```

The name of the resource is **omitCoordinatesWidgetClasses**.

Note, however, that this variable could prevent some test cases from being recorded properly. On applications that use custom drawn widgets, this option should not include the name of the widget class that is used as a canvas for the widget or dynamically created widget as part of the custom widget. The default value is:

```
XmRowColumn \
XmPushButton XmPushButtonGadget \
XmToggleButton XmToggleButtonGadget \
XmCascadeButton XmCascadeButtonGadget \
XmSeparator XmSeparatorGadget \
XmLabel XmLabelGadget \
XmArrowButton XmArrowButtonGadget \
XmDrawnButton \
XmForm \
Toggle Command
```

Omit Button Up Motion by Widget Class

Replay Xcessory has always provided the option of recording or ignoring pointer drags while all mouse buttons are in the up position via the **includeButtonupMotion** resource.

A new resource, **omitButtonUpMotionWidgetClasses**, gives you greater control over the granularity by allowing you to specify the widget classes where it is safe to ignore pointer motion when all mouse buttons are up. This new option is in effect only if the **includeButtonupMotion** resource is set to **True**.

The new default **.Replay** file omits recording of button up motion for most Motif widgets where coordinates are immaterial, such as push buttons, list widget, and toggle buttons. The complete list of widget classes can be found in the following directory:

```
$REPLAYHOME/lib/app-defaults/Replay.
```

Play Control Panel

The Play Control Panel provides the controls to conduct play sessions which can be started either from the Test Manager or the command line.

When using the Replay Xcessory Test Manager, you can select the icons that represent the script and baseline, and the result snapshot and report files before selecting the **Play** option. The name of the selected icons appear in the Play Control Panel. If none of the icons are selected, the script, snapshot, and report fields are filled with the values of the last record or play session used in the current test package.

The Play Control Panel consists of two pages.

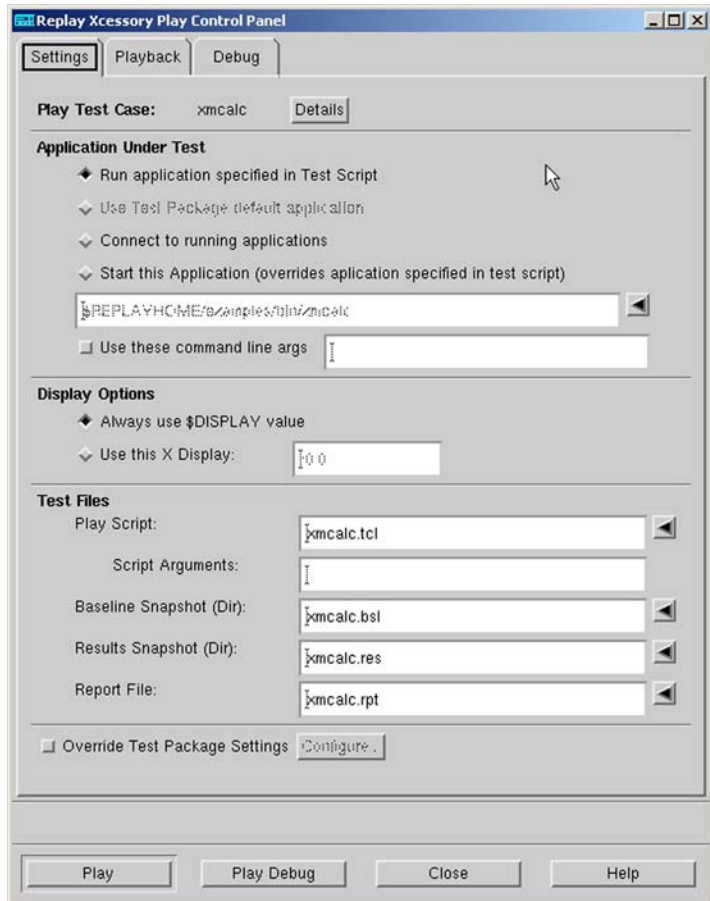


Figure 70 Play Control Panel, Settings Tab

Note: If you enter a relative path for a file name, it is relative to the path of the open test package.

Conducting a Play Session

Use the following steps to conduct a play session from the Test Manager:

1. Select **Play...** from the **Record/Play** menu in the test package window. The Play Control Panel appears.
2. Edit the text fields to fit your situation and make sure the check buttons are set as you wish. (The play controls are described in detail following these steps.)
3. To make this session execute as quickly as possible, rather than at record-session speed, drag the Speed slider all the way to the right (1.0).
4. When ready, press the **Play** button to begin the actual play process. A window appears with the test application and the buttons along the lower border of the Play Control Panel change to include **Pause**, **Resume**, **Cancel**, and **Help**.
If both baseline and result snapshot directories are specified, Replay Xcessory will perform snapshot comparisons at the same points in the script where the baseline snapshots were initially taken. If there are differences, you'll have the option to continue playing, accept the new results as the new baseline, or stop playing.
5. To stop the process temporarily, press **Pause** and then **Resume** to continue.
6. To terminate a session early, press **Stop**.

Once started, a play session proceeds automatically. It provides a subtest summary report if requested.

Play Controls

Here are detailed descriptions of the controls of the play control panel:

Settings Page

Play Test Case	The name of the current test case is displayed here. If you want to view a detailed description of the test case you can press the "Details" button.
-----------------------	--

**Application
Under Test**

Command line of the application under test; it can be omitted if the recorded command line in the start up command is acceptable. If it is omitted, just leave the “Run Application specified in the test script” active, or you can select “Connect to running applications”, to connect to all applications running at this moment that are properly linked against Instrumented Xt library applications.

**Display
Options**

Name of a display (for example **nodename:0**) in which to display the applications under test. This feature is useful when the applications under test occupy most or all of the available screen space, or to avoid side effects caused by the interaction between Replay Xcessory and the applications under test. The remote display can also be a nested server in the same physical display.

Besides using the DISPLAY value and the current application display, the user can also specify the display options using Xnest (nested X server) and Xvfb (virtual X server). It is assumed that these tools are installed on the system in the executive path. Replay Xcessory will not perform a search for the needed tools.

On selection, the nested X server will run as an additional local X server with display number “5” and the virtual X server will be run as display number “6”.

In the case that the batch mode was specified as nested, the virtual X server will close automatically, regardless of the results of the test case playback. In the case that the batch mode was not specified, the virtual servers will be alive and the application tester can check the results of the playback.

If using Xvfb was selected then no output of the application under test will be shown at all! The application tester still has the ability to monitor the virtual display using the “Shot” button on the second page of the playback dialog. This button will be accessible only if the Xvfb option is selected.

Default options of virtual displays are:

Width x Height: 1024x768

Color Depth: 24

Window manager: mwm (should be installed on system).

These options can be configured on the “General” page of the session properties for test cases and test packages.

Please note that the selected color depth should also be present in the X configuration file settings and available for active display. Otherwise, messages such as “Could not find available visual” will be generated by Replay.

Baseline Snapshot (Dir)	Name of the directory that contains baseline snapshots for this session. If a baseline snapshot directory is not provided, no baseline comparison will be performed.
Results Snapshot (Dir)	Directory name with a .res extension for the directory to contain the snapshot results for this session.
Report File	File name with a .rpt extension for this session's report file. This file summarizes the success or failure of snapshot comparisons, subtest by subtest. If a file name is omitted, the report is displayed to the standard output (stdout).
Play Script	File name with a .tcl extension for this session's script file.
Tcl Arguments	Arguments to be passed to a Tcl script. The arguments can be restored within the script using the built-in variable argv (a Tcl list). The number of arguments can be retrieved using the argc variable.
Override Test Package Settings	When the user switches this toggle on and presses the "Configure" button, the session properties window appears and the user is able to set the desired playback options. This will override test package options system-wide, but only for the current test case.
Playback Page	
Speed	Specifies the play speed when the slider is dragged. The speed is represented by a floating point number between -1 and 1 where -1 is slowest, 1 is fastest, and 0 is the speed of the record session. The slowest speed that can be set is roughly half as fast as the recorded speed. The fastest setting is the fastest possible speed, limited only by the minimum delay needed to guarantee correct play behavior; for example, to prevent two consecutive single clicks from being interpreted as a double-click.
Prompt on Snapshot Difference	Specifies that you are to be prompted whether to exit or continue, when a snapshot file fails to match the corresponding baseline file.

RECORD AND PLAY SESSIONS*Play Control Panel*

Before **Play** is pressed, these buttons are located along the lower border of the panel:

- | | |
|-----------------------|---|
| Play | Starts the actual play process. |
| Play
Debug | Starts the debug playback session. You will be able to debug your tcl code, changing it at runtime. |
| Close | Dismisses the play control panel before the session can begin. |
| Help | Brings up an explanation of these play options. |

After **Play** is pressed, new buttons appear along the lower border of the panel, as shown:

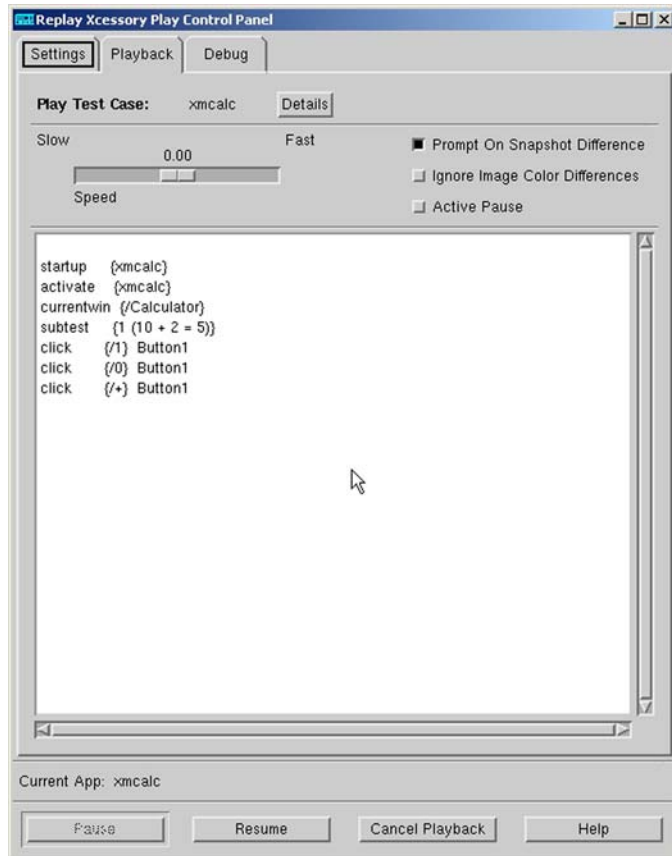


Figure 71 Play Control Panel—While Playing

After **Play** is pressed, the following buttons appear along the lower border of the panel:

- Pause** Temporarily interrupts the play session; press **Resume** to continue with the session.
- Resume** Causes the play session to continue, after a temporary interruption initiated by **Pause**.
- Close** Ends the play session.

Help Brings up an explanation of these play options.

If you chose **Play Debug**, the tcl debugger will be opened. For more information, please see Chapter 8 Script Debugger.

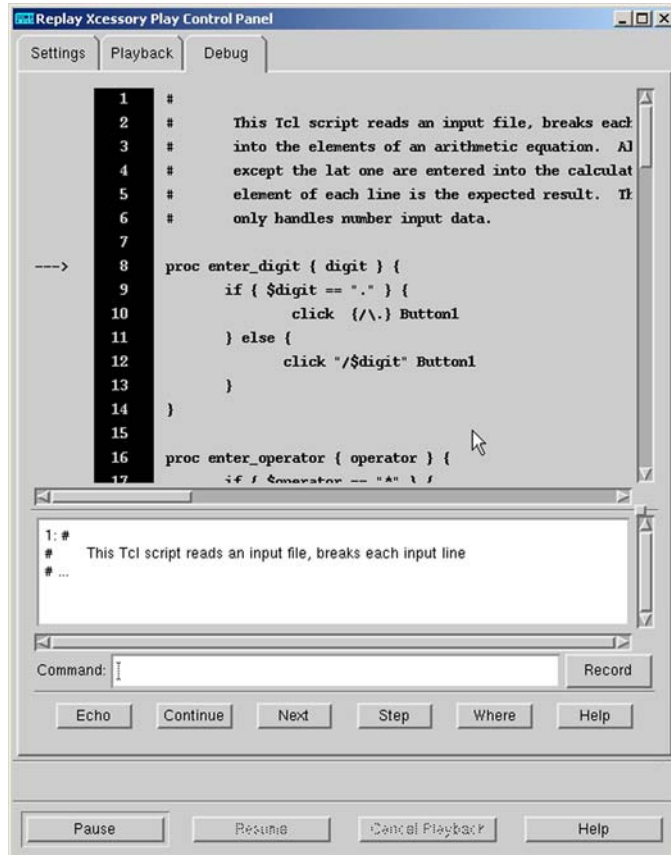


Figure 72 Play Control Panel- Debug Tab

Verifying Results

Results verification through snapshot comparison occurs automatically during playback when both the baseline and results snapshot directories have been specified. Comparisons are performed on the widget snapshots and image snapshots.

Whenever a difference is found, you can:

- ignore the difference and continue playing
- accept the new results as the new baseline
- stop playing

Verifying widget Snapshot Differences

The Snapshot Mismatch window, as shown in the following illustration, displays the command line used to compare the baseline and results widget snapshots, as well as the output of the comparison in a scrollable display area.

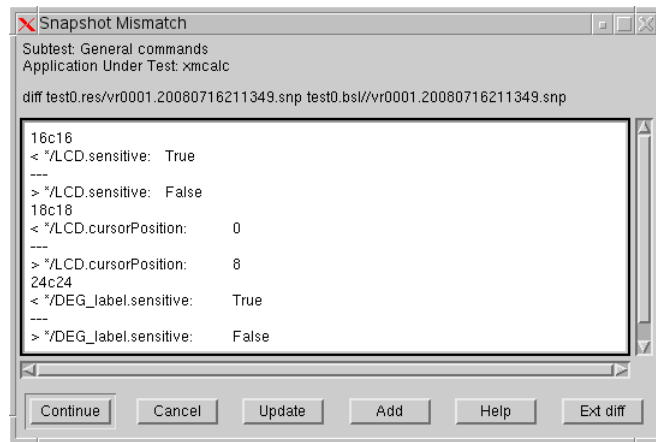


Figure 73 Snapshot Mismatch Window

The following buttons control widget snapshot differences:

- Continue** Ignore the widget snapshot differences and continue playing.
- Cancel** Stop the play session.

- Update** Update the baseline snapshot with the new results and continue the play session.
- Help** Brings up an explanation on these options.
- Ext diff** View snapshot differences in the external viewer.

Verifying Image Snapshot Differences

The Image Mismatch window, as shown in the following illustration, displays the command line used to compare the baseline and results widget snapshots.

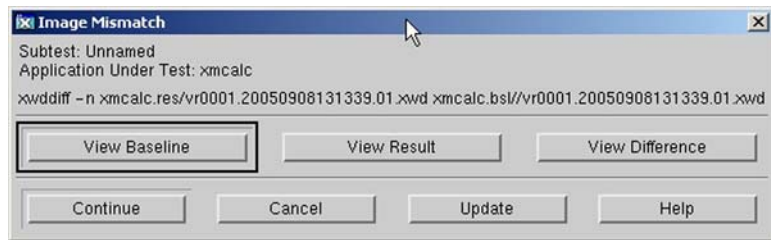


Figure 74 Image Mismatch Window

The buttons in the top row allow you to view the baseline, result, and difference images; a viewed image can be dismissed by clicking on it.

The following buttons control image snapshot differences:

- View Baseline** Brings up image viewer on the baseline image.
- View Result** Brings up image viewer on the result image.
- View Difference** Brings up image viewer on an image where the pixels which differ are shown in black and those which are the same one shown in white.
- Continue** Ignores the widget snapshot differences and continues playing.
- Cancel** Stops the play session.
- Update** Updates the baseline snapshot with the new results and continues the play session.

Help Brings up an explanation of these options.

Replay Xcessory Driver — Command-Line Interface

The driver command-line interface can start both record and play sessions in batch mode or interactively. This command is not needed when operating under the Replay Xcessory Test Manager. It is described here for users who prefer to use a command-line interface or would like to set up scripts for batch runs.

Batch Mode

In batch mode (using the **-b** option), Replay Xcessory load and use the options saved in the `tcd` file, Replay configuration files, and the command line. Options specified in the command line have more weight and override option values specified in the configuration files. The test application processes run unattended, and the results are placed in the files as specified.

Batch Mode without GUI

This is the general batch mode, except that the **-x** option is specified instead of **-b** and all the GUI output of Replay's driver is forwarded to the virtual Xvnc server. Users must have Xvnc binary installed on their system. The virtual Xvnc server will be run as `display #8`, such as `DISPLAY=127.0.0.1:8`. Please note that all the report output will be dumped to the `stdout` also.

Interactive Mode

In interactive mode, enter options on the command line, as convenient. You can enter additional information in dialog boxes—the same ones provided by **replaytm**.

The following command:

```
replay
```

calls for a play session (the default). It brings up the Play Options dialog box.

With a **-r** option, the command

```
replay -r -tcd xmcabc
```

or

```
replay -r
```

brings up the Record Options dialog box.

The following paragraphs describe the available options. For an on-line description, refer to the reference manual page **replay(1)**. You can get an on-line summary by entering the following command:

```
replay -h
```

Replay Command Form

The general form of the **replay** command line is:

```
replay options -- application_name  
application_args
```

For example,

```
replay -r --xm calc
```

or

```
replay -r -tcd xmcalc -- xmcalc
```

Following the descriptions of the command-line options are examples of record and play sessions using the command-line interface.

Options

The following options are available on the command line:

-tpd path	Path represents a path to a test package. Runs all test cases contained in the test package specified.
-tbf path	Path represents a path to a test package batch file. Runs all test cases contained in the test batch file specified.
-ste path	Path represents a path to a test suite. Runs all test cases contained in the test suite specified.
-tcd path	Path represents a path to a test case. Runs the test case specified.
-h	Provides an on-line list of replay options (instead of a record or play session.)
-r	Selects a record session.
-p	Selects a play session (default).
-b	Selects a play session in batch mode.
-E	Exit on a snapshot difference mismatch.
-f	Name of the <code>.Replay_testname</code> file to be associated with a test. The <code>.Replay</code> file contains test override options.

-v	Name of the .Vrdump file to be associated with a test. The .Vrdump file contains the snapshot specification used for snapshot granularity.
-I <i>script_filename</i>	File name of the script file.
-R <i>report_out_file</i>	File name of the file that reports the results of snapshot comparisons and subtest results.
-s <i>snapshot_dir</i>	Name of the directory for snapshots to be taken in the current record or play session.
-B <i>baseline_dir</i>	Name of the directory for baseline snapshots, against which current snapshots are compared.
-D <i>diff_command</i>	The command to make comparisons between current and baseline snapshot files (default: diff).
-T <i>play_speed</i>	A floating point number between -1 and 1 that sets the play speed: -1 is slowest, 1 is fastest, and 0 is the speed of the record session.
-I	Enables replay to pass the standard input to the application processes.
-L <i>dynamic_library_path</i>	Specifies the search path replay uses to find the specific Xt library to load (default: null).
-tclargs <i>args</i>	Specifies arguments to be passed to the Tcl script for a play session. Enclose the arguments in single quotes if more than one argument is being passed.
-xdebug	Start up the interactive Tcl debugger.
-debug	Start up the line Tcl debugger.
-connect	Connect to existing applications.

-appdisplay

Application display name.

Replay Xcessory also allows the use of virtual X servers as targets for this option. The value of the `-appdisplay` option can be `Xvnc`, `Xvfb`, or `Xnest` to have the application under test opened in the appropriate virtual display. Please note that setting this option overrides the option value set in the `tcd` file. For example, to run the batch playback session on the station without a graphical display, all of the following commands could be used:

```
replay -tcd xmcalc -x -appdisplay Xvnc
```

This command will run Replay Xcessory with two virtual displays: one for AUT and one for Replay Xcessory itself.

If the `-I`, `-s`, `-B`, or `-R` options are not specified, the Record or Play Control Panel will have fields that correspond to those options filled with the values specified in the last record or play session used in the current directory.

Command Line Examples

Here are examples of record and play sessions with the `xmcalc` application using the command line interface.

Play Example

The following steps take you through a play session with `xmcalc`:

1. Enter the following commands:

```
replay -tcd
/work/replay/tape/xcessory/examples/testsuite/pack_xmcalc/xmcalc
```

You will see the Play Control Panel dialog and the `xmcalc` application.

Note: If you want to run the application unintended, use the “`-b`” option.

Note: The options in the previous play example are described in the list under “Options.”

2. Press **Play** to begin a play session.

You will see the Replay Xcessory commands and diffs taken in the scrollable script view area as they are interpreted. You may wish to repeat this procedure, setting the **Speed Scale** to different speeds.

Note: Additional options: Users are also allowed to play scripts in batch mode without the use of tcfs (test case definition files). A user might use this if they had test cases stored in a repository and were going to be running the tests from different locations. In this instance, it is advantageous to be as flexible as possible. The following is an example of a playback that does not use the tcd file, but instead specifies the needed files with arguments:

```
replay -p -l testdirectory/a.tcl -B
testdirectory/a.bsl -s testdirectory/a.res -R
../reportsdir/a.rpt/
```

Record Example

The following steps take you through a record session with the **xmcalc** application:

1. Enter the following command:

```
replay -r -tcd  
/work/replay/tape/xcessory/examples/testsuite/pack_xmcalc/xmcalc1
```

First, you see the record control panel, then a window with the **xmcalc** application.

2. To actually start recording, press the **Record** button.
The record control panel appears in the lower right of your screen.
3. Use the calculator to solve an example problem—pressing any buttons in the **xmcalc** application that you wish.
Notice that your actions are displayed in the script view area.
4. At any time when the pointer is in the **xmcalc** application, enter **Ctrl+i**.
This produces a snapshot in the snapshot directory.
5. When you are ready to end the record session, press the **Stop** button.
6. To exit from the **xmcalc** application, press the **AC** button in **xmcalc** with the right-most mouse button.
7. To see the snapshot files that have been created, enter

```
ls a.bsl
```

which displays the snapshot directory included on the **replay** command. This listing might look similar to the following:

```
vr0001.01.  vr0002.snp  vr0004.01.x  vr0005.snp  vrin-  
xwd                               wd           it.01.xwd  
  
vr0001.snp  vr0003.01.x  vr0004.snp  vrfi-       vrin-  
wd          wd           nal.01.xwd  it.snp  
  
vr0002.01.  vr0003.snp  vr0005.01.x  vrfi-  
xwd          wd           nal.snp
```

Obtaining a Test Suite Report

Each play session generates a subtest report that details the result of all the executed subtests. A report summarizing the results of all sessions in a test suite can be generated using the **replayrep** command, for example:

```
replayrep ~/testsuite
```

RECORD AND PLAY SESSIONS*Obtaining a Test Suite Report*

generates a test suite summary report on the test suite located in the `~/testsuite` directory, as shown below. A `-c` option lists individual subtest results as well. A `-g` option generates an HTML report instead of a text report file.

```

Replay Xcessory Test Suite Report
Generated on Tue Jan 18 15:18:57 2005

TEST SUITE:
/p3/replay2+/target/examples/replay/testsuite

TEST PACKAGE: xmcalc1
record/playback demo using Athena widgets

Script:a.tcl                               Passed
Baseline:a.bsl
Result:a.res
Run Command:replay -display :0.0 -p

Run Date:Tue Jan 18 15:17:27 199
Elapsed Time:0:00:20.00
CPU Time:0:00:00.28
Pass Rate:100.0% (2/2)

TEST PACKAGE: xmcalc2
programmed script using an external input data file

Script:xmcalc.tcl                           FAILED
Baseline:
Result:
Run Command:replay -display :0.0 -p -l xmcalc.tcl

Run Date:Tue Jan 18 14:56:28 2005
Elapsed Time:                               0:00:23.00
Pass Rate:                                  90.0% (9/10)

TEST SUITE SUMMARY
-----
Percent of Test Scripts Passed:50.0% (1/2)
Total Elapsed Time:0:00:43.00
Total CPU Time:0:00:00.28

```

Introduction to the Scripting Language

6

Overview

Since the Replay Xcessory scripting language is based on Tcl, some understanding of Tcl is required before reading the Replay Xcessory Command Language chapter (Chapter 6). We recommend that you read this chapter through the sections on quoting if you plan to go beyond the use of recorded scripts.

Introduction

Tcl stands for “tool command language” (pronounced “tickle”). It is a shell-like interpretive programming language that originated at the University of California at Berkeley. Although Tcl has a simple syntax, it provides full-fledged programming constructs. Furthermore, it is a nonproprietary language that has been accepted by many organizations in the technical community.

Replay Xcessory uses Tcl as its scripting language. Each Replay Xcessory action is a Tcl verb.

The level of understanding that is required to write scripts varies with whether you plan to:

- use record session scripts as recorded
- modify and enhance scripts
- write scripts by hand

Note: To learn Tcl basics quickly, you may find it useful to enter simple scripts interactively and see the results immediately. An interpreter called **tclsh** is provided by Replay Xcessory as a learning tool. This interpreter translates each command as it is entered and echoes the command’s return value to the standard output. Set the `TCL_LIBRARY` environment variable to `$REPLAYHOME/lib` and invoke the interpreter by typing:

```
tclsh
```

It is important to remember that **tclsh** only accepts generic Tcl commands and not Replay Xcessory commands.

The classical reference book about the Tcl scripting language is “Tcl and the Tk Toolkit,” authored by John K. Ousterhout and published by Addison-Wesley (1994). For further information also see “Practical Programming in Tcl and Tk (4th Edition),” by Brent Welch, Ken Jones, and Jeffrey Hobbs, published by Prentice Hall PTR (2003), and “Tcl/Tk: A Developer’s Guide (2nd Edition),” by Clif Flynt, published by Morgan Kaufmann (2003). A useful website is <http://www.tcl.tk>.

Command and Script Basics

A script is a series of Tcl commands. It is important to note that *a new line begins a new command*; multiple commands can be placed on one line, but they must be separated by semicolons.

Note: In the commands shown, Tcl commands are shown in Courier (typewriter) bold font. Throughout this chapter, the command result is sometimes displayed on the line after the command in non-bold Courier. This display is used to aid understanding; when *Replay Xcessory* plays scripts, command results are displayed only when requested explicitly; for example, through an **echo** command.

Command Structure

Each Tcl command consists of one or more *words*—sequences of characters with no embedded white space. The first word of a command is the command name. Subsequent words, if any, are command arguments. Each command returns a result: a string of zero or more characters.

Each command has its own syntax, dictated by the command name and the arguments that follow. For example, the following **set** commands have two arguments—the first argument is the variable to set; the second argument is an arbitrary string:

```
set x 1
set y "Any long string"
```

Note: Using Tcl commands:

The most important rule to remember is that each command is always logically on one line. Hence a line that must be split in two due to its length should be terminated by a backslash (`\`). Commands inside curly braces (`{ }`) are also treated as if they were in the same logical line. More information about backslashes and other special characters is provided in the “*Controlling Character Interpretation*” section on page 173.

In addition, putting spaces in arguments without appropriate quoting is a common error of Tcl novices. Make sure that multiple arguments are properly surrounded with quotes.

Data Types—Strings

All command arguments and results are strings; a *string* is the only data type in Tcl. However, Tcl strings can represent integers, floating point numbers, lists, and other entities. A *list* is a string containing words separated by white space (see “*List Commands*” on page 187 for more information about lists).

Each Tcl command has its own requirements for the types of string arguments it accepts; for example:

- a **set** command accepts an arbitrary string
- an **expr** command requires that the string argument contains a series of operators and operands

Arrays

In addition to simple variables, Tcl supports arrays. In Tcl, both array names and multidimensional elements can be arbitrary strings. Arrays are also similar to variables in that they do not have to be defined before use; for example:

```
set color(red) 50
set color(green) 20
set baseColor $color(red)
50
```

Tcl defines a built-in array called **env** where each element corresponds to a currently-defined environment variable; for example:

```
set ICS $env(REPLAYHOME)
```

sets the variable **ICS** to the current setting of the REPLAYHOME environment variable.

Expression Evaluation

Expression evaluation in Tcl is not automatic. When required, evaluation must be explicitly requested. For example, the command:

```
set x 1+2+3
1+2+3
```

performs no expression evaluation; the variable **x** is simply assigned the string **1+2+3**.

Use the **expr** command, as in

```
expr 1+2+3
6
```

to request the evaluation of an expression.

expr also supports a number of math operations and integer/floating point conversion routines.

Comments

To add comments to a script, use the pound sign (#) as the first nonblank character of the line. Everything from the pound sign to a new line is treated as a comment.

```
# This is a comment line
set class XmPushButtonGadget
```

Controlling Character Interpretation

Characters in command arguments are controlled and interpreted by using substitutions or quoting.

Substitution

Tcl supports three types of substitution:

- variable
- command
- backslash

Substitution occurs before commands are interpreted.

Variable Substitution

Variable substitution inserts the contents of a variable into a command argument. To request variable substitution, precede the variable name with a dollar sign (\$). In the example:

```
set x 5
expr $x+3
8
```

the contents of **x** (defined by **set** as the number **5**) is substituted for **\$x**.

Variable substitution can occur anywhere within a command argument, for example:

```
set x 5
expr $x+$x
10
```

causes two substitutions (in this case, **5** plus **5**) within the same argument.

**Command
Substitution**

Command substitution inserts the result of a command (or other script excerpt) into a command argument. To request command substitution anywhere within a command argument, surround the command (or script excerpt) with square brackets ([]):

```
[ script ]
```

The script excerpt is evaluated and the result is substituted. In the example

```
set x 1
set y [expr $x+5]
6
```

6 (the result of **expr \$x+5**) is substituted into the **set** command, setting **y** to **6**.

**Backslash
Substitution**

Backslash substitution uses a backslash character (\) to suppress the special Tcl interpretation of the character that follows it, such as a dollar sign (\$), space, or left bracket ([], causing the special character to be interpreted literally instead of interpreting its Tcl-related characteristics.

The following examples show how a backslash can suppress the special meaning of a space and dollar sign, respectively:

```
set s a\ multiword\ string
a multiword string
set x 5.95; set y \$$x
$5.95
```

Backslash is also used to represent normally invisible characters. For example, **\t** represents a tab; **\n** represents a new line character. The next section, “Quoting,” contains an explanation of how the same multiword string can be expressed using double quotes instead of backslashes.

Quoting

Sometimes it may be necessary to suppress the special meaning of a dollar sign, space, or other character special to Tcl. For example, if an argument is to include embedded spaces, the special meaning of space (argument separation) must be suppressed for that argument. Quoting is a mechanism to suppress the effect of certain special characters.

Tcl provides two forms of quoting—quoting with double quotes and with braces (`{ }`). In both cases, the quote character must be the first character of a command argument. When an argument is passed to the command interpreter for translation, the quote characters themselves are stripped off.

Quoting with Double Quotes

Using double quotes forms one command argument that consists of all the characters that follow the first double quote up to the matching double quote. Within the argument, spaces, tabs, new lines and semicolons all lose their special interpretations; they are treated like ordinary characters. However, dollar signs and brackets retain their special functions.

Variable, command, and backslash substitutions are still performed within double quotes, as in these examples:

```
set s "a multiword string"
a multiword string

set s "The value of \ $x is $x"
The value of $x is 3

set s "The value of \[ expr 3+5 \] is [ expr 3+5
]"
The value of [ expr 3+5 ] is 8
```

Quoting with Braces

Quoting with braces—of the form `{argument}`—is similar to using double quotes, except almost all characters lose their special interpretation. Within a quote using braces, there is no variable substitution or command substitution.

Quoting with braces is commonly used to create lists. For example, the following `set` command

```
set b {c {d e f} { } }
c {d e f} { }
```

constructs a list (**b**), with the following three elements:

- **c**
- a list with three elements—**d**, **e**, and **f**
- an empty list

Note: A new line occurring inside curly braces does not logically split a line in two. This is very important to remember because it is a device that Tcl uses extensively when nesting commands within one another (for example, the block of Tcl commands executed inside a **for** loop is logically just one line).

Variable Manipulation Commands

The **set**, **append**, and **incr** commands are basic to variable manipulation.

set *variable* [*value*]

If *value* is specified, *variable* is set to *value*. The command returns the current value of *variable* as the result; for example,

```
set y 16
```

```
16
```

```
set x "Sums of squares"
```

```
Sums of squares
```

```
set y
```

```
16
```

append *variable* *value1* [*value2* ...]

Appends each parameter, in order, to *variable_name*. If *variable_name* is undefined, it is defined and set to null before the append operation. The command returns the new value of *variable* as the result; for example,

```
set x abcd
```

```
append x efg
```

```
abcdefg
```

incr *variable* [*value*]

Increments *variable* by *value*, if *value* is present; otherwise, it increments *variable* by 1. *variable* and *value* must be integer strings. The command returns the new value of *variable* as the result:

```
set x 0
incr x
1
incr x 5
6
```

Expressions

An *expression* is a combination of Tcl operands and operators. It is evaluated explicitly with the **expr** command, for example,

```
expr 5.0*($b+$c)
```

Operands

An operand can be a variable or a constant.

Variables

A variable name consists of letters, digits, and underscores. Use the **set** command to assign a value to a variable.

Constants

Although the only Tcl data type is a **string**, the strings can be in various forms, from arbitrary sequences of characters to integer or floating-point numbers.

Integers are in decimal unless there is a specification to the contrary. Of the following integers:

```
35
035
0x35
```

35 is decimal; **035** is octal, because it begins with a **0**; and **0x35** is hexadecimal, because it begins with the prefix **0x**.

Tcl accepts any of the forms defined for the ANSI C standard except that the **f**, **F**, **l** (lowercase **L**), and **L** suffixes are not supported. All of the following are valid floating-point numbers:

```
2.1
5.34e+12
5E2
5.
```

Operators

Tcl supports arithmetic, relational, logical, bit-wise, and other operators. Operators in Tcl are similar to the operators in C expressions.

Arithmetic Operators

The arithmetic operators are defined for either **int** or **float** operands:

-a	takes the negative of a
a+b	adds a and b
a-b	subtracts b from a
a*b	multiplies a and b
a/b	divides a by b

If one operand is floating point and the other is integer, the integer is converted to floating point and the result will be in floating point.

Relational Operators

The relational operators are defined for either **int**, **float**, or **string** operands:

a<b	1 if a is less than b ; otherwise, 0
a<=b	1 if a is less than or equal to b ; otherwise, 0
a==b	1 if a is equal to b ; otherwise, 0
a>b	1 if a is greater than b ; otherwise, 0
a>=b	1 if a is greater than or equal to b ; otherwise, 0

For strings, these operators test lexicographical ordering.

Logical Operators

The logical operators are defined for either **int**, **float**, or **string** operands:

- !** **1** if **a** is **0**; **0** if **a** is **1** (logical **not**)
- a&& b** **1** if both **a** and **b** are non-zero; otherwise, **0** (logical **and**)
- a||b** **1** if either **a** or **b** is non-zero; otherwise, **0** (logical **or**)

These operators are typically used in conjunction with relational operators; for example,

```
set x 3
3
expr ($x>=1) && ($x<=10)
1
```

Bitwise Operators

The Tcl bitwise operators manipulate bits of integers:

- &** bitwise AND
- |** bitwise OR
- ^** exclusive OR
- >>** right shift of the number of bits indicated by the right operand
- <<** left shift of the number of bits indicated by the right operand
- ~** one's complement

Ternary Operator

The ternary operator functions as an if-then-else expression, as in the C language. For example, the expression:

```
expr { ($a < 5) ? 1 : 0 }
```

returns **1** if **a** is less than **5** and returns **0** otherwise.

Math Functions

Tcl provides a number of built-in math functions that can be invoked using the **expr** command; for example:

```
expr { sin($x) + sqrt ($y) }
```

The list of built-in functions is provided in the **expr(3)** reference manual page.

Control Flow Commands

The following commands are representative of the control structures in Tcl:

break Terminates the inner loop of a looping command, such as **for**, **foreach**, or **while**. In this example, **break** stops a search when the *test* item is found:

```
foreach word $keyList {
    ...
    if test then {set found 1; break}
}
```

switch *[options] string pattern body [pattern body ...]*
switch *[options] string { pattern body [pattern body ...] }*

Matches *string* against each *pattern* until a match is found. When the matching *pattern* is found, the corresponding body is executed. The result of the executed *script* or an empty string is returned if no *pattern_list* matches.

The second form of **switch** presents the same information but as a single argument. This form is useful when it is convenient to spread patterns and scripts across several lines; for example:

```
switch $file {
*.res {set type "These are results
snapshots"}
*.bsl {set type "These are baseline
snapshots"}
*.tcl {set type "This is a script file."}
*.M   {set type "This is a map file."}
}
```

The options can be **-exact**, which specifies exact string matches, **-regexp**, and **-glob** which specify two different styles of pattern matching.

continue Terminates the current iteration inner loop of looping commands, such as **for**, **foreach**, or **while**. Unlike **break**, which stops a search when the item is found, execution continues with the next iteration of the loop. For example, in these commands:

```
while {$i>0} {  
    incr i -1  
    if expr[$divisor==0] then continue  
    ...  
}
```

continue causes the next iteration of the **while** loop to be executed.

eval *arg* [*arg* ...]

Concatenates the arguments, using spaces as separators, then executes the result as a Tcl script and returns that result. The artificial script:

```
set cmd expr  
set operation 4*4*4  
eval $cmd $operation
```

64

shows the generation and execution of the command **{expr 4*4*4}**. The **eval** command makes it possible to execute generated commands and scripts.

for *init test reinit for_body*

Executes *init* as a script, then evaluates *test* as an expression. If *test* is non-zero, it executes *for_body* as a script. For each subsequent iteration (if any), it executes *reinit* and evaluates *test*. As soon as *test* is **0**, **for** terminates. The **for** command returns an empty string as its result; for example:

```
for {set i 3} {$i>0} {incr i -1} {
    append answer "time $i "}
set answer
time 3 time 2 time 1
```

foreach *variable-name list body*

For each element of *list* (a Tcl list), the **foreach** command:

sets *variable-name* to the current list element

executes *body*, a Tcl script

This example squares the values in a list:

```
set squares {}
set numList {3 7 9 11}

foreach num $numList {
    append squares " " [expr $num*$num]}
set squares
9 49 81 121
```


if *expr1* [**then**] *body1* **elseif** *expr2* [**then**] *body2*... [**else**] *bodyN*

Evaluates *expr*, as an expression. The value of *expr* must be boolean (that is, numeric: nonzero or zero). If *expr1* is nonzero, *body1* is executed. Otherwise, the first **elseif** expression, if any, is evaluated. If it is nonzero, the corresponding *body* is executed; otherwise, the next **elseif** expression, if any, is evaluated, and so forth. If none of the preceding expressions is nonzero, the **else** *body*, if any, is executed.

```
set x $zz
5
if {$x==4} {
set res 1
} elseif {$x==6} {
set res 2
} else {
set res 3
}
3
```

source *file_name*

Reads the file *file_name* and executes its contents as a Tcl script. The **source** command returns the results of the script's execution.

while *test while_body*

Evaluates *test*. If *test* is nonzero, it executes *while_body* as a script. The **while** command keeps executing until *test* is zero, as in the following example:

```
set answer {}
set i 3
while {$i>0} {incr i -1; append answer $i "
"}
set answer
2 1 0
```

Note: Unmatched braces, parentheses, and other syntax errors can be found using the **tclcheck** utility. **tclcheck** accepts the name of the Tcl script as input.

Procedures

A Tcl *procedure* is a user-defined Tcl command. After definition, procedures are used the way built-in Tcl commands are used. That is, the procedure name becomes the name of a new Tcl command; *procedure* arguments become *command* arguments.

A procedure is defined through the **proc** command, which provides for specification of the procedure name, an argument list, and a procedure body. The *procedure body* is a Tcl script that is executed when the procedure is called.

For example, the following **proc** command defines a procedure that computes the average of two values:

```
proc getavg {x y} {
    expr ($x+$y)/2
}
```

After its definition, **getavg** can be used in the same way as a built-in Tcl command, as in this interactive example:

```
getavg 32.5 98.75
65.625
```

Within a script, a procedure call can also use one or more variables, as in the following example which is defining the average of several numbers:

```
set z [getavg $x $y]
```

Return Value

The *return value* of a procedure is the value of the last command executed within the procedure, often the last statement in the procedure. The **return** command enables an exit from anywhere in the procedure and the specification of a return value.

In some ways, a procedure call is similar to executing the script (procedure body) in-line. There are also some differences, for example:

- In a procedure, variable references refer to local variables—unless explicitly declared global through the **global** command.
- Procedure arguments are substituted in the script body, but cannot be set (the way they could be in line), because they are passed by value.

Global and Local Variables

By default, variables used within a procedure are local to that procedure. The **global** command changes the status of its *variable_name* arguments from local to global.

Since **global** is a command, rather than a definition, the variables used in a procedure are local until the **global** command that includes them is executed. The following example shows the way Tcl local and global variables work:

```
proc gproc {} {  
    global y  
    set x "cat"  
    set y "dog"  
}  
  
set x 3  
set y 5  
gproc  
echo $x
```

```

3
echo $y
dog

```

After the call to **gproc**, the value of **x** is unchanged, even though there is a **set x** command in **gproc**, because the **x** that **gproc** sets is local. The references to **y** inside and outside **gproc**, however, refer to the same variable, because of the **global** command. This is shown by the value of **y**, which is now **dog**.

Variable Number of Arguments

A particularly useful feature of Tcl procedures is the support of a variable number of arguments—provided through the **args** argument. If the final argument is named **args**, then **args** is a list that contains the rest of the arguments. Commands that work with lists, such as **foreach**, can process the variable arguments one by one.

Here is another version of the **getavg** example that computes the average of a variable number of values. In this case, **args** is the last (and only) argument.

```

proc getavg {args} {
    set sum 0
    set n 0
    foreach item $args {
        incr n
        set sum [expr $sum+$item]
    }
    set avg [expr ($sum/$n)]
    return $avg
}

```

Here is an example that shows this new version used interactively:

```

getavg 1.25 2.14 .56 2.0
1.4875

```

Procedure Commands

The following commands are associated with procedures:

global *name1 [name2 ...]*

Binds the names listed to global variables. The **global** command is used in procedure definitions because variables within a procedure are, by default, local to that procedure. Since **global** is a command, rather than a definition, any variables used within a procedure are local until **global** is executed; **global** returns an empty string.

proc *name arg_list body*

Defines a procedure named *name*. Each element of the list *arg_list* is a procedure argument, and *body* contains a script—the procedure body; **proc** returns the value of the last command executed in the procedure.

return [*value*]

Causes an immediate return from the procedure. If *value* is present, the procedure returns the specified value.

List Commands

The list feature in Tcl provides a convenient means to handle a collection of entities, for example, file names, words, or integers. List elements, like command arguments, are separated by white space. So, the command

```
set animal "cat dog canary hamster"
```

creates a simple list with four elements

```
cat dog canary hamster
```

Quoting with braces is commonly used to create lists. For example, the following **set** command

```
set zlist {s {t u v} { } }
s {t u v} { }
```

constructs a list, specified as **zlist**, with the following three elements:

- **s**
- a list with three elements—**t**, **u**, and **v**
- an empty list

Here are some of the common list commands:

concat *list_1 list_2 [list3...]*

Concatenates the lists in order to form a single list; for example:

```
set animal "cat dog canary hamster"
concat $animal {warthog tiger}
cat dog canary hamster warthog tiger
```

join *list separator*

Concatenates the lists' elements together with *separator* and returns the result; for example:

```
set animal "cat dog canary hamster"
join $animal " -- "
cat -- dog -- canary -- hamster
join converts a list to a string; split
performs the opposite operation.
```

lappend *variable_name value [value ...]*

Appends each *value* to the variable as a list element, and returns the new value of the variable; for example, the previous list **animal** could have been created without quotes as follows:

```
set animal cat
lappend animal dog canary hamster
cat dog canary hamster
```

lindex *list index*

Returns the element specified by *index*; for example:

```
set animal "cat dog canary hamster"
lindex $animal 0
```

cat

```
lindex $animal 3
```

hamster

Tcl uses zero indexing, so the first element is always element **0**.

linsert *list index value value ...*

Inserts each *value*, as a list element, before the element specified by *index*. Here is an example that inserts “aardvark” and “anteater” at the beginning of a list:

```
set animal "cat dog canary hamster"
linsert $animal 0 aardvark anteater
```

aardvark anteater cat dog canary hamster

list *value value ...*

Creates and returns a new list in which each *value* is an element; for example:

```
set animals [list avocet egret stilt]
avocet egret stilt
```

llength *list*

Returns the number of elements in *list*; for example:

```
set animal "cat dog canary hamster"
llength $animal
```

4

lrange *list index1 index2*

Returns a list consisting of element *index1* through *index2* of list; for example:

```
set animal "cat dog canary hamster"
```

```
lrange $animal 1 2
```

```
dog canary
```

lreplace *list index1 index2 value value ...*

Returns a new list formed by replacing elements *index1* through *index2* with zero or more new elements, each formed from one *value* argument; for example:

```
set animal "cat dog canary hamster"
```

```
lreplace $animal 1 2 wolf hawk
```

```
cat wolf hawk hamster
```

lsort *list*

Returns a new list made by sorting the elements of *list* in alphabetical order; for example:

```
set animal "cat dog canary hamster"
```

```
lsort $animal
```

```
canary cat dog hamster
```

split *string [split_chars]*

Returns a list formed by splitting the string at *split_chars*; for example:

```
split 9/14/93 /
```

```
9 14 93
```

This command converts a string to a list; **join** performs the opposite operation.

String Manipulation Commands

The string facilities of Tcl are similar to those provided by the C library, with a few additions. These are the string manipulation commands in Tcl:

- | | |
|-----------------------|--|
| format | <i>format_string</i> [<i>value</i>] [<i>value...</i>] |
| | Returns the characters of <i>format_string</i> with converted value arguments replacing % sequences. The format command is the output analog of the scan command. It supports the same formatting facilities as the ANSI C library function sprintf . |
| regexp | Determines whether a regular expression matches part or all of a string. The syntax of the regular expression is similar to that of the grep command. See the regexp(3) reference manual page for a full description. |
| regsub | Determines whether there is a regular-expression match, as with regexp , and substitutes a string for the matching portions. |
| scan | <i>string format</i> [<i>var_name</i>] [<i>var_name...</i>] |
| | Reads characters from <i>string</i> , interprets them according to the % specification of <i>format</i> , and places the converted values into the <i>var_name</i> variables. The scan command is the input analog of the format command. It supports the same formatting facilities as the ANSI C library function scanf . |
| string compare | <i>string1 string2</i> |
| | Returns -1 , 0 , or 1 if <i>string1</i> is lexicographically less than, equal to, or greater than <i>string2</i> . |
| string first | <i>string1 string2</i> |
| | Returns the index in <i>string2</i> of the left-most substring that matches the characters in <i>string1</i> , or -1 if there is no match (see also string last). |

string index	<i>string1 char_index</i> Returns the character of <i>string</i> that has index <i>char_index</i> if that character exists or an empty string if there is no such character. The first character in a string has index 0 .
string last	<i>string1 string2</i> Returns the index in <i>string2</i> of the right-most substring that matches <i>string1</i> , or -1 if there is no match.
string length	<i>string</i> Returns the number of characters in <i>string</i> .
string match	<i>pattern string</i> Returns 1 if <i>pattern</i> matches <i>string</i> using Tcl pattern-matching rules (using <i>*</i> , <i>?</i> , <i>[]</i> , and <i>\</i>) and 0 if it doesn't.
string range	<i>string first last</i> Returns the substring of <i>string</i> between the indexes <i>first</i> and <i>last</i> .
string tolower	<i>string</i> Returns the characters of <i>string</i> with any uppercase characters converted to lowercase.
string toupper	<i>string</i> Returns the characters of <i>string</i> with any lowercase characters converted to uppercase.
string trim	<i>string [chars]</i> Returns the characters of string, but stripping off certain leading and trailing characters. The <i>chars</i> argument specifies the characters to strip; <i>chars</i> defaults to white space characters (space, tab, new line, and carriage return).
string trimleft	<i>string [chars]</i> Similar to string trim , except that only leading characters are stripped off.

string *string* [*chars*]
trimright Similar to **string trim**, except that only trailing characters are stripped off.

File Access Commands

The file access facilities of Tcl are similar to those provided by the C standard library, with a few additions. These are the file access commands in Tcl:

cd [*dir_name*]
 Changes the working directory to *dir_name* if present. If omitted, *dir_name* defaults to the home directory, as given by the HOME environment variable.

close *fileId*
 Closes the file specified by *fileId*; returns an empty string.

eof *fileId*
 Returns **1** if an end-of-file condition has occurred on *fileId*; otherwise, returns **0**.

file *option file_name* [*arg*] [*arg...*]
 Performs one of several supported operations on the file name or on the corresponding file.

flush *fileId*
 Writes out any buffered output that has accumulated for the specified file.

gets *fileId* [*var_name*]
 Reads the next line from *fileId*, discarding its terminating new line character. If *var_name* is present, **gets** places the line in the variable and returns a character count as the command result (or an empty string for end of file). If *var_name* is omitted, the line is returned as the result (or an empty string is returned for end of file).

glob [-nocomplain] [*pattern*] [*pattern...*]

Returns a list of all file names that match any pattern argument. This is the file name expansion shell that languages do automatically; Tcl performs this expansion only when requested explicitly—through **glob**. The **glob** command uses **csh** rules for pattern matching (with special characters **?**, *****, **[**, **{**, and ****). When **-nocomplain** is specified, an error message is returned if no file names match the pattern.

open *file_name* [*access*]

Opens the file *file_name* in the mode given by *access*; returns a file identifier for use in commands such as **gets** and **close**. These are the values for *access*:

- r** Opens for reading only; the file must already exist. This is the default access mode, if *access* is omitted.
- r+** Opens for reading and writing; the file must already exist.
- w** Opens for writing only. If the file already exists, discards previous contents; otherwise, creates a new file.
- w+** Opens for reading and writing. If the file already exists, discards previous contents; otherwise, creates a new file.
- a** Opens for writing only and sets the initial access position to the end of the file (that is, opens for an append). If the file does not exist, a new file is created.
- a+** Opens for reading and writing and sets the initial access position to the end of the file (that is, opens for an append). If the file doesn't exist, a new file is created.

- puts** *[no_new_line] fileId string*
- Writes the string and a terminating *new_line* character to the file specified by *fileId*. If *no_new_line* is present, the terminating new line is not written.
- pwd** Returns the full path name of the current working directory.
- read** *fileId string [no_new_line]*
- Reads and returns all the bytes remaining in file *fileId*. If *no_new_line* is specified, then the final new line character, if any, is dropped.
- read** *fileId num_bytes*
- Reads and returns the next *num_bytes* bytes from *fileId*, or up to the end of the file, if fewer than *num_bytes* bytes are left.
- seek** *fileId offset [origin]*
- Positions the file *fileId* so that the next access starts at *offset* bytes from *origin*. *Origin* can be **start**, **current** or **end**, with **start** as the default. Returns an empty string.
- tell** *fileId*
- Returns the current access position for *fileId*; that is, the byte offset from the file origin.

Extended Tcl Commands

Replay Xcessory also accepts most commands from an extended Tcl command set, referred to as “TclX.” Extended Tcl commands that may be useful in developing Replay Xcessory scripts include:

- additional specialized control flow commands (**for_array_keys**, **for_recursive_glob**, **loop**, and so on)
- Tcl debugging and development commands (**cmdtrace**, **profile**, and so on)
- UNIX/Linux system commands for signal or process handling, file and user attributes modification, and date and time conversion
- file scanning commands that allow the type of processing usually performed with **awk**
- additional interfaces to the math commands, including a random number generator
- additional list processing and string manipulation commands, including a special type of list—called a *keyed list*—that can be used when a C-type data structure is required

The extended Tcl commands are described in detail in the **TclX(3)** manpage, which you can access by typing:

```
$ man tclx
```

Replay Xcessory Command Language

7

Overview

This chapter describes the commands used by the Replay Xcessory command language—the language used to create Replay Xcessory scripts. Since Replay Xcessory commands are based on Tcl, you should read about Tcl in Chapter 6 before beginning this Chapter.

Introduction

Replay Xcessory commands are extensions to the Tcl language and therefore follow Tcl conventions for argument separation, command and variable substitution, and quoting. Replay Xcessory commands fall into these categories:

- user interaction commands
- test management commands
- widget information commands
- session management commands

Information about initialization scripts, accessing command-line arguments, and run time parameters is provided at the end of this chapter. Replay Xcessory also provides a set of extended commands for use with OSF/Motif widgets. The extended commands are described in Chapter 8.

User Interaction Commands

These commands represent user interactions through mouse, keyboard, and window menus.

Mouse Commands

The **click**, **dblclick**, and **multiclick** mouse actions perform a single, double, or multiple clicks on a widget. The **nclick** parameter indicates the number of consecutive clicks in a **multiclick** action. The **press** and **release** are used when the mouse button is held down for longer than a click. The **drag** tracks some or all mouse movements, depending on the setting of the **Button-Up-Motion** record option. Note that a separate *delay* is associated with each coordinate in a drag verb. The **mousemove** command moves the mouse pointer to a widget and/or location.

click	<i>widget state [location [delay]]</i>
dblclick	<i>widget state [location [delay]]</i>
drag	<i>widget state [location [delay]][location [delay] ...]</i>
multiclick	<i>widget state nclicks [location [delay]]</i>
press	<i>widget state [location [delay]]</i>

release *widget state [location [delay]]*

mousemove *widget [location [delay]]*

Moves the mouse pointer to the specified widget. In the case that the location parameter is absent, the pointer will be centered on the specified widget. Otherwise, it will be shifted from the upper left corner of the widget to the coordinates entered by the user.

Note: The various types of click commands are generally not equivalent because Replay Xcessory ensures that machine or network load has no effect on the interpretation of mouse events; two consecutive click actions are not treated as a double click, and vice versa.

Specifying the Target Widget

The widget is identified by its name, which can be fully-qualified or minimized. If the widget ID is known, that can also be used. However, since the widget IDs change with each session, they should only be used if the widget ID was obtained in the same session. The widget ID is returned by some commands (see “*Widget Information Commands*” on page 206).

Some applications also use a subwindow in addition to a target window; in such cases, Replay Xcessory records a subwindow designation. This designation can be in one of the following forms:

- *widget+DC*—The widget has Motif DragContext, but no widget could be found lower than the subwindow.

```
press {my_button+DC} Button1
```
- *widget+DC:subwindow_widget_name*—The widget has Motif DragContext. The *subwindow_widget_name* is the lowest child window that is a widget.

```
press {my_button+DC:.my_app/topLevelShell}
Button1
```
- *widget+S:subwindow_widget_name*—The event subwindow is a widget with *subwindow_widget_name* as its identifier.
- *widget+WM:subwindow_widget_name*—The event subwindow is not a widget, but a lower child subwindow window was found that is a widget.

```
press {my_button+WM:.my_app/topLevelShell}
Button1
```

- *widget+FW:subwindow_widget_name*—The event subwindow is a window in another application.

```
press {my_button+FW:.my_app/topLevelShell}
Button1
```

- *widget+FWD:subwindow_widget_name*—The event subwindow is a window in another application with Motif **DragContext**.

```
press {my_button+FWD:.my_app/topLevelShell}
Button1
```

In cases where the widget name is **<root>** and there is a corresponding subwindow, the recorded coordinates of the transaction will be relative to the subwindow; in this case, **<root>** is a special *widget_name* that represents the **RootWindow**.

```
press {<root>+DC:.my_app/topLevelShell} Button1
```

Specifying Button State

Mouse buttons, keys, or combinations, are represented by one or more of the following key words:

```
Btn1Down  Btn2Down  Btn3Down  Ctrl  Shift  A
lt  Mod1...Mod5
```

Two or more keywords are separated by a plus sign (+). The combination of all keywords, except the last one, represent the state of the keyboard and mouse at the same time the button action (press or release) was activated. The button activated is indicated by the last keyword, which must be one of **Button1** through **Button5**, for example:

```
release {*Gadget} Shft+Btn1Down+Button1 @44,12 27
```

The presence of **Shft+Btn1Down** shows that the **Shift** key and mouse button 1 was down during the **release** command; the presence of **Button1** shows that mouse button 1 was released.

Specifying Cursor Location

Location can be specified in three ways:

- `@x,y[d]` specifies the x and y coordinates in pixels:
 - for window commands, coordinates are relative to the root window
 - for all other commands, coordinates are relative to the upper left corner of the widget
 - the optional qualifier, *d*, indicates which edge of the widget the pointer was crossing; this notation is used by Replay Xcessory when recording window crossing events. The position value of the qualifiers is **T** (Top), **B** (Bottom), **L** (Left), **R** (Right).
- */string* is a string that matches an item to be selected; for example, a file name in a list of files. This specification style is used for Motif list widgets; C shell wild card notations (`*`, `?`, `[]`) are accepted.
- *:position* specifies a relative position in Motif text widgets.

When an optional *location* is omitted, Replay Xcessory determines the cursor position, typically placing it at the center of the given widget.

Keyboard Commands

The following commands represent the input of text from the keyboard:

text *widget* "*text_string*" [*delay*]

Sends a series of key-press events to the specified widget, as if a user were typing the characters; for example:

```
text *text_box_2 "A long text string"
```

key *widget* *key_name* [*delay*]

Sends a key-press event to the specified widget; for example:

```
key *text_box BackSpace
```

causes a key-press event to be sent to the ***text_box** widget, as if a user typed a backspace in the ***text_box** widget. Use **key** to send unprintable characters.

The easiest way to determine the proper key name is to hit the desired key or key combination while in a record session. The names are usually listed under `/usr/include/X11/keysym.h` or files included by it.

Window Management Commands

These commands represent the selection of menu commands from the window menu and, in some cases, actions on the window frame.

move *widget location [delay]*

Specifies the movement of a shell window by the window manager. The location is specified as @x,y coordinates (the @ symbol meaning located “at” x and y) relative to the root window.

resize *widget widthXheight [location [delay]]*

Specifies the interactive resizing of a shell window by the window manager. The optional @x,y location can be used if resizing also causes the top left corner of the window to change; that is, if the window was resized through the resizing handles of the left corner of the window. Resizing can move a window; *widthXheight* specifies the position of the window after resizing. The *delay* argument can be present only if *location* is also present.

iconify *widget [delay]*

Specifies that a shell window has been iconified.

deiconify *widget [delay]*

Specifies that a shell window has been deiconified.

raise *widget [delay]*

Specifies that a shell window has been raised to the top of the window stack.

lower *widget [delay]*

Specifies that a shell window has been lowered to the bottom of the window stack (not recorded).

closewin *widget [delay]*

Specifies that a shell window has been closed from the window manager menu.

Miscellaneous User Interaction Commands

These commands represent miscellaneous user interaction commands that do not fit in any of the previously described categories.

popdown *widget*

Records that a pop up widget, such as a dialog box, disappeared. On playback, **popdown** causes an automatic synchronization with the widget popping down before continuing with the next script action.

popup *widget*

Records that a pop up widget, such as a dialog box, appeared. On playback, **popup** causes an automatic synchronization with the widget popping up before continuing with the next script action.

getconnected apps Returns a list of applications currently connected to Replay Xcessory. If no applications are connected, **_NULL_** is returned.

message *widget msg_type_ID msg_data msg_fmt [delay]*

widget specifies the widget that received this ClientMessage event.

msg_type_ID is the name of a property; for example: **_MOTIF_WM_OFFSET**.

msg_data is the message data, itself.

msg_fmt is **32**, **16**, or **8**. These alternatives specify the way the ClientMessage is formatted: as **32-**, **16-**, or **8-bit** entities, respectively.

The **message** command transcribes a ClientMessage event as sent to the process under test. ClientMessage events are received when sent to the test process by another application process or by itself. Some window manager actions also generate client messages (not recorded).

isconnecteda pp Returns **TRUE** if the specified application is currently connected to Replay Xcessory. **FALSE** is returned otherwise.

- prompt** Provides a way to create a simple prompt containing the prompt text and an input text field. This might be useful where the test script needs parameters that must be entered interactively.
- version** Returns the Replay Xcessory version number as a string, as in “3.0”.

Test Management Commands

The subtest management commands provide for the splitting of a single test script into a number of subtests. The success or failure can be determined explicitly by pass/fail commands or implicitly by snapshots.

- Explicit pass/fail commands must be used in conjunction with some programmatic mechanism for determining that the process under test behaved as expected. This mechanism could be by querying the widget state via the **getvalue** command (see “*Getting and Setting Widget State*” on page 210 for information about **getvalue**) and comparing against an expected value. It could also be by executing some external command via the **system** or **exec** call.
- Implicit pass/fail occurs when a snapshot command is used. If the comparison finds a mismatch, an implicit failure occurs.

To prevent inconsistencies resulting from multiple snapshots or pass/fail commands being associated within a subtest block, a match is considered to have failed whenever any fail command is encountered or any snapshot failure occurs. Run time failures, such as the inability to find a widget, also result in subtest failure.

Subtest *subtest_name*

subtest_name is an arbitrary string that identifies a specific subtest. In some situations, a serial number is preferable; at other times, a word or quoted string.

subtest marks the beginning of a subtest within the script. The end of the subtest is marked by the subtest case verb or by the end of the script. The part of the script that appears prior to the first explicitly identified subtest is implicitly associated with a subtest called “Unnamed.”

pass *pass_annotation*

pass_annotation is an optional string to be printed in the subtest report.

pass registers the success of a subtest or part of a subtest.

fail *fail_annotation*

fail_annotation is an optional string to be printed in the subtest report.

fail registers the failure of a subtest or part of a subtest.

snapshot [**-object** | **-window** | **-viewable** | **-full**] [*widget*]

Indicates that a snapshot was taken during the record session—an image or widget snapshot or both, depending on the **.Vrdump** file. During play, a comparison is made between the baseline and the result snapshots

If **-object** or **-window** is specified, the snapshot only includes the widget subtree anchored at the specified *widget*; popup branches are not included in the snapshot.

If **-full** is specified, a full snapshot is taken which includes the complete widget tree of all application shells of the application, including popup branches.

A **-viewable** entry is similar to **-full** except that only the visible portion of the widget tree is included.

Please note that if the AUT is being run in an Xnest server and the Xnest main window is partly moved off the screen border, the resulting snapshot will be appended as blank areas, since Xnest is not able to get the image of invisible areas.

takesnap **snapname -image | -subimage widget [geom WIDTHxHEIGHT+X+Y]**

Takes an image widget snapshot without making an attempt to compare it with the baseline. **snapname** is the file name where the results should be stored.

In the case that **-image** is specified, the only remaining needed parameter is “widget”.

In the case that **-subimage** is specified, the user should specify the widget and **-geom** option with properly formed coordinates of the needed area.

mergespec [**-file | -string**] *dump_spec*

Used to change snapshot granularity during a record or play session. It merges dump specification resources for the process under test. A first argument of **-file** merges the second argument as a file name; a first argument of **-string** merges the second argument as a string.

loadspect [**-file | -string**] *dump_spec*

Used to change snapshot granularity during a record or play session. It loads or replaces dump specification resources for the application under test. A first argument of **-file** loads the second argument as a file name; a first argument of **-string** loads the second argument as a string.

Note: Both **mergespec** and **loadspect** are automatically generated by the Replay Xcessory **Snapshot Specification Entry** form.

Widget Information Commands

The following commands provide a mechanism for gathering information about widget information, including identification of a specific widget as well as its location.

Widget Identification

The following commands are used for widget identification.

currentwin	<i>widget</i>	Sets <i>widget</i> to be the current window (a shell widget). All subsequent user interactions are named relative to the current window; however, a user interaction command can be named using an absolute path by prefixing a dot to the name of the command.
getcurrentwin		Returns the current window as an absolute name.
widgetid	<i>widget</i>	Converts a widget name to its widget ID for use later in the same session.
windowid	<i>widget</i>	Returns the window ID of the specified widget for use later in the same session. The window ID can be used with a number of X Window utilities (such as xwd).
alias	<i>alias_name widget_name</i>	Assigns another name to a widget. The <i>alias_name</i> can be any Tcl variable name.
widgetname	<i>widget</i>	Returns the fully-qualified name of a specified widget, which could be identified by its widget tag or ID.
widgettag	<i>widget</i>	Returns the fully-qualified tag to the specified widget, which could be identified by its widget name or ID.
iswidget		Can be used to determine whether a widget exists. This might be useful for tests where it is important to determine whether a widget has already been created or destroyed. Returns TRUE or FALSE .

getfocuswidget Returns the widget that has the current input focus, given a shell widget. This function is limited to OSF/Motif widgets.

isconnectedapp *app*
Returns TRUE if the specified application is connected, and FALSE otherwise.

Walking the Widget Tree

The following commands provide a mechanism for walking the widget hierarchy.

They provide flexibility for programmatic script generation. For example, they can be used to ensure that all widgets are exercised in some manner.

getclass *widget*

Returns the widget class name of *widget*.

Example:

```
echo "Classname of { *button40 } is
[getclass *button40]"
Classname of { *button40 } is Command
```

(See “*Miscellaneous User Interaction Commands*” on page 203 for more information about the **echo** command)

getparent *widget*

Returns the fully-qualified name of the parent of *widget*.

Example:

```
echo "getparent of { *button40 } is
[getparent\ *button40]"
getparent of { *button40 } is ti
```

getchildren *widget*

Returns a space-separated list of the widget names of all regular children of *widget*.

Example:

```
echo "getchildren of { *ti } is
[getchildren *ti]"
getchildren of { *ti } is bevel
button1 button2 button3 ... button40
```

getpops *widget*

Returns a space-separated list of the widget names of all pop up children of *widget*.

getdescwidget *widget x y*

Returns a top widget that is located in the rectangle the given coordinates belong to and is a successor from the given widget. This command skips all unmapped widgets during processing and returns only the mapped found result.

Example:

```
set widget [getdescwidget{.} 10 10]
```

Assign widget to the top level widget which is on the current shell {.} and shifted 10 points to the right and down from the upper left corner.

This command can be useful in conjunction with the `seektext` command that returns coordinates of some text on the widget surface. For example, if the user is looking for the “tan” string in the `xmcalc` window that represents the tangent button, they will not be able to click on that button, even if they have the proper coordinates. To do this, the user should traverse the widget tree from the known widget to get the widgets which have the known coordinates as a point in the window rectangle.

Getting and Setting Widget State

The **getvalue** command is the primary way for obtaining a widget's state when comparing such states is the desired approach for result verification. The **ismanaged** and **ismapped** commands are useful when it is necessary to find out whether a widget is currently managed or mapped (in the X Toolkit/X library sense of the word). The **setvalue** command can be used to alter widget state and force some path to be taken.

The result is always a character string. The conversion from the internal resource format (which may be an arbitrary data structure or pointer) to a character string is done via resource converters which have been registered by Replay Xcessory or by the application itself. The result type field provides a level of control over the format of the result string.

The result type field can usually be omitted, which means that a converter to type "VrString" is used. If such a converter is not present, a converter to type "String" (same as **XtRString**) is used. If neither exists, the internal data structure is simply returned as a decimal number.

The default converter for resources of type **XtRWidget** to **VrString** converts the widget ID to the fully-qualified widget name. To obtain the widget ID directly, use **Wid** as the result type. Information about registering custom converters can be found in "Writing custom type converters" on page 210.

getvalue *widget_resource_name [result_type]*

Returns the resource value of the *widget*. Resource names can be obtained from the widget documentation, which typically lists them with **XmN** or **XtN** prefixes. Drop the prefix when using these names in **getvalue** or **setvalue** commands. Null values are returned as the string **_NULL_**. See also **setvalue**.

Example:

```
getvalue {*scale_scrollbar}
troughColor
```

getvalue *client_data*

The **getvalue** command now accepts an optional client data argument. This allows you to pass arbitrary data to the **getvalue** command. The data can then be interpreted by custom resource converters registered via **VrSetTypeConverter**. This argument provides greater flexibility to custom resource converters.

See the UNIX/Linux reference pages available with the **man** command for more information about these commands.

setvalue *widget widget_resource_name widget_resource_value*

Sets the specified widget resource of *widget* to the specified value. See also **getvalue**.

Example:

```
setvalue {*scale_scrollbar}  
troughColor blue
```

seektext *widget text [x1[y1[x2[y2]]]]*

Based on the font list composed on the “Recognition” page of the Session properties dialog. It generates a string image and looks for that image on the “widget” surface. “**widget**” is the name of the topLevelShell where the user wants to take a snapshot.

In case the “widget” is absent, the searching is performed using the current shell.

In case the string that is being search for is not found, the result value is “-1x-1x-1x-1”.

If the string is properly found, the resulting value is in the form “X1xY1xX2xY2”, where X1 and Y1 are the coordinates of the upper left point of the rectangle where the string is down and X2,Y2 are the coordinates of the right bottom part of that rectangle.

This command can optionally get several parameters that represent the limitation rectangle where searching should be processed. In the case where x1 or x2 are not specified or specified incorrectly, Replay Xcessory will use zero as their values. In the case where x2 or y2 are not specified, Replay will use the widget width and height respectively.

It is possible to change the behavior of this command for non-batch sessions. The user has the ability to manually check a point on the screen to properly continue playback. Please refer to *Appendix —Controlling Session Properties* in the *Builder Xcessory 4.0 User’s Guide* for more information.

Please note that in the case that Replay Xcessory is unable to seek the needed string and return -1x-1x-1x-1 as the result in debug mode, Replay will automatically turn on the playback pause button. Action buttons will be disabled until the user presses the “Continue” button.

- seektext** (cont.) The user will see short flash while Replay generates the string image to be looked at later. To speed up the search, Replay Xcessory uses two-pass processing. On the first turn, Replay Xcessory generates an image of the first letter of the needed string and looks for all matches on that specified image. (The user can see the generated image of the letter for a small period of time on the playback history or debug history window depending on which mode is active). On the second iteration, Replay generates a full string image and tries to match the full image starting from the earlier found coordinates. At the moment when there is a **full** successful match, Replay finishes searching and returns the coordinates of the upper left corner of the matched string.
- ismanaged** *widget*
Returns TRUE if the geometry of *widget* is currently managed by its parent. This command corresponds to the X Toolkit function **XtIsManaged**.
- ismapped** *widget*
Returns TRUE if the *widget* is mapped or can be viewed.
If the name of the *widget* is invalid, `_INVALID_WIDGET_` will be the return value.
- getproperty** *widget property_name*
Returns the value of X Window System string properties associated with the specified widget window. Properties associated with the root window can be obtained by passing `<root>` as the widget name.

Session Management Commands

This section describes the commands and their arguments used for session management. This means that you can customize the way a session runs by setting certain arguments and variables depending on your own personal preferences of how a session runs.

Play Synchronization

The following commands are used to define play synchronization.

activate *client-tag*

The **activate** Tcl command identifies the target process for the actions that follow. The **activate** argument (*client_Tag*) represents the application name; this argument is usually the second argument to **XtAppInitialize**. If there is more than one instance of the same application, the duplicate instances are indexed by the instance number; for example, the second instance of **xmcalc** would be **xmcalc[2]**. Instance numbers conform to a fixed indexing scheme so they always maintain the same index number even if applications with a lower index number have exited.

The *\$replay_args* (*clientTag*) array field is used to inquire the current client tag.

delay *number_of_milliseconds*

Causes a short delay (specified in milliseconds) before the next script command is interpreted.

pause The **pause** command can be added to a Tcl script. When the command is executed, the replay driver enters pause mode similar to being turned on by using the **Pause** button on the Play Control Panel. To resume, click **Resume** on the Control Panel or use the **Pause** hot key (**Ctrl-p**).

termsync *sync*

The *sync* string can be separated with colons to indicate multiple prompts. The sync string can be initialized by setting the `REPLAY_TERM_SYNC` environment variable. Use **termsync** whenever the sync string needs to change. (Refer to “Using terminal emulators” on page 201 for additional information.)

Session Communication

The **echo** and **echoreport** commands provide a way to display informative messages and are also useful in debugging. The **narrative** command provides a way to use Replay Xcessory to drive self-running demonstration sessions with narratives.

The following commands are used for session communication.

echo	<i>text</i>
	Writes an arbitrary string of text to stdout .
echoreport	<i>text</i>
	Writes an arbitrary quoted string of text to the Test Report file.
narrative	<i>text [location [delay]]</i>
	Displays the specified text in a dialog box, positioned at the specified coordinates. If no coordinates are given, the window manager determines the location. If <i>delay</i> is present, the text is displayed for <i>delay</i> seconds. If <i>delay</i> is omitted, an OK button appears and the text is displayed until the user presses the OK button.
choose	[TextString [Name1 [Name2 [Name3]]]][location]
	This function creates a window that is used to request user feedback. It provides the user information about what button was selected, and then waits for feedback. <i>TextString</i> is the Test to be displayed in the window that specifies the type of information or feedback requested. <i>Name1</i> is the text on the first button. It is “OK” by default. <i>Name2</i> is the text on the second button. It is disabled by default. <i>Name3</i> is the text on the third button. It is disabled by default.
	This function returns 3 values: 0, 1, or 2 if the first, second, third buttons, respectively, is selected.
	The <i>location</i> is specified as @x,y coordinates (the @ symbol meaning located “at” x and y) relative to the root window.

Note: The delay specified in the explicit **delay** command and the **narrative** command is independent of playback speed

Application Management

The following commands are used for application management.

startup *application args*

Automatically records the application under test and its parameters at the beginning of the record script. If multiple application processes are started, those names and arguments are recorded within the script.

connect [*client-tag ...*]

Connect to existing applications that are linked with the Replay Xcessory X Toolkit library.

It returns 0 if at least one of the specified applications is connected and 1 if it failed to connect to any of the specified applications.

The return value can be used for setting an application delay time limit:

```
set w 10
set ok 0
for {set i 0} {$i<$w} {incr i 1} {
    set ret [ connect xmcalc ]
    if { ${ret} == 0 } {
        break
    }
    delay 1000
}

if { ${ret} == 1 } {
    puts "Could not connect during 10
seconds!!!"
}
```

system *shell-script-string* [**continue**]

Executes the shell script in the command argument. You can execute a multi-lined script by quoting it with curly braces ({ }). If the **continue** option is present, execution continues without waiting for completion of the shell command; otherwise execution waits for command completion. The return value is the return value of the command executed.

exec *[switches] arg [arg...]*

Treats its arguments as the specification of one or more subprocesses to execute. The arguments take the form of a standard shell pipeline where each *arg* becomes one word of a command, and each distinct command becomes a process. **exec** is a standard Tcl command and not a Replay Xcessory extension. **exec** is useful for setting Tcl variables to **stdout** of a UNIX/Linux system command. See the **exec** reference manual page for further information.

Return value:

Please note that `exec` command is asynchronous and its return value does not represent the exit status of the child process. To get the return value of the child, you should use the “system” command or just use the next kind of approach:

```
exec ./test.sh
while { $replay_args(execReturnValue) < 0 } {
    puts "Script still running..."
}
puts $replay_args(execReturnValue)
```

The special internal variable `$replay_args(execReturnValue)` is set to the return value of the child process when it terminates.

The `exec` command can also be used to get output of the command being executed. For example:

```
set start_time [exec date]

puts $start_time

will print the current date.
```

rxexit This command closes the Replay Xcessory driver in the same manner as if the script ended during the batch mode session.

Initialization Scripts

Replay Xcessory automatically executes a script called **.Vrtclrc** (created in your home directory) before the **play** script is started. If the **.Vrtclrc** script is not located in the home directory, the script located in the directory pointed to by `REPLAY_TESTSUITE_PROPS` is used.

Accessing Command-Line Arguments

Replay Xcessory and **tclsh** provide the following built-in variables for accessing command-line arguments. These arguments must have been passed using the **-tclargs** options to the **replay** command, or via the **Tcl Arguments** field on the Play Control Panel.

argc	the number of arguments
argv	the list of arguments
argv0	the name of the Tcl script

Accessing Run Time Parameters

Several Replay Xcessory arguments are available in play mode using a built-in array called **replay_args**. Individual elements are available as array elements where the element name is the same as the X resource name for the corresponding argument. For example, the playback speed can be accessed as

```
$replay_args (playbackSpeed)
```

and it can be modified within a script using the **set** command

```
set replay_args (playbackSpeed) .80
```

The name of the argument is the same as the corresponding resource name in the session property file (**.Replay**). The following arguments can be accessed, but not modified, with **\$replay_args**:

scriptFile	the name of the Tcl script
appDisplay	the application display
useVirtualKeyNames	record key symbols using Motif key symbol names

baselineDir	the baseline snapshot directory (<code>_NULL_</code> , if not specified)
resultsDir	the results snapshot directory (<code>_NULL_</code> , if not specified)
reportFile	the report file (<code>_NULL_</code> if not specified)
clientPid	the process ID of the current application under test (not available as a resource)
replayPid	the process ID of replay

The following arguments can be accessed and modified with **\$replay_args**. They are divided into three property argument categories:

- general
- play
- record

Refer to “*Controlling Session Properties*” on page 103 or the appropriate reference manual page for additional information about each of these options.

General Property Arguments

Here is a list of the general property arguments:

clientTimeout	maximum application start up time (in milliseconds)
activePause	allows processes under test to be operated during pause mode while recording a play session
compressImageSnapshots	request image snapshots to be stored in a compressed format to save disk space
useTagNames	request for widgets identification by their tags whenever possible
pauseKey	key combination used to request a pause in a session (default pause hot key: Ctrl+p)

stopKey	key combination used to terminate a session (default hot key is: Ctrl+s)
learnTagKey	hot key used to activate a dialog box to learn widget tag names
ungrabPointerKey	key combinations used to ungrab the cursor if it is currently grabbed by Replay Xcessory or by the process under test
compressImageCommand	use to compress image snapshots
uncompressImageCommand	uncompress image snapshots

Play Property Arguments

Here is a list of the play property arguments:

promptOnError	<p>If set to true, a prompt dialog appears whenever an error occurs; this dialog provides two options: the errors can be ignored, or execution of the command can be suspended.</p> <p>If set to false, the prompt dialog does not appear and execution of the command continues; the command being executed returns an error return code when appropriate</p>
promptOnSnapshotDiff	request for a prompt to continue or exit following a snapshot comparison mismatch
displayImageDiff	request for image differences to display in an independent window
exitOnDiffMismatch	request to stop a play session when a snapshot difference is encountered
retryTimeout	maximum delay time, in milliseconds, to allow for widgets to map

REPLAY XCESSORY COMMAND LANGUAGE*Accessing Run Time Parameters*

defaultDelayTime	time used when the delay time is not explicitly included in a script command
keyDelay	time, in milliseconds, to simulate typical lapse between entries of characters
debuggerWarpBackKey	hot key used to return the pointer to the debugger control panel
playbackSpeed	specify the play speed
ignoreImageColor Differences	request to ignore differences in image colors
generateHtmlReport	request to generate an additional HTML report. This option does not affect the generation of text reports.

Replay Xcessory Extended Commands

8

Overview

This chapter describes extended commands of the Replay Xcessory command language. Since the material in this chapter builds on the material in Chapters 6 and 7, we recommend these chapters be read prior to reading this chapter.

Introduction

Replay Xcessory extended commands provide a compact way to specify interactions with certain common OSF/Motif widgets. The commands, which simplify writing scripts by hand, support the following Motif widgets:

- menu items
- scroll bars
- scales
- lists
- text and text fields

Menu Selection Commands

These commands simplify menu selection, which normally involves multiple **click** commands or **press**, **drag**, and **release** commands. Selection of multiple menu items occurs in connection with cascading menus—that is, nested submenus.

There are two sets of menu selection commands:

- a standard set, that uses menu labels to make menu choices
- an alternate set, that uses widget names

Standard Menu Selection Commands

These menu commands can be used when menu labels are known and stable:

menubar_pick

widget menu_labels

Selects one or more menu items from a menu bar. The *widget* parameter identifies the menu bar widget (which is actually the **XmRowColumn** widget that contains the menu panes).

option_pick

widget menu_labels

Selects a menu item from an option menu. The *widget* parameter identifies the option button.

popup_menu_pick *base_widget widget menu_labels button location*

Selects a menu item from a pop up menu. The *base_widget* identifies the widget in which the user presses the mouse button to display the menu. The *widget* identifies the pop up menu. This button identifies which button should actually make the actions. The default value is `Button3`. The *location* is specified as `@x,y` coordinates (the `@` symbol meaning located “at” `x` and `y`) relative to the root window.

This argument specifies the menu choice in the preceding menu-selection commands.

menu_labels A string with one or more labels. Multiple labels are concatenated and separated by semicolons. White space in the menu label is included in the string. Wild cards that are recognized are similar to those used by the C shell: an asterisk (`*`) matches zero or more characters; a question mark (`?`) matches any single character; square brackets (`[]`) enclose sets of characters for single-character matches.

Examples:

```
menubar_pick { *menubar } { File;New }
menubar_pick { *menubar } { File;Open* }
option_pick { *option } { *red* }
popup_menu_pick { *drawing } { *popupMenu }
{ View;Zoom;Larger }
```

All the commands from this section return:

0 = success with active menu item

1 = success with inactive menu item

2 = failure

Alternate Menu Selection Commands

These commands can be used when it is more practical to use the names of menu buttons rather than menu labels, to make menu choices; for example, when the labels are unknown or change frequently.

menubar_select	<i>widget menu_button [menu_button ...]</i>	Selects one or more menu items from a menu bar. The <i>widget</i> parameter identifies the menu bar widget.
option_select	<i>widget menu_button [menu_button ...]</i>	Selects a menu item from an option menu. The <i>widget</i> parameter identifies the option button.
popup_menu_select	<i>base_widget menu_button [menu_button ...] [mouse_button][location]</i>	Selects a menu item from a pop-up menu. The <i>base_widget</i> identifies the widget in which the user presses the mouse button to display the menu. <i>Mouse_button</i> is the mouse button identifier (Button1, Button2, Button3) that can be used to click on the target menu item. The <i>location</i> is specified as @x,y coordinates (the @ symbol meaning located “at” x and y) relative to the root window.

This argument specifies the menu choice in the preceding menu-selection commands:

<i>menu_button</i>	As a series of menu button names, this argument can be used when the menu labels are not known; for example, when menu labels change frequently.
--------------------	--

Examples:

```
menubar_select {*menubar} {*fileButton}
               {*newButton}
option_select  {*option}  {*redButton}
popup_menu_select {*drawing} {*viewButton}
```

```
{*zoomButton}\      {*largerButton}
```

All the commands from this section return:

0 = success with active menu item

1 = success with inactive menu item

2 = failure

Scroll Bar Commands

These commands support operations on horizontal or vertical scroll bars.

scroll_set *widget pixel_position*

Moves the scroll bar slider to the specified number of pixels from the origin. For a vertical scroll bar, the origin, **0**, is at the top; for a horizontal scroll bar, the origin is at the left. If there is an error, it returns 1.

scroll_get *widget*

Assuming that XmScrollBar is the child widget of the XmScale widget, returns value of the parent XmScale as proper widget value. If there is an error, it returns 1, otherwise it returns the value of the scroll position.

scroll_min *widget*

Moves the scroll bar slider to the origin. For a vertical scroll bar, the origin is at the top; for a horizontal scroll bar, the origin is at the left.

scroll_max *widget*

Moves the scroll bar slider to its maximum distance from the origin. For a vertical scroll bar, this moves the slider to the bottom; for a horizontal scroll bar, to the far right.

scroll_line*widget numlines*

Moves the scroll bar slider by the specified number of lines. A line is defined to be the distance moved by the slider when the arrow heads on the ends of the scroll bar are clicked on. When *numlines* is positive the movement is to the right (horizontal) or down (vertical). When *numlines* is negative, movement is in the opposite direction.

scroll_page*widget numpages*

Moves the scroll bar slider by the specified number of pages. A page is arbitrarily defined to be the distance moved by the slider when the scroll region (the area between the two arrows) is clicked on. When *numpages* is positive, the movement is to the right (horizontal) or down (vertical); when *numpages* is negative, movement is in the opposite direction.

proc*ensure_visible parent w_name*

Scrolls both the vertical and the horizontal scroll bars (if any) to make the widget's *w_name* visible on the left top side of the screen. *parent* - is the name of the *XmScrolledWindow* widget that holds the *w_name* widget in *ClipWindow*.

Examples:

```
scroll_set {*scrollbar} 120
scroll_min {*scrollbar}
scroll_max {*scrollbar}
scroll_line {*scrollbar} 2
scroll_page {*scrollbar} -3
```

Scale Commands

Note: The XmScale widget contains two subwidgets: XmLabelGadget (Title) and XmScrollbar (Scrollbar). Replay allows the user to set a value to the XmScale using either the XmScale widget directly or a subitem widget (XmScrollbar). Users who wish to get the value of Scrollbar should use the XmScale widget (base widget) and get the value resource from it. Otherwise, the value will be unpredictable. The Scrollbar subwidget has its own resource “value” that unfortunately does not represent the actual value of the Scrollbar position in the needed widget.

The following commands support the setting of a scale widget.

scale_set	<i>widget value</i>	Sets the scale to the specified value. Returns 1 on error and 0 on success.
scale_get	<i>widget</i>	If the “widget” parameter is the XmScale widget then it returns its “value” resource. In case it is the XmScrollbar widget, then it returns the “value” resource of the parent widget as scroll_get does. Returns 1 on error and the value of the scale on success.
scale_min	<i>widget</i>	Sets the scale to its minimum value. Returns 1 on error and 0 on success.
scale_max	<i>widget</i>	Sets the scale to its maximum value. Returns 1 on error and 0 on success.

Examples:

```
scale_set { *scale } 41
scale_min { *scale }
scale_max { *scale }
```

List Commands

List commands support the selection of *items* (or ranges of items) of a list component. List items or ranges can be identified by specifying the item's text (that is, by pattern matching: */string*) or its coordinate (*@x,y*). The pattern-matching string accepts C shell wild cards: an asterisk (***) matches zero or more characters; a question mark (*?*) matches any single character; square brackets (*[]*) enclose sets of characters for single-character matches. If an item name contains a space (), add a leading backslash (**). For example, for the item "Monday Day", the correct command is *"/Monday\ Day"*.

Coordinates can be unreliable in situations where the list might change size; for example, if the font of the list's text changes.

list_select *widget from_item [to_item]*

Selects all items in the specified range. *to_item* can be omitted if only one item is to be selected. Returns 1 on error and 0 on success.

list_extend *widget from_item [to_item]*

Adds all items in the specified range to the currently selected set of items. *to_item* can be omitted if there is only one item in the range. Returns 1 on error and 0 on success.

list_find_string *widget str*

Selects a listbox entry based on a substring of the text. It returns the result of the underlying click command on success, and 1 in the case that the item does not exist.

list_find_regexp *widget regexp_str*

Selects a listbox entry based on a regular expression of a text. It returns the result of the underlying click command on success, and 1 in the case that the item does not exist.

Examples:

```
list_select {*list} @10,50
list_select {*list} /Monday
```



```
list_extend {*list} /Wednesday
list_select {*list} /Monday /Wednesday
list_extend {*list} /Thursday /Friday
list_select {*list} /\[Mm\]o?day
list_extend {*list} /Wednesd*
```

Note that if a list widget contains items that consist of several words, it is necessary to split the items with a new-line character. For example:

```
set items [ getvalue $widget items]
set ilist [ split $items "\n" ]
return [ lindex $ilist $number_of_item ]
```

The bold string in the example above is very important. If it is not included, part of the random list item will be returned as the result.

Below is an example of a procedure that returns the correct last item of a list:

```
proc getLastItem { widget } {
    set items [ getvalue $widget items ]
    set ilist [ split $items "\n" ]
    set count [ getvalue $widget itemCount ]
    return [ lindex $ilist [ incr count -1 ] ]
}

startup tester
activate {tester}
currentwin {/tester}
puts [ getLastItem {List} ]
```

Text Manipulation Commands

These commands support various operations with text in a text area. The text area is regarded as a continuous string, even if it appears in multiple rows. Location within the text area is given by character position (*: pos*), as shown in Figure 75 .



Figure 75 Text Area

For example, *from_pos to_pos* for the entire string **ABCD** is **:1 :4** and, for **B**, is **:2 :2**.

These are the text commands:

text_set

widget string

Sets the text area to the specified string. You can use multi-line strings as well as single-line strings. Returns 0 on success and 1 on error.

text_select	<i>widget from_pos to_pos</i> Selects the text from <i>from_pos</i> to <i>to_pos</i> , inclusively. Returns 0 on success and 1 on error.
text_select_string	<i>widget string</i> Selects the first occurrence of <i>string</i> . Returns 0 on success and 1 on error.
text_find_pos	<i>widget pos</i> Moves the insertion caret to the specified position. Returns the result of the underlying click command on the needed position.
text_find_string	<i>widget string</i> Moves the insertion caret just before the first occurrence of <i>string</i> . Returns 0 on success and 1 on error.
text_insert	<i>widget pos string</i> Inserts the specified string at the specified position; to insert before the first character, use :0 . Returns 0 on success and 1 on error.
text_delete	<i>widget from_pos to_pos</i> Deletes the text from <i>from_pos</i> to <i>to_pos</i> , inclusively. Returns 0 on success and 1 on error.
text_clear	<i>widget</i> Deletes all the text from the given widget. Returns 0 on success and 1 on error.
text_delete_string	<i>widget string</i> Deletes the first occurrence of <i>string</i> . Returns 0 on success and 1 on error.

text_replace	<i>widget from_pos to_pos string</i>
	Replaces the text in the specified range by <i>string</i> . Returns 0 on success and 1 on error.
text_replace_string	<i>widget from_string to_string</i>
	Replaces the first occurrence of <i>from_string</i> to <i>to_string</i> . Returns 0 on success and 1 on error.

Examples:

```
text_set {*text} {quick brown fox}
text_select {*text} :1 :3
text_find_pos {*text} :3
text_replace {*text} :6 :13 {yellowish}
text_find_string {*text} {brown}
text_select_string {*text} {brown}
text_replace_string {*text} {brown} {red}
text_delete_string {*text} {quick}
```

Note: The colon (:) operators are optional.

Tabs Control Commands

To control Tabs, use the following commands:

search_tab	<i>widget name</i>
	Returns > 0 if Tab <i>name</i> is available. If not found, 0 is returned.
select_tab	<i>widget name</i>
	Opens Tab with the appropriate name. Returns > 0 on success.

XmTree Control Commands

The following are the commands to control XmTree.

search_tree_child	<i>widget</i>	Returns the widget id if <i>widget</i> is a child of the XmTree widget. If the widget is not found or the parent is not the XmTree widget, returns 0.
collapse_node	<i>widget node</i>	Collapses a node that is a child of the XmTree widget. Returns > 0 on success.
expand_node	<i>widget node</i>	Expands a node that is a child of the XmTree widget. Returns > 0 on success.

XmContainer Control Commands

The following are the commands to control XmContainer.

get_container_item	<i>widget item</i>	Returns the value of an item that is a child of the XmContainer widget. Returns > 0 on success.
set_container_item	<i>widget item value</i>	Sets the value for an item that is a child of the XmContainer widget. Return > 0 on success.

Script Debugger

Overview

This chapter describes the Graphical User Interface (GUI) debugger that can be used to view and debug Tcl scripts (described in Chapters 6 through 8).

Introduction

Using the Replay Xcessory script debugger enables you to:

- determine where a program crashed
- view the values of variables and expressions within the script
- run and trace a program
- set breakpoints within the code
- facilitate the setting and clearing of breakpoints

The following sections describe the various menus and windows associated with the Replay Xcessory script debugger. Although emphasis is placed on the GUI interface, included in this chapter are cross-references to command-line usage, as well as a short description of the command-line interface.

Debugger Interfaces

This section describes the Replay Xcessory Tcl debugger. The command-line options used for debugging are also available; however, as a timesaving and easy-to-use debugging tool, it is recommended that you use the graphical user interface. A brief section at the end of this chapter describes the command-line interface.

The Replay Xcessory debugger is used in play mode and can be executed using the command-line or by accessing the Debugger Control Panel through the Replay Xcessory Test Manager. The following sections describe how to use the main play panel, as well as initializing the play session automatically with a command-line option.

Using the Replay Xcessory Script Debugger

The script debugger can be initialized using the Replay Xcessory Test Manager graphical user interface (GUI), or by invoking the Replay Xcessory driver via the command-line.

Starting the Script Debugger Using the Test Manager

The script debugger can be initialized from the Replay Xcessory Test Manager window. The following steps contain a short tutorial of how to start the debugger:

1. Select the directory that contains the script that will be tested.

Note: The following tutorial assumes that the proper environment has been set up as specified in “*Setting up the environment*” on page 47.

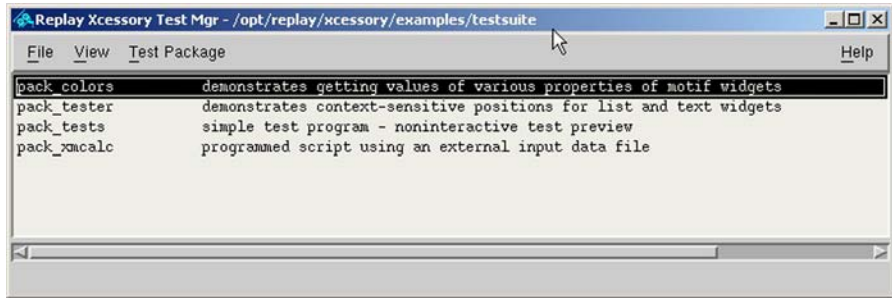


Figure 76 The Test Manager Main Window

The Test Package window appears with icons displayed within the window:

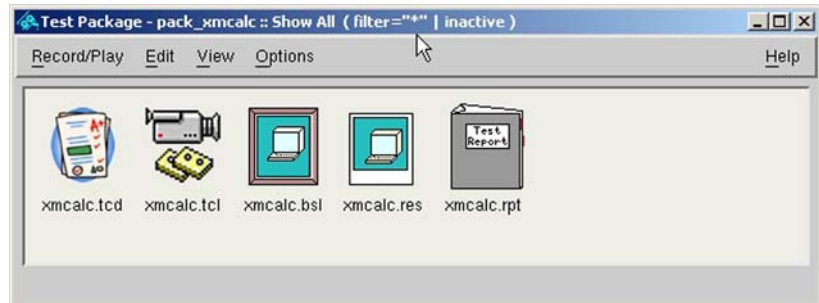


Figure 77 The Test Package Window

2. Click on the icon that will be used for the debugging session (in this case, **a.tcl**) and select **Play** from the **Record/Play** menu.

The main Debug Play Control Panel appears:

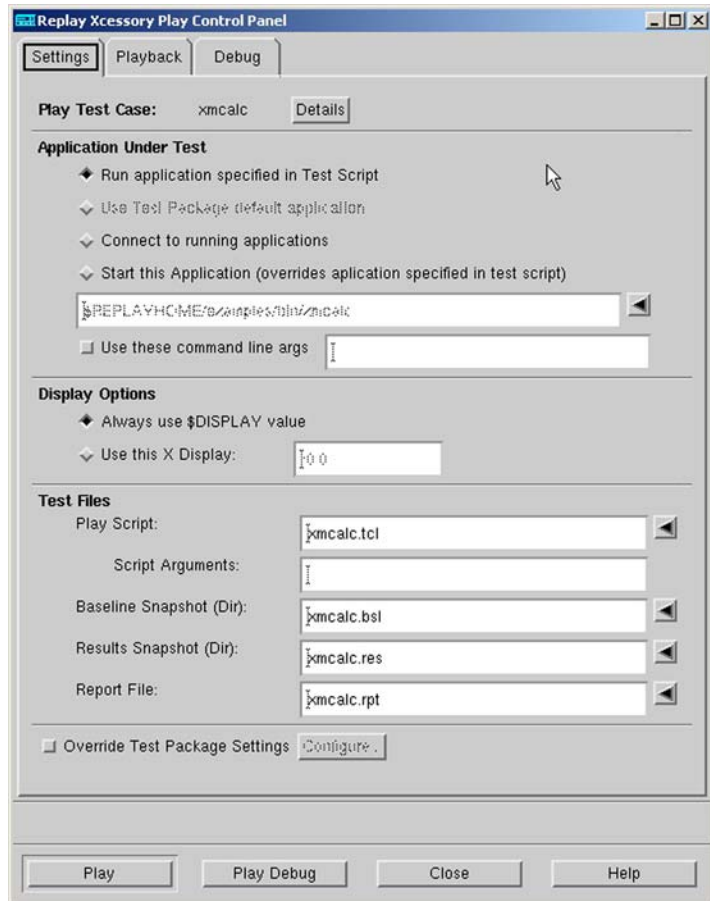


Figure 78 The Main Debug Play Control Panel

The following table lists the fields and buttons on this panel and describes their purposes.

Table 2: Debug Play Control Panel Entities

Application Under Test:	The name of application being tested.
Application Display	The display where the application is being tested.
Play Script	The name of the script being tested (in this case it is called a.tcl).
Script Arguments	Arguments to the script.
Baseline Snapshot (Dir):	This is the directory where the baseline snapshot resides.
Results Snapshot (Dir)	This is the directory where the results snapshot resides.
Report File	This is the name of the report file that is generated.
Slider (Speed)	This slider can be set to run the debugger at a rate determined by the user
Prompt on Snapshot Difference	During playback snapshot comparison (which is essentially a diff of two files), a dialog box pops up and asks if you want to continue; if you do not want this dialog box to appear during debugging, set the button to off (default: on).
Ignore Image Color Differences	Disregard color any color differences that may be encountered.

Table 2: Debug Play Control Panel Entities (Continued)

Active Pause	<p>Allows processes under test to be operated during pause mode while recording a play session. Mouse and key interactions with the application under test will not be recorded.</p> <p>Use this option carefully, as it is possible that the nonrecorded interactions may substantially change the application state and result in invalidating a play session; for example, popping down a dialog box while in Active Pause could be dangerous because the script is unaware that the pop down has occurred. The inconsistent state could result in a false snapshot comparison failure. If the application state is changed, the original state should be restored before continuing record or playback.</p>
Play Button	Begin a play session.
Cancel Button	End testing.
Help	Access the on-line help information.

In the **Application Under Test:** field, type:

```
xmcalc
```

- In the **Play Script:** field, the name of the script already appears. Fill in the remaining fields with the appropriate names.
- Click on the **Play** button to begin the debug session.
The Replay Xcessory Debugger window appears:

Note: When the debugger is initialized, the calculator appears with the debugger window.

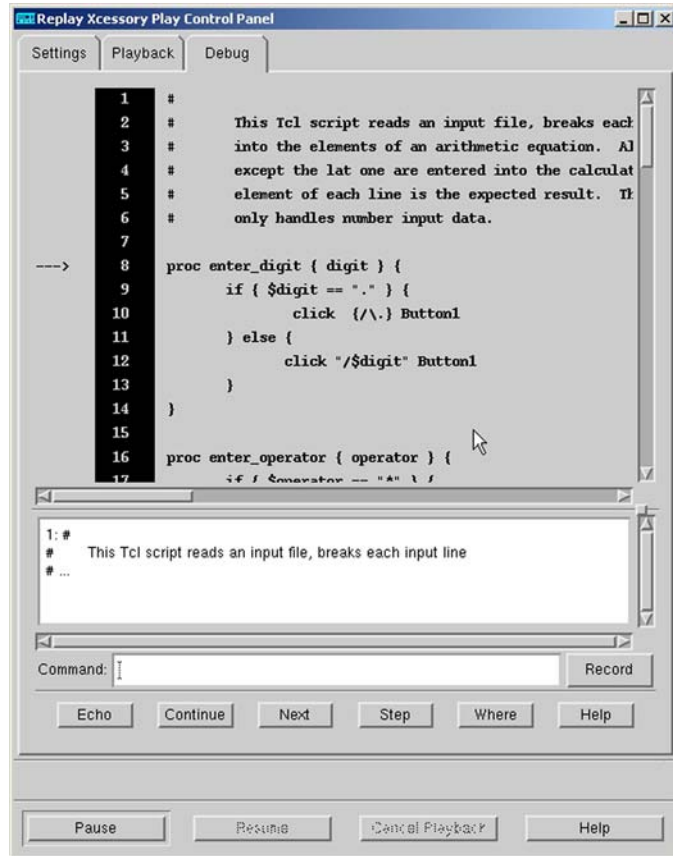


Figure 79 The Replay Xcessory Debugger Window

The top portion of the panel is the text window where actions within the script are displayed. You can view your script in a continuous stream, or line by line by using the buttons at the bottom of the window.

The **Command:** text field is used to enter command-line instructions. In addition to command options, breakpoints are set and manipulated by typing commands within this area. (See “*Setting Breakpoints with the b Command*” on page 249 for a discussion of breakpoints.) Any valid Tcl command can be entered in the **Command:** field.

When the **Record** button is sensitive, the user can press it to temporally switch the current session to record mode.

The Debugger page will change to the next view:

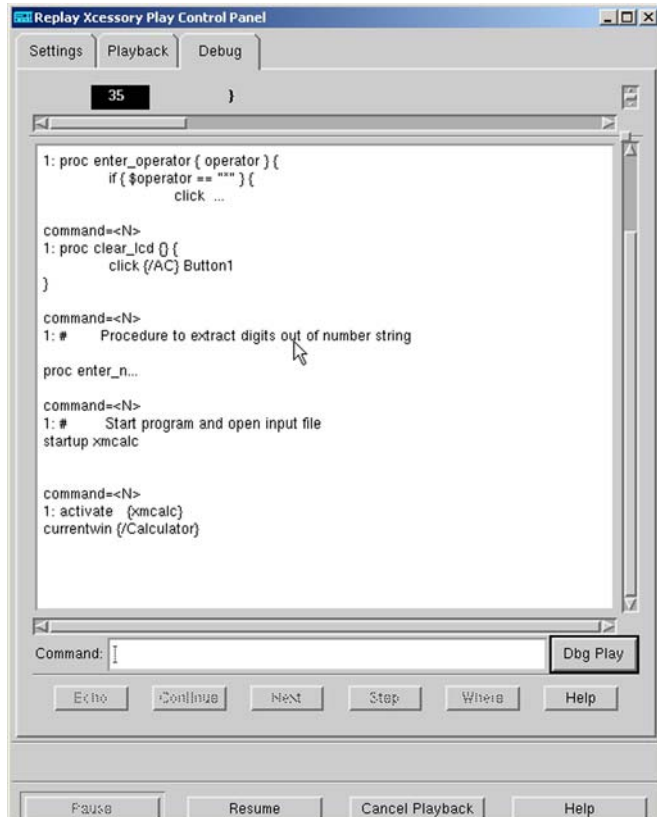


Figure 80 The Replay Xcessory Debugger Window Record View

All action on the application under test after that will be recorded to a temporary file. After the user presses this button again, a dialog will appear:

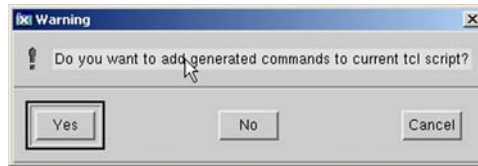


Figure 81 Ask Dialog

To save the updated code, the user selects “Yes”. The additional commands will be merged into the current tcl script and the current instruction pointer will be adjusted appropriately.

If “No” is selected, Replay’s window will change back to the usual window and the debug session will continue.

The “Cancel” button closes the dialog and allows the user to continue adding additional actions to be recorded on the application under test.

Note that the user cannot use this functionality when inside nested code bodies (i.e. inside procedures, “for”, and “while” loops).

The buttons at the bottom of the window control how the next debugger action or command is executed; these buttons are described in the following table.

Table 3: The Replay Xcessory Debugger Window Buttons

Echo	Print the value of a variable that was selected in the debugger window prior to pressing this button.
Continue	Continue to the next breakpoint or to the end of the script if no breakpoints are found. (The command-line equivalent is <code>c</code> .)
Next	Continue to the next statement. (The command-line equivalent is <code>N</code> .)

Table 3: The Replay Xcessory Debugger Window Buttons

Step	Execute the pending command and display the next command. (The command-line equivalent is <code>s</code> .)
Where	Display the execution stack. (The command-line equivalent is <code>w</code> .)
Help	Access the on-line help information.

These buttons correspond to command-line options that are described later in this chapter.

5. Select **Where**, **Next**, and **Step** in various combinations to familiarize yourself with the workings of these buttons.
6. Press **Continue** to run the rest of the script to completion.

Note: The cursor travels back and forth between the calculator and the text window as it plays back and displays the `xmcalc` application. You may pause the script, or view it line by line by using the buttons at the bottom of the window. The cursor can be quickly returned (or warped) to the debugger window by typing the cursor warp key, which defaults to `ctrl-w`. The user can also set the “Freeze pointer on commands” option on the Playback page of the settings dialog. In this case, Replay’s driver will execute commands as usual but will return the mouse pointer to the position where it was before the command execution. Note that in test cases where the mouse pointer position is important for the test context, this can affect to the result of the playback session.

The user can also use hotkeys for accessing the button callbacks in the debugger dialog.

To execute the “Next” command, the user can press the “Shift+n” key combination. It is no matter what application is the currently active Replay Xcessory driver or application under test.

To execute the “Step” command, “Shift+s” combination should be used.

Finally, to execute the “Continue” command, “Shift+c” will be used.

Note: You can change default hotkeys combinations on the playback page of the settings dialog which can be accessed through the Replay Xcessory Test Manager menu for test packages and using the “Override Test Package settings” button on the playback dialog for test cases.

The **replay** command cannot be run in the background if the command-line debugger is used.

Note: The Xnest pseudo server can be used to prevent the pointer from being grabbed on the display. The application under test will run on the Xnest display (as a window on the actual display); for example:

```
Xnest :1 &  
replay -tcd xncalc -appdisplay mach:1
```

See Chapter 9 for additional information.

Debugger Commands

The buttons on the Debugger Control Window can be used to easily and quickly perform debugging activities. The commands on the window correspond to command-line commands; the command-line usage allows for additional commands not available on the Debugger Control window.

Controlling Executions

This section describes the command-line equivalents to GUI options and button selections.

-s The **s (step)** command executes a pending command (the one just displayed by the debugger) and then displays the next command to be executed. If the command to be executed is a procedure call, the debugger executes the first command within the procedure. This command takes an optional argument to denote how many commands are to be executed.

[*num_of_cmds*]

- n** [num_of_cmds] A lowercase **n** (**next**) command is similar to the **s** command. Procedure calls, however, are treated differently. Procedures are executed as a unit, rather than executing just the first command within the procedure. Use this option to step into source commands.
- N** [num_of_cmds] An uppercase **N** is similar to the lowercase **n** command except that commands in files that are being sourced are treated as a unit.
- r** The **r** (**return**) command executes the rest of the commands in the current procedure as a unit and suspends execution. The debugger displays the command after the procedure call as the command to be executed next.
- c** The **c** (**continue**) command causes normal execution of the script to resume. Use **c**, for example, to resume execution after stopping at a breakpoint. Execution proceeds normally until it reaches a breakpoint or the end of the script (see “*Setting Breakpoints with the b Command*” on page 249).

Note: When using the GUI for debugging, Replay Xcessory prevents the debugger from stopping when the pointer is grabbed by a process. This prevents the pointer from becoming frozen; pointer grabs occur most often during menu selections.

Showing the Execution Stack with the **w** Command

The **w** (**where**) command displays the execution stack. The stack is displayed in several lines, each of which show the ending scope, for example:

```
0: application
1: the top-level scope of the application
```

the last line repeats the evaluation and the command that will be executed

Note: When **w** prints commands, those commands are displayed using the literal values of each parameter; when the debugger prints the next command to be printed, the command prints as it was originally entered in the script.

Controlling Scope with the *u* and *d* Commands

Use the **u** (up) and **d** (down) commands to move between scopes. **u** moves up; **d** moves down.

Both **u** and **d** accept arguments that represent the number of levels by which to move. For example, **u 3** moves up three levels in the script; **d 5** moves down 5 levels in the script.

Absolute scope can be specified by preceding the scope level with a pound sign, for example, **u #2**.

Controlling Output with the *-width* Option

The syntax for the **-width** option is as follows:

```
w -width [ width ]
```

where *width* indicates how many characters to print when outputting each logical line. This affects the length of the pending command as well as the result from the **w** (show stack) command.

If *width* is omitted, the current width is returned.

Setting Breakpoints with the *b* Command

A breakpoint provides a way to conditionally stop the execution of a command. These conditions include:

- expression testing
- matching command and argument name
- clearing a breakpoint
- listing a breakpoint

Breakpoints establish locations in the script or conditions that cause a halt to execution. A breakpoint can also specify a procedure to be executed when the breakpoint is triggered. The general form of the breakpoint command is:

```
b [ location ] [ condition ] [ action ]
```

The *location* specifier identifies the command at which execution should stop. The *condition* is an arbitrary Tcl expression that also causes execution to stop when the expression is true; *location* and *condition* can be used independently or together. The *action* is either a block or arbitrary Tcl statements that are executed when the breakpoint is triggered.

Specifying a Script Location

The *location* specifier identifies the breakpoints by specifying pattern matches on the statements themselves. The patterns can be specified using regular expressions similar to those used in the UNIX/Linux system C shell string matching, as described in the following.

*	Matches zero or more characters.
?	Matches any single character.
\	Removes any special significance of following character.
[]	Matches any single character listed in the enclosed sequence. A “-” between two characters indicates a range; for example: [abc0-9] matches the characters a , b , c , and any digit between 0 and 9 , inclusive.

The regular expression must be prefixed by **-glob** or **-g**, as in the following examples:

- To break on all source statements:

```
b -g "source *"
```
- To break at the definition for procedure **myproc** :

```
b -g "proc myproc *"
```
- To break whenever **myproc** is called:

```
b -g "*myproc *"
```
- To break whenever variable **mycount** is referenced:

```
b -g "mycount*"
```

An alternative syntax for specifying regular expressions similar to the UNIX/Linux system **grep** command is available. To use this alternate syntax, specify **-regexp** instead of **-glob** or **-g**. Descriptions for the matching values for the **-regexp** syntax can be found on the **regexp** Tcl reference manual page.

Using the Noop Command

Replay Xcessory provides a **NULL** command called **noop**, which takes a single argument. The **noop** command itself does not perform any function, but by inserting the **noop** command in strategic locations in the script using uniquely-named arguments, the arguments can be used as convenient labels for breakpoints; for example:

```
click {Apply}
noop before_cancel
click {Cancel}
b -g "before_cancel"
```

In this way, complex matches can be eliminated.

Specifying a Conditional Expression

Conditional expressions are specified using regular Tcl **if** statements, as in the following examples.

- To break anytime the expression is true anywhere in the script:


```
b if {$mycount > 0 && $increment > 1}
```
- To break whenever **myproc** is called if **mycount** is greater than 0:


```
b -g "mycount *" if {$mycount > 0}
```

Specifying a Breakpoint Action

An *action* is a block of Tcl statements which get executed when the breakpoint is triggered. Example:

- To print the value of **mycount** at a break for a call to **myproc**:


```
b -g "myproc *" then { echo "mycount = $mycount" }
```
- To print the value of **mycount** at a break for a condition:


```
b if {$mycount > 0} then { echo "mycount = $mycount" }
```
- To print the value of **mycount** at a break specified by location and condition:


```
b -g "myproc *" if {$mycount > 0}
then { echo "mycount = $mycount" }
```

Listing a Breakpoint

If no arguments are supplied to the **b** command, all breakpoints are listed. Each breakpoint is identified with a number. When multiple breakpoints occur on the same line, the actions are executed in the order that they are listed. To list a breakpoint, type:

```
b number
```

Examining Variables

Printing variable values is done by entering **echo**, followed by the variable, into the **Command:** field. Arrays can be printed using **parray**; widget resources can be examined using **getvalue**.

Clearing a Breakpoint

A breakpoint can be deleted by specifying the number of the breakpoint that you want to remove. All breakpoints can be removed by omitting a specific breakpoint number. To remove a specific breakpoint, type:

```
b -number
```

To remove all breakpoints, type:

```
b -
```

Other Commands

The following Tcl commands are useful in debugging:

info	Provides information about variables and procedures in the current script.
trace	Allows a procedure to be executed whenever a variable is accessed, modified, or reset.

Enhanced Tcl Debugger

The Replay Xcessory Tcl debugger has been enhanced to facilitate the setting and clearing of breakpoints. The debugger window now includes a script area in addition to the debugger output area, the command input area, and the debugger action buttons. The script area displays the current script together with line numbers.

Here is an illustration:

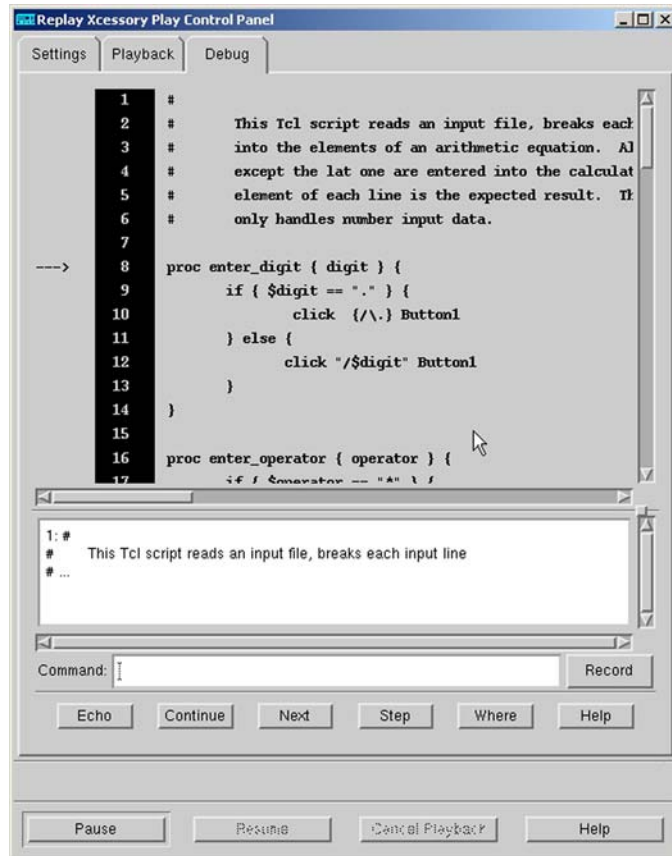


Figure 82 Replay Xcessory Tcl Debugger Showing Line Numbers

The arrow in the first column highlights the next line to be executed. Current breakpoints are indicated by the word **Break**, or **B--->** if the next line to be executed happens to be a breakpoint.

The easiest way to set or clear a breakpoint is to move the mouse pointer over the line where you want to set the breakpoint. Then use the right mouse button to select **Set Break** or **Clear Break** from the popup menu.

SCRIPT DEBUGGER

Enhanced Tcl Debugger

You can also set breakpoints by line numbers by entering a command of the form

```
b line_number
```

in the command area (example: **b 42**).

To view or to set or clear breakpoints outside of the current file, use the list command:

```
l file_name
```

Replay Xcessory displays *file_name* in the script area. You can then set or clear the breakpoint as described previously.

Alternately, use a command of the form:

```
b file_name:line_number
```

to specify a breakpoint in a different file without changing the current file in the script display area.

For additional information about the Tcl debugger, see “*Script Debugger*” on page 237.

Note: The **debug** and **xdebug** commands are no longer supported. All other debugger commands operate in the same way as before.

The Popup Menu

You have several additional action options on the main script area of the Tcl debugger. These actions are available through the popup menu. Right click and bring up the menu as shown:

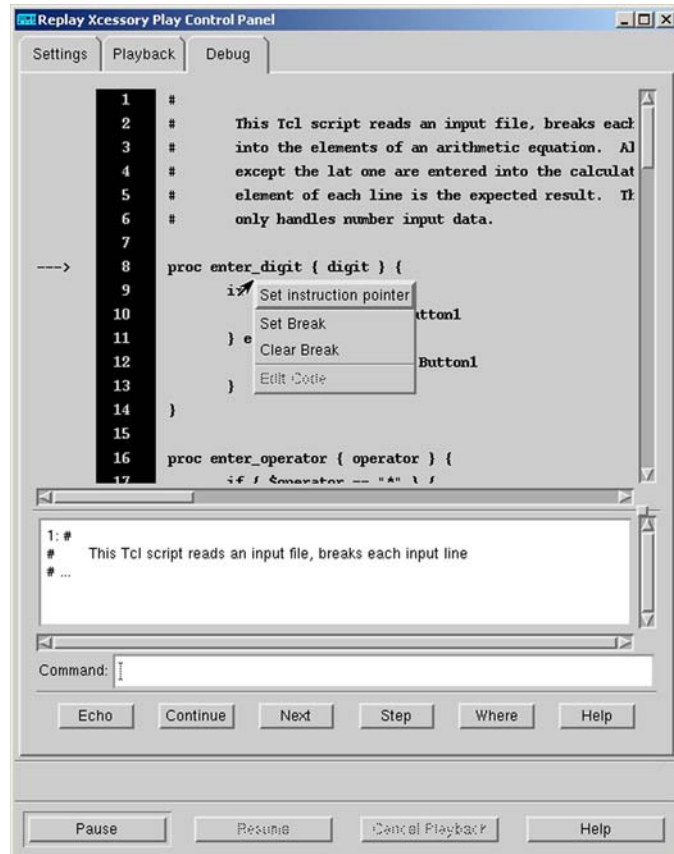


Figure 83 The Debugger Popup Menu

You can move the current instruction pointer when clicking on the first item of the popup menu (Set instruction pointer). Please note that there is one limitation of this functionality: you can change the current instruction only in the current scope, e.g. you cannot click on the lines that are not in the current procedure (or main tcl body) if you entered it during execution. To imagine

this situation in the figure above, you can change the current instruction only to line 16 (definition of procedure `enter_operator`) or further similar instructions. In other words, you cannot force the Tcl interpreter to change the current stack.

Also, you can change the Tcl code without exiting from the Tcl debugger. To do this, you should select the region of the code that you wish to edit. The region can be selected by clicking on the left mouse button while pressing the Ctrl key. The first click will highlight one line and show, on the status bar message, what line was selected.

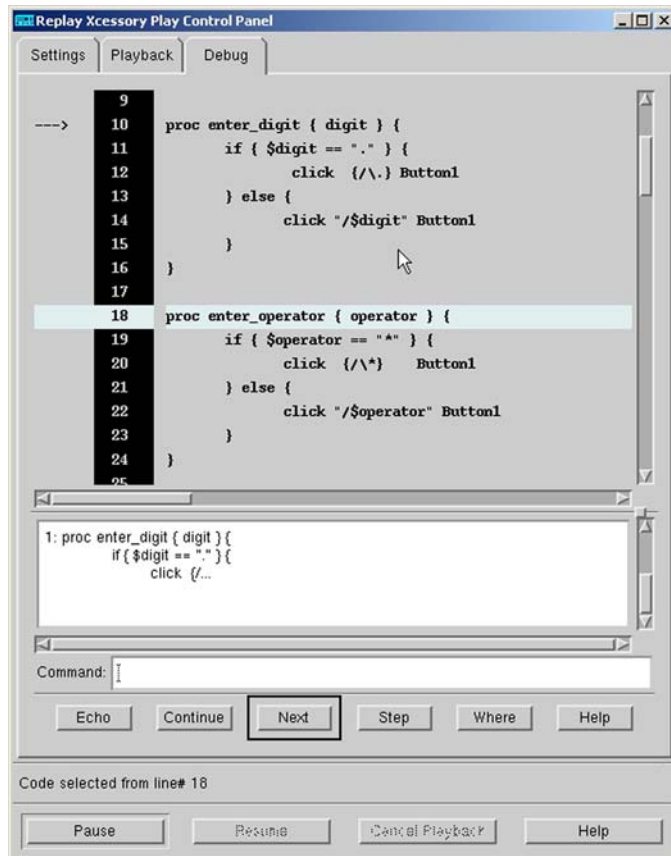


Figure 84 Changing one line of Tcl Code

After a second mouse click, a multiline area will be highlighted:

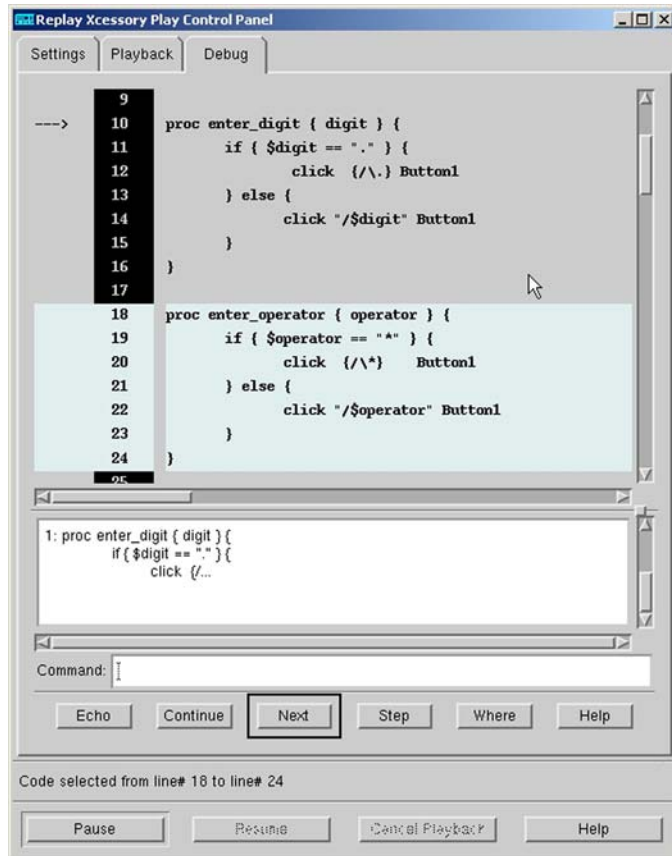


Figure 85 Changing Multiple Lines of Tcl Code

After a region is selected, right click on it and select “Edit Code” from the popup menu.

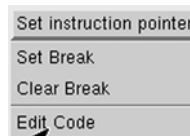


Figure 86 Popup Menu

This will bring up Tcl editor:

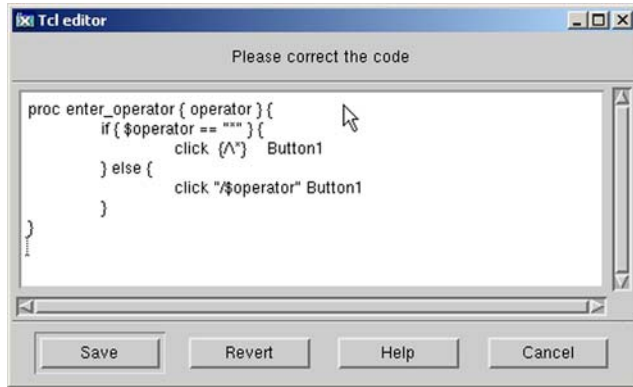


Figure 87 The Tcl Editor

Using the Tcl editor, the user will be able to make some changes in the code.

Note that new tcl code will be saved as actual script for the current test case and previous version will be named as `old_testcase_name_backup.tcl`.

- | | |
|---------------|---|
| Save | After pressing this button all your changes will be saved in the current scriptor. Be careful with tcl syntax; the editor has the responsibility for checking the code. |
| Revert | Undo all current changes and shows the code that existed at the beginning of the edit operation. |
| Help | The Help dialog. |
| Cancel | Closes the dialog without saving any changes. |

If you change code that is higher than the current instruction pointer it will be executed only during the next session. If you changed the body of some procedure, you may move the current instruction pointer to the definition of this procedure and re-execute it again to make sure that the tcl interpreter will use the new code that you provided.

There are currently several limitations for using the editing functionality. You cannot edit the code of the instruction that will be executed next (usually this is a line where the current instruction pointer is set), and you should not edit the body of the nested pieces of code when you are. Nested code is the procedure body and body of the “while” and “for” commands.

Debugging Using the Command-Line

In some cases, debugging an X application from the same display in which the application is running is inadvisable because the cursor must be moved between the debugger window and the process window. When you are proceeding in this manner, the pointer can get “grabbed” which prevents the user from entering debugging commands. To ensure that the cursor is no longer moving between the debugger and the process, execute the replay debugger from another terminal, X display, or use Xnest (see the note in “*The replay command cannot be run in the background if the command-line debugger is used.*” on page 247 for additional information).

Commands can be issued using the command-line, or by using the GUI provided with the Replay Xcessory software. The following command and options are used to debug scripts from the command-line:

```
replay -debug [ -l script ]  
               [ -s snapshot_directory ]  
               [ -R report_file ]  
               [ -B baseline_dir ]  
               [ app_name app_args ]  
               [ -tcd tcdname ]
```

Command-Line Options

The options for the **replay** command are as follows:

- | | |
|-------------------------------------|---|
| -l <i>script</i> | Name of the play or record script file. |
| -s <i>snapshot_directory</i> | Name of the directory that will be used to collect current snapshots during a record or play session. |
| -R <i>diff_cmd_file</i> | The name of the file to which all differences and test stats will be dumped. |

SCRIPT DEBUGGER*Debugging Using the Command-Line***-B** *baseline_dir*

The name of the baseline directory to which all widget tree and screen dumps will be placed.

For additional information about command-line instructions and options, refer to Chapter 4 and to the **replay(1)** and **replaytm(1)** reference manual pages.

Advanced Topics

Overview

This chapter covers miscellaneous topics that are generally not needed for everyday usage of Replay Xcessory.

Using Terminal Emulators

Replay Xcessory provides a specially-instrumented version of **xterm** for record and playback sessions. This functionality makes it possible to synchronize text input and output, as well as extract the visible area of displayed text. Synchronization usually occurs automatically, depending on the state of the widget that is to receive input. This approach is insufficient for **xterm** synchronization because the widget always appears ready for input, even if the invoked application process is not.

To bypass this inadequate synchronization, Replay Xcessory provides the **termsync** Tcl command. **termsync** accepts an argument that specifies one or more strings that Replay Xcessory recognizes as an indication that input can be accepted. This string recognition is usually in the form of a prompt, for example:

```
termsync {host1>:host2$:%}
```

The sync string can be separated with colons to indicate multiple prompts. The sync string can be initialized by setting the **REPLAY_TERM_SYNC** environment variable. Use **termsync** whenever the sync string needs to change.

Note: If the process that is running in the **xterm** window does not have a reliable prompt string with which to synchronize, do *not* exceed normal speed when you play back the session.

Ensure that the Replay Xcessory version of **xterm** is invoked by placing **\$REPLAYHOME/bin** in your path *after* the */path_to/replay/bin* and *before* the directory that contains the standard **xterm** executable. Use the UNIX/Linux system shell **which** command to confirm that the correct version of **xterm** is being executed.

The **visibleText** resource is used to obtain the strings located in the visible portion of the **xterm** window. Strings are directly obtained using the **getvalue** command, or by using logical snapshots. Use of the **visibleText** resource provides an efficient way to verify correctness, as opposed to comparing **xterm** image differences. The name of widget containing the string is **vt100**.

Example using logical snapshots:

```
currentwin /xterm
set value [ getvalue {vt100} {visibleText} ]
if { $value == "abc" } {
```



```
        pass
    } else {
        fail
    }
}
```

Example using `getvalue`:

```
*visibleText.vrSave: True
```

Using VNC, Nested and Virtual X Servers

Additional information on Xnest can be found in the man pages at:
<http://www.xfree86.org/4.2.0/Xnest.1.html>

Currently, Replay Xcessory supports the fully automatic use of Xnest Xvfb and Xvnc virtual X servers. Users can make the appropriate selection on the first page of the playback dialog. This option is saved to the `tcd` file and saved across sessions.

Three new servers are included in Replay Xcessory:

- **VNC server**
- **Nested X server**
- **Virtual X server**

These servers provide additional flexibility when testing applications.

VNC

VNC (Virtual Network Computing) is a platform-independent system for remote desktop control. **VNC** gives an opportunity to connect to a remote machine desktop and work with it as if it is the desktop of a local machine. A **VNC** system consists of a client, a server, and a communication protocol.

- The **VNC** server is the program on the machine that shares its screen.
- The **VNC** client (or viewer) is the program that watches and interacts with the server.
- The **VNC** protocol (RFB)

Note that on some machines, the server does not necessarily have to have a physical display. **XVNC** is the Unix **VNC** server, which is based on a standard X server. **Xvnc** can be considered to be two servers in one; to applications it is an X server, which is based on a standard X server. **Xvnc** can be considered to

be two servers in one; to applications it is an X server, and to remote **VNC** users it is a **VNC** server. Applications can display themselves on **Xvnc** as if it were a normal X display, but they will appear on any connected **VNC** viewers rather than on a physical screen.

In addition, the display that is served by **VNC** is not necessarily the same display seen by a user on the server. On Unix/Linux computers that support multiple simultaneous X11 sessions, **VNC** may be set to serve a particular existing X11 session, or to start one of its own. It is also possible to run multiple **VNC** sessions from the same computer.

The benefits of **VNC** are the following:

- Tests can be run on a local or remote machine without grabbing the local display.
- Multiple tests can run in parallel on different **VNC** displays.
- Tests can be managed and viewed from any remote or local machine. Since **VNC** clients (and servers) are available for almost all operating systems, tests can be managed from Windows or MacOS hosts and run on a machine having no physical display.
- On a Unix machine, a **VNC** server doesn't need a real X server to run.
- It's easy to share **VNC**, so more than one screen can view the same workspace at a time.
- Tests can be run on a pseudo screen which may be larger or smaller than the actual screen.

VNC benefits make it the best solution for using with Replay, even on a local machine.

Here are the steps for starting a **VNC** server:

1. Edit `~/vnc/xstartup` file. It's a shell script started by **Xvnc**. It should include command for starting the desirable window manager. Example:

```
#!/bin/sh
#file ~/vnc/xstartup
xrdb $HOME/.Xresources
xterm &
```

```
#start MWM window manager  
mwm &
```

2. Set VNC server password:

```
vncpasswd
```

It will ask to set new VNC password. Password will be prompted when establishing connection via VNC client to this server,

3. Start VNC server:

```
vncserver :1 -alwaysshared &
```

The :1 indicates display number to be used by clients to connect to server. -alwaysshared option indicates that server allows simultaneous connections of many VNC clients.

To connect to the VNC server, do the following:

```
vncviewer host:1 &
```

Parameter 'host' indicates the machine where the server is being run. For local machine use 'localhost' as host parameter.

:1 indicates the display which is used by the VNC server to connect to.

Password will be asked. If valid password is given, the window showing the remote desktop will be opened.

Replay Xcessory can now be started inside virtual server and used in the same way as it would on real X display. For running multiple test sessions in parallel, each session should be run in separate VNC server.

See **Xvnc(1)**, **vncserver(1)** and **vncviewer(1)** for complete information on using these tools.

Nested X Server

Xnest is a pseudo server located between the application under test and the actual server. The contents of the pseudo display are depicted as a window in the real display. The nested server provides several benefits:

- Tests can be run without monopolizing the display; the keyboard and mouse can still be used to interact with other processes.
- Multiple tests can be executed in parallel using a single display.
- Tests can be run on a pseudo screen which may be larger or smaller than the actual screen.

Keyboard and pointer control parameters modified by **Xnest** are not reset after **Xnest** terminates. These parameters can be reset using the **xset** command; for example:

```
xset -r
```

restores the auto repeat feature normally disabled by **Xnest**.

Starting a Nested Server

Here are the steps for starting a nested server:

1. Start the nested server by typing:

```
Xnest :1 &
```

The **:1** indicates that the nested server is to be addressed as **:1** by applications which want to connect to it. Color flashing may occur as the pointer is moved in and out of the nested server. This flashing is unavoidable as the nested server has its own color map.

2. Start an **xterm** window and a window manager to run inside the nested server:

```
xterm -display :1 &
```

3. From within the newly-created xterm, type:

```
mwm -display :1 &
```

Replay Xcessory tests can now be started inside the nested server. If you are running multiple test sessions in parallel, each test session should be run in its own nested server. Each nested server can be resized or iconified. Virtual desktop environments, such as **KDE** or **GNOME**, are also useful for managing the screen surface area on the real display, placing each nested server in a different area or “room.” Refer to the **Xnest(1)** reference manual page for complete information about using this command.

Virtual X Server

Additional information on Xnest can be found in the man pages at:

<http://www.xfree86.org/4.2.0/Xnest.1.html>

Xvfb (virtual frame buffer) is a virtual server. This virtual server appears to be a real server to the application, but does not interactively display any results. Thus it is only useful for background tests running. Interaction with application running on Xvfb server is impossible. The virtual X server provides the same functions as the nested server, and includes the following:

- Unlike **Xnest**, an actual server does not have to be running.
- Processes can be tested against different screen dimensions and depths, without the need for investing in additional hardware.

Starting a Virtual Server

Here are the steps for starting a virtual server:

1. Start the virtual server by typing:

```
Xvfb :1 &
```

The **:1** indicates that the virtual server is to be addressed as **:1** by applications which want to connect to it.

Other useful parameters are:

-screen screennum WxHxD

This option creates screen screennum and sets its width, height, and depth to W, H, and D respectively.

-pixdepths list-of-depths

This option specifies a list of pixmap depths that the server should support in addition to the depths implied by the supported screens. **list-of-depths** is a space-separated list of integers that can have values from 1 to 32.

2. Start a window manager and execute **Replay Xcessory** tests in batch mode. If multiple sessions are to be run in parallel, each test session should be run in its own virtual server.

To view the static image of a virtual display, use the **-fbdir** *directory_name* option to capture the contents of the virtual display and then use **xwud** to view.

Note: **Xnest** is easier to use than **Xvfb**; in a situation where either server can be used, it is recommended that you use **Xnest**.

Note: The **Xvfb** default keyboard map may not match the keyboard map of the real server on which the application is intended to run. Since missing keys can cause scripts to fail, we recommend that you ensure all necessary keys are defined before you start a play session.

You can display the current keyboard map using the following command:

```
xmodmap -pk -display display
```

You can add additional key definitions to a file; be sure to use unused keycode numbers for the definitions you add. Here are some examples:

```
keycode 91 = BackSpace
```

```
keycode 92 = Home
```

```
keycode 93 = End
```

To load a key definition file, use the following command:

```
xmodmap -display display key_definition_file
```

Refer to the **Xvfb(1)** reference manual page for complete information about using this command.

Monitoring Background Tests

The **servwatch** utility allows you to monitor any server to which you have permission to connect, including virtual servers. The virtual server is a useful tool for running Replay Xcessory scripts without monopolizing a real server, or for running tests with different characteristics from the available hardware. The server being monitored is displayed as a window on your display. The refresh interval can be adjusted from its default of one second.

Note: Server monitoring is a fairly CPU and network intensive operation. Hence, we recommend that you limit server monitoring to short time periods, or keep the refresh interval to a high number, such as every ten seconds instead of every second.

Using Replay Xcessory with Source Debuggers

Replay Xcessory can be used with source debuggers. To use debugging, place the debugger command line where the application command line would normally be; for example, to use the debugger with a play session, enter:

```
replay -tcd xmalloc -- gdb
/opt/replay/examples/bin/xmalloc
```

To use the debugger with a record session, enter:

```
replay -r -tcd xmalloc -- gdb
/opt/replay/examples/bin/xmalloc
```

Source debuggers can also be used in conjunction with the Replay Xcessory Tcl debugger during play sessions. The concurrent use of both debuggers provides parallel views of the application state and the Tcl script.

Using Custom Widgets

Replay Xcessory supports recording and playing back applications which use custom widgets without additional preparation needed. This support is possible because the Replay Xcessory instrumentation is limited to the X Toolkit and does not rely on widget-specific information. If the custom widget also includes custom resource types that need to be dumped to the widget snapshots or returned in **getvalue** commands, custom type converters need to be written. The following sections describe type customization.

Using Type Converters

Replay Xcessory converts widget resources from the internal format to ASCII strings in two cases:

- when creating a widget snapshot
- in response to **getvalue** commands

The resource type conversion is performed by employing the standard mechanism used by the X Toolkit.

X Toolkit type Converters

Each widget resource belongs to a specified type that identifies how that resource is internally represented. For example, **XtRInt** indicates a resource type that is an integer, while **XtRString** indicates a character pointer. Types can be arbitrarily complex, as in the case of **XtRFontStruct** which is a pointer to a complex structure of type **XFontStruct**.

Type converters make it possible to register functions that convert from one resource type to another. Most existing converters take an input of type **XtRString** (character strings) and convert this type to the one of an internal type. This conversion takes place because the most common usage of converters is in taking specification in **.Xdefaults** or another resource file.

The X Toolkit provides converters that convert strings to **XtRShort**, **XtRInt**, **XtRCardinal**, **XtRBoolean**, **XtRDimension**, and other types defined in the X Toolkit base classes. Widget sets, such as Motif, register their own converters from **XtRString** to resource types defined by the widget set.

Replay Xcessory Type Converters

Replay Xcessory conversion, however, goes in the opposite direction: from internal types to character strings. Replay Xcessory also defines the resource type **VrRString** (“VrString”). Replay Xcessory always attempts to convert to type **VrRString** before converting to **XtRString**, making it possible to write a converter to string which may need to be different from the regular **XtRString** type.

Replay Xcessory provides default resource converters for most resource types defined by Motif and the X Toolkit base widgets. If no converter to **VrRString** or **XtRString** is found, the behavior depends on whether that converter was needed to generate a widget snapshot or to respond to a **getvalue** query. If needed for a snapshot, the snapshot entry is output as a comment—prefixed by an exclamation point—and the resource type will be listed in parentheses rather than the resource value; for example:

```
!*mylabel.lfont: (XtRFontStruct)
```

Alternatively, if an appropriate converter to string is not found for a **getvalue** command, the resource value is converted as if it was a number.

Note: This default may be inappropriate for resources that are of pointer types.

Alternate String Formats

In addition to string formats defined by “VrString” and “String,” any number of string formats can be defined for each resource type. Conversion to the alternate formats can be obtained using the **getvalue** command. The name of the alternate type must be specified as the last argument. Replay Xcessory provides a default converter that changes a widget ID to its qualified name. Replay Xcessory also provides an alternate converter from **XtRWidget** to “Wid” which returns the widget ID as a hexadecimal string.

Extracting Application Data

Custom converters can be used to extract application data. For example, a converter can be used to extract application data structures or internal widget data. Data is extracted by registering a converter with any generic **from_type**. This corresponding **from_type** is then used with the Tcl **getvalue** command. The following example illustrates how the address of a widget core field can be retrieved.


```

#include <X11/Intrinsic.h
#include <X11/IntrinsicP.h

static void _FreeInf(char* str)
{
    XtFree(str);
}
static String ExtractWidgetInfo(Widget w)
{
    String str = XtMalloc(100);
    sprintf(str,"The widget core address
is %x", w->core);
    return str;
}
static String _WidgetInfoToString (Widget w, String \
resource_name)
{
    String ToVal;
    ToVal = ExtractWidgetInfo(w);
    return ToVal;
}
VrRegisterTypeConverters()
{
    VrSetTypeConverter("widgetInfoText",
        _WidgetInfoToString,
        _FreeInfo);
}

```

The following **getvalue** command accesses the **_WidgetInfoToString** type converter:

```
echo [ getvalue {*text} {widgetInfoText} ]
```

The **tcl.h** file must be included and is distributed in the Replay Xcessory **/lib/replay.src** directory. The Tcl library must be included when relinking **replay**, and is also distributed in the Replay Xcessory **/lib/replay.src** directory. An extended Tcl library may be used in its place.

Tcl commands are created by calling `Tcl_CreateCommand()`. The following example shows how a new Tcl function named “mytime” is added to the Replay Xcessory Tcl command set. This example may be found in `$REPLAYHOME/lib/replay.src/cuscmds.c`.

```

/*****
/* file name: cuscmds.c */
#include <time.h>
#include <stdio.h>
#include <tcl.h>

static int my_time(int clientData, Tcl_Interp* interp,
int \ argc, char** argv)
{
long tloc;
static char string[50];
time(&tloc);
sprintf(string, "This time is: %s",
ctime(&tloc));

Tcl_SetResult(interp, string, TCL_STATIC);
return (TCL_OK);
}

/*****
* Register your Tcl commands here.*
*****/

void VrRegisterCusCommands(Tcl_Interp *interp)
{
Tcl_CreateCommand(interp, "mytime"
(Tcl_CmdProc*)my_time, (ClientData)0,
NULL);
/*****/

```

After registering the commands, the replay binary must be relinked. You can find the `Makefile.replay` for linking in the `$REPLAYHOME/lib/replay.src/`

The X Toolkit library, **libXt.a**, must be statically linked to ensure that it links with the Replay Xcessory X Toolkit library.

To build a new **replay** binary:

1. Compile **cuscmd.c** with the new custom Tcl command added.
2. Change to the `$REPLAYHOME/lib/replay.src` to directory.

```
make -f makefile.replay
```

Environment Variables That Can Be Used with Replay Xcessory

PATH	Will look at the AUT at the path at the start of the session if the full path is not specified.
TCL_LIBRARY	Will use the specified tcl library for the embedded Tcl interpreter.
REPLAYHOME	Evaluates automatically if not specified - points to the Replay Xcessory install dir.
LD_LIBRARY_PATH	Needed for binaries linked against shared libraries. Have to contain the path to the customized lib Xt (REPLAYHOME/lib/Xt) for the binaries which the user intends to use with Replay using "connect" command.
SHLIB_PATH	For HP, see LD_LIBRARY_PATH above
LIBPATH	For IBM, see LD_LIBRARY_PATH above
DISPLAY	The desired display name.
RX_NATIVE_XT	Disables extra event processing in the customized Xt library (for programs which are sensitive to the speed of Xt).
REPLAY_DEBUG_LEVEL	Manages how much information must be dumped into the log.
DRIVER_DEBUG_ON	Switches debugging of the Replay Xcessory driver.
XTLIB_DEBUG_ON	Switches debugging of the customized Xt library.
REPLAY_DDEBUG_ON	Switches detailed debugging on.
ONEFILE	Forces all logs to be dumped in one log file - /tmp/xtliblog for the customized libXt and /tmp/driverlog for the driver itself.
DUMP_WIDGET_NAMES	If set, this dumps all the widget names to the standard output (stdout) during the widget processing on the logical (widget)snapshots. It can help in case of a custom type converting procedure of a particular resource type that seems to be working incorrectly.
REPLAY_PROPS_DIR	Properties for the session ().
REPLAY_TESTSUITE_PROPS	Properties for the current testsuite (for the applicable order of properties see in chapter 4).

RX_EXCLUDE_SH Indicates an application shell class that must be explicitly ignored by Replay Xcessory when it scans client shells and stores them in the internal list for future reference. This can be applied when the application has two toplevel shells and only one of them is the correct ancestor of the widget tree. If Replay selects the one that is wrong, we can resolve this by setting the variable with the name of the wrong shell class.

Using Additional Libraries with Tcl: Itcl

The user can use any additional shared libraries which provide extended Tcl commands. Itcl is a well known tcl extension which provides availability to use objects in Tcl.

To use Itcl, it must be properly installed on your system. There has to be a specified env variable `ITCL_LIBRARY` on the shell before invoking Replay, and lines for loading the tcl package or shared library must be the first lines of tcl.

You can find an example of itcl script in `REPLAYHOME/lib/tcl/custom/itcl_ex.tcl`. All other scripts can be used as with general tcl script.

Automating Application with Data from RDBMS

This section describes how text widgets in an application can be automatically filled with data from a database. The Sybase database is taken as an example database system.

First of all, we need several packages installed on a system to have access to our DB from TCL. These are UnixODBC, FreeTDS (for Sybase), tclodbc.

1) Unix ODBC installs an ODBC driver on a system.

<http://www.unixodbc.org/>

Download it, compile and install. If you do not have the Qt library on your system, use option `--desable-gui` for the configure script. With this option, you will not have `ODBConfig` and `DataManager` graphical utils. First, it is a handy util for configuring ODBC, but you will be able to configure ODBC from the console.

Install and do not configure it. First install the next package.

2) Install Free TDS. It is a TDS (Sybase RDBM protocol) driver.

<http://www.freetds.org/>
 use “configure-with-tdsver=5.0

Note: Sybase is used here only as an example database system.

3) Next, set up your ODBC. Use the ‘odbcinst’ console util for this. Create a new DSN and select FreeTDS driver. Read the manual on unixodbc site.

4) Check ODBC. Use the isql console util shipped with unixODBC to test the connection to your database server.

```
#man isql
```

5) Install TclODBC

<http://sourceforge.net/projects/tclodbc/>

It is installed simply.

It will install libtclodbc2.x.so and several tcl scripts somewhere in your path. You must know its path. Normally it is installed in /usr/lib/tclodbc2.x or /usr/local/lib/tclodbc2.x

After setting up unixODBC you should know your DCN - Database Source Name. Set up dbsubs.tcl script to fit your environment. It is located in \$REPLAYHOME/lib/db. Open it in your text editor. In the header of the script, edit the following variables:

TCL_ODC_PATH - Path to tcl odbc files

DSN - Your DSN

USER - Sybase user name

PASSWORD - Sybase user password

For example:

```
set TCL_ODBC_PATH "/usr/lib/tclodbc2.5"
```

```
set DSN "My_DSN"
```

```
set USER "Nick"
```

```
set PASSWORD "MyPassword"
```

Test your connection with testsql script. For example:

```
$ testsql"select name from sysobjects where
type='U' "
```

Its first parameter must be a SQL string. This TCL script connects to your database, sends this SQL string to it and prints reply from DB. In the above example, it would print all tables in your database.

If it prints valid data, then the connection works well.

If it prints an error message, then tclODBC has the wrong setup or you have a wrong path to tcl odbc. Check unixODBC with 'isql' util. TclODBC can be checked with this script.

Now you can play with dbsubs.tcl script.

Start a new record. Make corresponding steps in your application to open a form that should be automatically filled with data from the database.

When you see the widgets that must be filled automatically, do the following:

- Click the 'Learn Tag' button. A new dialog window will appear. In this window, you will assign tags to widgets that must be filled.
- Click on the 'Identify' button. The mouse pointer will change.
- Click on your widget. In this dialog you will see a minimized name of the widget and a full internal name. These are internal Replay names of AUT widgets. They are generated automatically by Replay.
- In the field 'Assigned Tag,' name your widget as you prefer. This name can be used in RX as {\name}.

For example, click {\my_button} Button 1 :0 100

- Click the 'Apply' button. This applies the tag to the widget.

Assign tags to all your widgets as you prefer and save these names somewhere temporarily. When all necessary widgets are named with tags, click the 'Cancel' button.

Click the 'Append' button. The 'Comment Entry' dialog will appear.

Activate the 'Command Mode' toggle button.

Type the following script:

```
#- DB AUTOFILL Script Start-----
#PATH TO dbsubs.tcl. Usually $REPLAY
HOME/lib/dbsubs.tcl
```

```

set DBSUBS_FILE "/path/to/dbsubs.tcl"

#TABLE OF WIDGET NAMES (/TAGS)
#tags (and only tags) must begin with '/'!!!
set TBABLE {
    {/last_name/first_name/phone}
    {/last_name2/first_name2/phone2}
    {/last_name3/first_name3/phone3}
}

#SQL STRING
set SQL "select au_lname, au_fname, phone from
authors where state='CA'"

source $DBSUBS_FILE

fill_table $TABLE $SQL

#-SCRIPT END-----

```

This template is shipped in the file `$REPLAYHOME/lib/db/template.tcl`.

It contains three variables that must be modified:

`DBSUBS_FILE` - Path to `dbsubs.tcl` file

`TABLE` - Two-dimensional list of widget tags (or names) that you assigned in the previous step

`SQL` - sql string

Click the 'OK' button.

Continue record.

You should know that you can use just widget names instead of tags. But tags are more flexible. Additionally, you can automatically generate tags for all your widgets.

It is recommended to use the Tag Manager component from `replaytm` or `rtm`. It is an instrument for working with widgets.

It is recommended to test your SQL string with the `testsql` program before inserting it in the script.

Index

Symbols

173
 \$ 27, 174
 \$REPLAY_PROPS_DIR 102
 @x,y coordinates 230

A

activate 214
 Active Pause 110, 220
 adding comments to a script 173
 additional reading
 tcl 170
 adjusting play speed 17
 slider 17
 alias command 207
 Allow Wrong Geometry 116
 alternate menu selection commands 226
 alternate string formats 270
 app-defaults 123
 appDisplay 219
 append command 176
 application
 display 111
 life cycle 3
 retesting 42
 verification points 4
 application management commands
 activate 214
 application shells
 multiple 39
 array 172
 built-in 172
 ASCII
 comparisons 42
 editor 41
 files 63
 resources 42
 strings 269
 values 42

asterisk 225
 atomic transaction 40
 automated tests
 developing 2, 269
 automatic widget synchronization 44

B

b command 254
 backslash
 substitution 173, 174, 175
 baseline file 61
 baseline review area 61
 baseline snapshot 5, 42
 directory 54
 view pane 62
 baselineDir 220
 bash_profile 6
 batch creation 60
 batch mode 6, 32, 161
 bitmaps 3
 bitwise operators 179
 boolean resource 125
 braces 175
 breakpoint 243, 249, 253
 action 251
 clearing 252
 deletion 252
 listing 251
 removing 252
 setting 243, 249
 specifying actions 251
 .bsl extension
 See also baseline snapshot directory 54
 built-in array 172
 button
 Cancel 12, 14
 Comment 12
 Dump Hits 12
 Help 12
 Launch 32
 Learn Tag
 See also learnTagKey 140

Index

- Library Path 110
- Record 11, 12
- Snapshot 5, 12, 13
- Source 12
- Stop 14
- Test Case 12, 13

button state 200
Button-Up-Motion 198

C

- calculator program 7
- calculator test driver 22
- calculator test driver script 23
- changing the test suite directory 51
- children list 39
- click 4
- click commands
 - See also mouse commands 198*
- Click Offset Tolerance 115
- Client Startup Timeout 111
- clientPid 220
- clients
 - multiple, dynamically-linked 99
- clientTimeout
 - See also Client Startup Timeout 220*
- closewin command 202
- color flashing 266
- command
 - actions and objects 35
 - activate 214
 - append 176
 - click 44
 - close 63
 - compress 110
 - debugger 247
 - delay 35, 39
 - diff(1) 43
 - echo 215
 - echoreport 215
 - exec 44, 218
 - expr 172
 - extended 198

- extended Tcl 196
- file access 193
- getproperty 213
- getvalue 43, 44, 270
- glob 194
- global 185
- incr 176
- keyboard 201
- ldd 100
- list 187, 230
- main menu 58
- menu selection 224
- message 203
- mouse 198
- narrative 215
- objects 36
 - See also widget names 35*
- pass/fail 204
- press 44
- proc 184
- procedure 187
- Record/Play 61
- release 44
- replay 32
- resize 40
- return 185
- scale 229
- script actions 35
- scripts 35
- scroll bar 227
- set 171, 172, 176
- setvalue 43
- string manipulation 191
- substitution 173
- system 44
- termsync 262
- test case management 204
- test management 198
- testcase 33
- text 40
- text field 243
- uncompress 110

- user interaction 198
- whatlib 49
- widget information 198
- widgetname 207
- widgettag 207
- window management 202
- Xnest 266
- xwddiff(1) 43
- command argument 174
- command delay 39
- command line
 - debugging 259
 - interface 3, 6, 238
 - options 259
- command substitution 174
- comment entries 124
- comments 173
- comparing snapshots programmatically 43
- component
 - names 39
 - null 39
- compress command 110
- Compress Image Command 110, 221
- Compress Image Snapshots 110, 220
- compress motion 114
- condition Tcl expression 250
- conducting a play session 152
- conducting a record session 143
- Connect Application 115
- connected applications 203
- Context Sensitive Lists 114
- control file 5
- control panel
 - play 15
- controlling debugger executions 247
- controls
 - record 134
- coordinates 149
 - @x,y 230
 - GUI 3
- creating a new test package 7, 51, 53
- creating lists 175

- .cshrc 6, 49
- current active client
 - displaying 32
- currentwin 207
- cursor
 - grabbing 111, 259
- customizing utility commands 51

D

- data types
 - strings 171
- dblclick 4
- debug 254
- debug play control panel 241
- DEBUG_SLOTS 113
- debugger commands 247
- debugger interfaces 238
- Debugger Warp Back Hot Key 118
- debuggerWarpBackKey
 - See also Debugger Warp Back Hot Key 222*
- deconify command 202
- Default Application 115
- default command
 - to bring up an edit window 56
- default properties
 - record and play 108
- defaultDelayTime 149
 - See also Delay Time 222*
- delay command 214
- Delay Time 117
- delays 149
- Deleting 9
- deleting a Test Package 9
- dialog boxes 3
- diff 43
- Diff Command 118
- direct comparison of the widget contents 43
- directory
 - script 6
 - snapshot 6
 - test package 6
- directory listing display 51

Index

- Display Image Differences 116, 221
- double quotes 175
- drag 4
- DRIVER_DEBUG_ON 111
- DUMP_TAG_INFO 112
- DUMP_WIDGET_NAMES 112
- dynamic dependencies
 - checking 100
 - listing 100
- dynamically-linked multiple clients 99
- E**
 - echo command 215
 - echoreport command 215
 - editability of recorded scripts 35
 - editres 36
 - embedded spaces 174
 - env variable 172
 - environment variable 48
 - env 172
 - for the tutorial 6
 - LD_RUN_PATH 99
 - library path 31, 98, 100
 - MANPATH 49
 - REPLAY_TESTSUITE_PROPS 102, 123
 - setting 6
 - errors 21
 - event synchronization 44
 - excludeApps 101
 - exec command 44, 218
 - execution stack 248
 - Exit On Snapshot Mismatch 116
 - exitOnDiffMismatch
 - See also Exit On Snapshot Mismatch 221*
 - explicit pass/fail 204
 - expression evaluation
 - Tcl 172
 - expressions 177
 - extended commands 36, 198, 223, 224
 - extended Tcl commands 196
 - External diff program 118

- F**
 - fail command 205
 - features
 - Test Manager 33, 48
 - \$file 57
 - file
 - .cshrc 6, 49
 - .profile 6, 49
 - .Replay 109, 121, 135
 - .Replaytm 56
 - .Vrdump 5, 123, 135, 140
 - .Xdefaults 124
 - ~/Xdefaults 135
 - baseline 61
 - report 17, 33
 - results 61
 - script 138
 - snapshot 20
 - test suite 50
 - test.data 27
 - vrImageSaveFile 127
 - wcChildren 127, 130
 - wcClassName 127, 129
 - wcManaged 127, 129
 - wcPopups 127, 130
 - file access commands 193
 - file icon 55
 - file menu commands 59
 - forms
 - Snapshot Specification Entry 140
 - full snapshot 139

- G**
 - generateHtmlReport 222
 - getchildren command 209
 - getclass command 208
 - getconnectedapps 203
 - getcurrentwin 207
 - getfocuswidget 208
 - getparent command 208
 - getpopups command 209
 - getproperty command 213

getting widget state 210
 getvalue command 43, 44, 210, 211, 270
 glob 194
 global variables 185
 grabbing
 cursor
 See also warp 111, 248
 granularity
 snapshot 5
 GUI debugger options 247
 GUI interface 2
 script debugging 238

H

hot key 109, 137
 end a session (Ctrl+s) 140
 entering 109
 pause (Ctrl+p) 111, 140
 request a snapshot (Ctrl+i) 108, 115, 139
 Resume 140
 specifying 109
 Stop 140
 terminate a session (Ctrl+s) 111
 w 139
 warping 118

I

icon 4
 baseline snapshot directory 54
 directory 55
 file 55
 report file 55
 results snapshot directory 55
 script 19
 script file 54
 selecting 59
 selecting multiple 59
 test package elements 48
 iconify command 202
 image comparisons 43
 image snapshots 5, 42, 64, 122, 122, 125
 implicit pass/fail 204

includeButtonupMotion 150
 incr command 176
 insertion caret
 moving 233
 instance number 39, 214
 instrumented Xt library 98
 interactive mode 161
 isconnectedapp 203, 208
 ismanaged command 213
 iswidget 207

K

key command 201
 Key Delay 117, 222
 keyboard commands 201
 keyboard shortcuts
 See also hot keys 109
 keyed list 196

L

labels
 multiple 225
 Launch button 32
 LD_LIBRARY_PATH 99
 setting library paths 50
 LD_RUN_PATH environment variable
 setting 99
 ldd 100
 Learn Tag 37, 140, 142
 Learn Tag Hot Key 111
 learnTagKey 221
 library path 110
 button 110
 environment 110
 environment variable 31, 98, 100
 LD_LIBRARY_PATH 50
 modifying 98
 multiple 50
 setting 49, 99
 life cycle of an application 3
 link options 99
 linking library routines dynamically 98

Index

- list
 - items 230
 - ranges 230
 - Tcl 171
- list commands 180, 187, 224, 230
- list_extend command 230
- list_select command 230
- loadspec 141, 142
- local variables 185
- location specifier 250
- lower command 202

M

- main menu commands 58
- main window
 - Test Manager 7
- MANPATH
 - setting 49
- menu
 - items 224
 - Record/Play 22
 - Test Package 54
 - Test Suite 52
- menu selection command 224
- menubar_pick command 224
- menubar_select command 226
- mergespec 141, 142
- message command 203
- minimized name 38
- miscellaneous commands 203
- monitoring servers 268
- Motif™ 41
- mouse buttons 63
- mouse commands 198
 - click 198
 - dblclick 198
 - drag 198
 - multiclick 198
 - press 198
 - release 199
- move command 202
- msg_data 203

- msg_fmt 203
- msg_type_ID 203
- multiple application shells 39
- multiple applications
 - dynamically-linked 50
- multiple labels 225
- multiple library paths 50, 99
- multi-process testing 4

N

- names
 - widget 142
- narrative command 215
- nested X server 263
 - starting 266
- new test package
 - creating 53
- NULL 220
- null components 39

O

- omitButtonUpMotionWidgetClasses 150
- omitCoordinatesWidgetClasses 149
- omitDelay 149
- ONEFILE 112
- Open Look™ 41
- opening a test package 7, 51
- operands 177
 - constants 177
 - variables 177
- operators 178
 - arithmetic 178
 - bitwise 179
 - logical 179
 - relational 178
- option_pick command 224
- option_select command 226
- options
 - record and play 163
 - Record Tag Name 37, 39

P

- p option 7
- panes 4
- pass command 205
- pass/fail commands 204
- PATH 49
 - setting 49
- pattern matches
 - See also wild cards* 250
- pause command 214
- Pause Hot Key 111, 140
- pause session 111
- pauseKey
 - See also Pause Hot Key* 220
- play
 - control panel 15, 152
 - example 165
- play mode debugger 238
- play session 15
- play speed 17, 39
 - adjusting 39
- playbackSpeed
 - See also speed* 219, 222
- popdown command 203
- popup command 203
- popup_menu_pick command 225
- popup_menu_select command 226
- portability of scripts 35
- portable test cases 22
- preparing an application for VistaREPLAY 98
- press command 44
- proc 184
- procedure commands 187
- process 31
- .profile 6, 49
- program synchronization points 44
- programmability of scripts 35
- programmatic testing 3, 22
- programmed synchronization 44
- prompt 204
- Prompt For Test Case Name 115
- Prompt on Error 116, 221
- Prompt on Snapshot Difference 116
- promptOnSnapshotDiff
 - See also Prompt On Snapshot Difference* 221
- properties
 - Active Pause 110
 - Click Offset Tolerance
 - See also clickOffsetTolerance* 115
 - Client Startup Timeout
 - See also clientTimeout* 111
 - Compress Image Command
 - See also compressImageCommand* 110
 - Compress Image Snapshots
 - See also compressImageSnapshots* 110
 - context-sensitive lists 114
 - Debugger Warp Back Hot Key
 - See also debuggerWarpBackKey* 118
 - Delay Time
 - See also defaultDelayTime* 117
 - Diff Command
 - See also diffCmd* 118
 - Display Image Differences
 - See also displayImageDiff* 116
 - Exit On Snapshot Mismatch
 - See also exitOnDiffMismatch* 116
 - Key Delay
 - See also keyDelay* 117
 - Learn Tag Hot Key 111
 - learnTagKey
 - See also Learn Tag Key* 221
 - Library Path 110
 - Pause Hot Key
 - See also pauseKey* 111
 - Prompt For Test Case Name 115
 - Prompt on Error
 - See also promptOnError* 116
 - Prompt on Snapshot Difference
 - See also promptOnSnapshotDiff* 116
 - Record Button Up Motion 114
 - Retry Timeout
 - See also retryTimeout* 117
 - Snapshot Hot Key 115

Index

- Stop Session Hot Key
 - See also stopKey* 111
- test-suite wide 102
- Uncompress Image Command
 - See also uncompressImageCommand* 110
- Ungrab Pointer Hot Key, *See also ungrab-PointerKey* 111
- Use Tag Name
 - See also useTagNames* 110
- properties settings
 - test suite 50
- pseudo resources 127
- Q**
- question mark 225
- quoting 175
 - braces 175
 - double quotes 175
- R**
- raise command 202
- README
 - file 9
- record
 - control panel 10, 134
 - example 167
- record and play options 163
- Record Button Up Motion 114
- record controls 134
- record session 10, 33
 - properties 114
- Record Tag Names option 37, 39
- Record/Play 15
 - commands 61
 - menu 22
- recorded script
 - editing 22, 35
- recording widget names 39
- recording widget tags 145
- regression testing 4, 5, 42
 - multi-client 32
 - multiple applications 32
 - regression tests 5
 - relative path for a file name 133
 - release command 44
 - removing a breakpoint 252
 - .Replay 135, 149
 - file 121
 - replay command 32
 - form 163
 - Replay file 103, 109
 - replay session 33
 - replay.sh 7
 - .Replay_Appname 103
 - \$replay_args
 - baselineDir 220
 - clientPid 220
 - reportFile 220
 - resultsDir 220
 - scriptFile 219
 - replay_args 219
 - REPLAY_TERM_SYNCS 214, 262
 - REPLAY_TESTSUITE_PROPS 102
 - environment variable 123
 - setting 50
 - .Replaytm 56
 - Replaytm 56
 - replaytm 7
 - report file 17, 33, 55, 220
 - report view area 61, 63
 - .res 155
 - See also results snapshot directory* 55
 - resize command 40, 202
 - resource type conversion 269
 - result snapshot view area 61
 - result snapshot view pane 62
 - results file 61
 - results snapshot directory 55
 - results verification 41
 - resultsDir 220
 - Resume hot key 140
 - retesting applications 42
 - Retry Timeout 117, 221
 - return value

- procedure 185
- .rpt 155
 - See also report file 55*
- rubber-banding snapshots 129
- rxexit 218

S

- scale commands 229
- scale scrollbar 44
- scale_max command 229
- scale_min command 229
- scale_set command 229
- scales 224
- screen results
 - verification 3
- script 4
 - adding comments to 173
 - editability 41
 - portability 35, 41
 - Tcl 27, 170
 - viewing
 - continuous stream 243
 - writing 40
- script commands 35
- script debugger 238
 - popup menu 255
 - starting 238
- script editing 41
 - ASCII editor 41
- script file 54, 138
 - portability 41
- script file directory
 - viewing 57
- script icon 19
- script location
 - specifying 250
- script view area 17, 61
- script view pane 19
- script.tcl 241
- scriptFile 219
- scroll bar commands 227
- scroll bar slider 228
 - procedure 185
 - scroll_line command 228
 - scroll_max command 227
 - scroll_min command 227
 - scroll_page command 228
 - scroll_set command 227
 - see also commands 44
 - seektext 212
 - servwatch 268
 - session
 - play 33
 - record 33
 - session properties
 - controlling 103
 - set command 171, 176
 - setting
 - breakpoints 249
 - environment 48
 - environmental variables 6
 - library path 49
 - play speed 155
 - the environment 48
 - widget state 210
 - setting test suite properties
 - REPLAY_TESTSUITE_PROPS 50
 - setvalue command 43, 211
 - shell script 217
 - shortcuts
 - keyboard
 - See also hot keys 109*
- slider
 - scroll bar 228
 - setting speed 241
- snapshot 4, 41
 - baseline 5, 42
 - button 5, 13
 - command 205
 - comparing programatically 43
 - comparisons 152
 - compressed image 110
 - directory 6
 - viewing 57

Index

- file 20
 - full 139
 - granularity 140
 - hot key 115, 139
 - image 42, 125
 - scope 115
 - types 125
 - widget 41, 125
 - snapshot granularity 141
 - snapshot specifications
 - loaded 141
 - merged 141
 - .snp suffix 20
 - source debugger 268
 - special characters 174
 - interpreting 174
 - specifying a script location 250
 - specifying cursor location 201
 - spee
 - setting 155
 - speed
 - setting
 - See also playbackSpeed* 155
 - speed slider 152
 - square brackets 225
 - starting the script debugger 238
 - starting the Test Manager 7, 48
 - state
 - verification 44
 - static images
 - viewing 267
 - status information
 - widget 44
 - stdout 155
 - Stop button 14
 - Stop hot key 140
 - stop session 111
 - Stop Session Hot Key 111
 - stopKey
 - See also Stop Session Hot Key* 221
 - string 171
 - arbitrary 172
 - Tcl 171
 - string manipulation commands 191
 - sub-image snapshots 129
 - substitution
 - backslash 173, 174
 - command 173
 - quoting 174
 - variable 173
 - Subtest 204
 - system command 44, 217
 - invoking 44
- ## T
- Tag Manager 79
 - tag name
 - See also widget tags* 110, 114
 - target widget 199
 - Tcl 34, 170
 - arrays 172
 - command structure 171
 - debugger 32, 252
 - in play mode 32
 - expression 177
 - expression evaluation 172
 - list 171
 - procedure 184
 - string 171
 - verb 170
 - zero indexing 189
 - .tcl 155
 - See also script file* 54
 - Tcl commands
 - extended 196
 - Tcl scripts
 - writing 27
 - Tcl_CreateCommand 272
 - tclsh 170
 - terminal emulation 203
 - termsync 262
 - test case 6, 12, 13
 - management commands 204
 - multiple 33

- report 4
 - report summary 23
 - success 43
 - verification 15
 - test data 27
 - test management commands 198
 - Test Manager 6
 - features 33
 - main window 7, 52
 - set up and starting 3, 6
 - starting 7
 - test package 6, 33
 - commands 48, 57
 - creating 7
 - deleting 9
 - directory 6
 - menu 54
 - opening 7
 - window 8, 239
 - test portability 3
 - test scripts
 - manual 22
 - test sequence number 27
 - test suite 6, 31, 33
 - commands 48, 51
 - directory listing 45
 - displaying test packages 33
 - menu 52
 - properties setting 50
 - test suite-wide properties 50, 102
 - testcase command 33
 - testing
 - multi-process 4
 - programmatic 3
 - regression 4, 5
 - text 4
 - text area
 - multiple rows 232
 - text command 40, 201, 224, 232
 - text_delete command 233
 - text_delete_string command 233
 - text_find_pos command 233
 - text_find_string command 233
 - text_insert command 233
 - text_replace command 234, 235
 - text_replace_string command 234, 235
 - text_select command 233
 - text_select_string command 233
 - text_set command 232
 - \$title 57
 - Tool Command Language
 - See also Tcl 34*
 - tutorial
 - play session 15
 - type converters 269
- ## U
- unattended VistaREPLAY sessions
 - See also batch mode 6*
 - uncompress command 110
 - Uncompress Image Command 110, 221
 - Ungrab Pointer Hot Key 111
 - ungrabPointerKey
 - See also Ungrab Pointer Hot Key 221*
 - Update Baseline Transparently 116
 - updating baselines 5
 - usage model 3
 - Use Tag Name 110, 220
 - user interaction commands 198
 - useVirtualKeyNames 219
 - utilities
 - diff 43
 - xwddiff 43
- ## V
- variable number of arguments 186
 - See also arg 186*
 - variable substitution 173
 - variables
 - environmental 6
 - global 185
 - local 185
 - verification of screen results 3
 - verification of states 44

Index

- verification points 4, 43
- verifying results 41
 - widget snapshots 42
- verifying widget snapshot differences 159
- version 204
- viewing test package elements 57
- virtual server 268
- virtual X server 266
 - starting 267
- VISTAHOME environment 49
- VISTAHOME/bin 262
- VistaREPLAY
 - architecture 30
 - customizing options 6
 - driver 31, 32
 - features and concepts 30
 - preparing an application 98
 - script debugger 238
 - Test Manager main window 7
- VistaREPLAY Test Manager
 - features 48
 - starting 51
- Vralias 135
- .Vrdump 123, 140
- Vrdump 5, 123, 135, 205
- vrImageSaveFile 127
- VrRString 270
- .vrSave 125
- VrSetTypeConverter 211
- W**
- warp 118
- warping
 - See also cursor and grabbing 111*
- wcChildren 127, 130
- wcClassName 127, 129
- wcManaged 127, 129
- wcPopups 127, 130
- whatlib command 49
- widget
 - ancestor 38
 - custom 269
 - identifying 36, 38, 207
 - instance 125
 - name 36, 39, 142
 - minimized 37
 - qualified 37
 - recording 39
 - pop up children 127
 - regular children 127
 - resources 269
 - snapshot file format 128
 - status information 44
- widget class 125, 127
- widget class specification 125
- widget comparisons 43
- widget hierarchy 37, 208
 - graphical view 36
- widget ID 199, 210
 - obtaining 210
- widget information commands 198, 199, 206
- widget snapshot differences
 - verifying 159
- widget snapshots 41, 125
- widget state
 - getting 210
 - setting 210
- widget synchronization
 - automatic 44
- widget tags 36
 - fully-qualified 39
 - minimized 39
 - recording 145
- widget tree 38
- widgetid command 207
- widgetname command 207
- widgettag command 207
- wild cards 225, 230
 - asterisk(*) 38
- wildcard notation 41
- window management 202
- windowid command 207
- writin Tcl scripts 27

X

- X resource file format 20
- X Toolkit 269
- X Toolkit library 31
- X11R5 99
- xdebug 254
- .Xdefaults 124, 269
- xmcalc 11
- XmPushButton 125
- Xnest 265, 266
- Xt library 100
 - instrumented 98
 - linking dynamically 98
 - routines 110
- XTLIB_DEBUG_ON 111
- XtRWidget 270
- xwd 5
- xwd bitmap file 42
- xwddiff utility 43
- xwddiff(1) 43
- xwud 267

Z

- zero indexing 189

Index