$RCSfile: header.mm,v $ $Revision: 1.1.2.1 $ $Date: 1994/02/03 22:41:22 $ # $RCSfile: ch04.sm,v $ $Revision: 1.1.2.1 $ $Date: 1994/02/03 22:42:07 $

Chapter 4. Debugging the Automated Tests

Interpreting test results often requires debugging each individual discrepancy until its cause can be determined. The follow sections describe how the automation technology supports debugging.

4.1 Types of Failures

This section describes the types of failures which may be found by automated testing. The next section provides some hints on how to go about debugging those failures.

4.1.1 Visual Comparison Failures

A common visual comparison failure occurs when the original, recorded image differs in size from the current image under test. Because the automation code does not know how to compare different-sized images, it simply issues a warning. In interactive mode, the original region will be displayed in the dialog box, although the overlay feature is disabled. One can, however, use the X client **xmag** and count pixels to help determine what has changed.

When two images are the same size, the automation code does a pixel-by-pixel comparison of the images. If any one pixel differs between the two images, the automation code prints out a message that a CompareVisual failure has occurred. In interactive mode, the tester can examine both the recorded and current images, and overlay the differences (highlighted in red) on top of either image.

There are several problems that can produce a **Visual Image Not Found** error: the recorded image is missing or corrupted; the environment variable **$AUTOVPATH** is not defined; the directory defined by **$AUTOVPATH** does not exist, or the visual file in that directory is not readable or is missing.

This error can also occur if the test or the script was modified between the recording and the playback. For instance, a test is recorded with a script that specifies six visual comparisons. Six images are stored in the **.vis** file. Later, a tester adds an additional **CompareVisual** command to the script but fails to rerecord the test. On the next playback, the seventh visual comparison fails because no image is stored with which to compare the current image.

Either a **Checksum** or a **Visual Image Corrupt** error can occur when something in the visuals file has become corrupted. The only way to fix this problem is to rerecord the original, using the original baseline.

4.1.2 Fatal Errors

The signals are trapped by the automation code, and redirected to a special routine **AutoExitSignal** which simply prints out a "signal received" message, and the type of signal. Signals so captured are:

SIGHUP	hangup
SIGINT	interrupt
SIGQUIT	quit
SIGKILL	kill
SIGSEGV	segmentation violation
SIGFPE	arithmetic (floating point) exception
SIGILL	illegal instruction
SIGTERM	software termination signal
SIGABRT	abort
SIGBUS	bus error
SIGSYS	bad argument to system call

Be certain in reviewing the differences between files that you search for signal messages.

4.1.3 Other Failures

Many other kinds of failures only show up by examining the contents of the differences file (**.err**) produced by the RUN script; this differences file is the product of the command **diff** run against the output file stored at record time and the output file for the current run. This section describes some examples of what differences might turn up; very subtle bugs may make their appearance known only through small differences in the output.

- Messages printed by callback routines can indicate more or fewer callbacks are occurring on some action.
- Differences in size reported by callback routines may show layout differences not captured by CompareVisual, usually because the top-level manager is not photographed.
- User data printed out may indicate a test was run in the wrong mode.
- CompareVisual failure messages will turn up in diffs.
- Running a non-automated **mwm** can produce window command failures which show up as differences. This may precipitate CompareVisual failures further along in the test.
- Key-binding error messages may appear in diffs if the **qauser** environment in not used.
- **xisMovePointer** errors may mean that layout has changed, and the pointer can no longer move to a certain widget.
- If tests are run in interactive mode, "Accept" messages will appear in diffs but can be ignored.

4.2 Debugging Tricks and Testing Strategies

This section will describe several automation features which can be used to help debug test failures. While nothing is as good as first-hand experience, this section will give you some idea of the tools at your disposal.

Generally, one will wish to run the entire test suite automatically then debug the failures one at a time. If the failures appear to be due (in whole or in part) to visual differences, it often makes the most sense to run the offending test in interactive mode. The various overlay capabilities of interactive mode have been described earlier. This is the quickest way to assess the nature of the change; once you know what the difference is you can debug the cause. However, there are other modes and features which can be used.

4.2.1 The **-a** Flag

The precursor to the interactive window was the **-a** flag. When run with this flag, tests print out the x,y location of each pixel that differs from the original. If the differences are really a single or two pixels and you have difficulty seeing the differences in interactive mode, the information provided by **-a** may be helpful.

4.2.2 Window Dumps

Another visual debugging tool which predates the interactive mode window is the window dump feature. Tests run with the **-W** flag create xwd-format window dumps which can subsequently be viewed with the **xwud** client. Your company may have other tools that use xwd-format images; if so, this option may be useful for you.

4.2.3 The Manual Command

For certain problems, you will need to be able to manipulate the widgets in the test client directly. Because automation takes control of the pointer, we have provided a scripting language command **Manual** which tells the automation code to suspend its normal event handling and return control to the user, until the user clicks the Continue button on the Manual mode DialogBox. At that point the test will resume. This gives you an opportunity to examine closely what happens at very specific points during the test.

To use **Manual**, simply add the command at the appropriate point in the test's **.scr** file and remake the **.Scr** file. When the Manual command is reached, a special dialog box will appear and you may exercise the test client normally. At this point, you can also invoke other useful X debugging clients, such as **xmag**, **xwininfo**, or the OSF/Motif test tool **xmruler**.

4.2.4 Delay Mode

If you need the playback slowed enough for a human tester to watch the changes occurring, but don't actually require the suspension of the event loop provided by **Manual**, the **-D** or **Delay** mode might be helpful. Run with **-D** *seconds*, the code "sleeps" for the requested number of seconds before executing the next instruction. The Delay flag can also be used when very fast workstations execute instructions before the visuals have "caught up".

4.2.5 Trace Mode

The **-T** or **Trace** mode is useful in many debugging situations. In Trace mode, the scripting instructions are echoed to the output file as they are executed. This greatly simplies, for instance, the process of determining which button produces an extra Activate callback. It may be worth considering using the Trace mode by default for all recording and playback. This can also help determine cases where failures are merely the result of recorded output files getting out-of-sync with the tests or scripts currently executing.

4.2.6 The xmruler Tool

The tool **xmruler** is provided in the directory **$TOP/tests/tools**. This client allows the user to "measure" widgets on screen which claim to be a certain size.