$RCSfile: header.mm,v $ $Revision: 1.1.2.1 $ $Date: 1994/02/03 22:41:22 $ # $RCSfile: ch03.sm,v $ $Revision: 1.1.2.1 $ $Date: 1994/02/03 22:42:06 $

Chapter 3. Running Automated Tests

This section describes the runtime environment for automated testing.

3.1 Environment

The automation code does make some assumptions about key bindings, window manager modes, and the like. Shipped with the test suite is an entire home directory for a test user, which we refer to as the **qauser** environment. Test environments should use the **.Xdefaults**, **.mwmrc** and **.motifbind** files from this environment in their testing. The **.motifbind** file may need modification for hardware platforms other than the reference machines used by OSF. Consult the **Virtual Bindings** man page for more information.

3.2 RUN Scripts

Each automated directory contains a file **RUN.custom**, which is used to store specifics about what flags to pass to each test which should be executed. An executable **RUN** script is created by issuing the command:

make RUN

The RUN script is "made" by substituting the unique **RUN.custom** section into the appropriate place in the standard template **$TOP/tests/General/RUN_template**. In addition to the flags specified in the customization file, additional flags may be passed to all tests by passing those flags to the **RUN** script. For example, to run all the tests in Trace mode (**-T**), the following command would be used:

RUN -T

Examine a built **RUN** script. Note that tests can be defined with special flags and run more than once, with different flag combinations. These flags are most often used to allow for multiple executions of the same test for different test purposes:

-I Denotes a specific instruction file.

-S Denotes a specific script file.

-V Denotes a specific visuals file.

-O Denotes a specific output (stdout) file.

-u User data, often used by tests to support different modes.

By default, a test named **Traversal1** will read in an instruction file **Traversal1.Dat**, a script file **Traversal1.Scr**, redirect standard output and standard error to a file **Traversal1.prt**, and store in or read images from a file **Traversal1.vis**. If a test supports two modes, **destroy**, where widgets are destroyed when activated, and **unmap**, where widgets are unmapped when activated, we might wish to run the test twice, once in each mode. During the two runs, one can use the same instructions and scripts but expects different visuals, behavior and output. In the **RUN.custom** file, one might specify two runs this way:

```
set Traversal1a = ‘Traversal1 -I Traversal1.Dat \
                       -S Traversal1.Scr -V Traversal1a.vis \
                  -O Traversal1a.prt‘
set Traversal2a = ‘Traversal1 -I Traversal1.Dat \
                  -S Traversal1.Scr -V Traversal1b.vis \
                       -O Traversal1b.prt‘
```

The two variables **$Traversal1a** and **$Traversal1b** would appear in the **foreach** list that loops through each test case. Of course, if the two modes of the test also required different instructions and different scripts, one could specify that, as well.

The last section of the **RUN** script loops through the new output files, comparing them (via **diff**) to the saved output files from a previous run, and finally prints out a summary of which files showed differences. *These differences should be examined very carefully.* Often, a serious error may be revealed only by a minor difference in the output file. For example, if PushButtons suddenly stopped generating callbacks on activation, the only clue might be a diff between output files that showed a printed message about "Callback received" missing! Also, visual differences will be highlighted in the differences. Please do not overlook this important step in analyzing test results.

3.3 Execution modes

There are a large number of modes and flags available. Only the major features will be discussed in this section. The complete list of flags can be viewed by running any properly compiled test with the **-help** option. Also, the Reference section of this book repeats the same information available through **-help**.

3.3.1 Record mode

As stated earlier, the purpose of a regression test is to compare the appearance and behavior of one, stable version of software, against a newer, less stable version. Thus, one must first *record* an automated test.

The term *record* as used in this context should not be confused with the actions necessary to record a test under most record-and-playback test software available today. Because of the scripting language, it is not necessary for the user to physically carry out the actions he or she wishes to generate. Instead, those actions are described programmatically via the script. Our belief is that conventional record-the-user-actions mechanisms are more cumbersome and error-prone, since any mistake made by the user can often be repaired only by rerecording the entire session. Scripts, like software programs, can be built up section by section until a long working session can be simulated.

A test is recorded by running it with the flag **-r**. If the environment variable **$AUTOVPATH** is set, the output (**.prt**) and visual (**.vis**) files will be written out in the directory specified by **$AUTOVPATH**. If it is not, the files will be written out in the current directory, overwriting any other files that might be present.

The QATS shipped by OSF contains visuals and output files recorded against OSF/Motif Version 1.2 during the final regression tests conducted at OSF. Those using this package to retest implementations of OSF/Motif might choose to use OSF's visuals rather than record their own.

Tests being recorded look quite odd while they are running. This is because the automation code must take control of the colors used in order to notice certain variations in widgets between record and execution. Do not be concerned about the strange appearance of the tests being recorded.

3.3.2 Batch Mode

Batch mode can also be called comparison, playback or execution mode. In this mode, which is the default, visuals located in the the **AUTOVPATH** directory are compared with the visuals of the currently executing test. Failures are listed in the output file. There is a flag, **-b**, for batch mode, but it is optional. A test executed with no flags specified runs in batch mode.

Be certain that **$AUTOVPATH** is specified, and that the visual files in that directory are readable. Otherwise, the test will simply print out a warning that visual comparisons will be ignored and continue. This could waste quite a bit of time if it is not discovered until a run is complete!

3.3.3 Nocompare Mode

The flag **-c** lets tests be run without any visual comparisons. This mode is useful as one is debugging a script, since eliminating visual comparisons speeds execution. It is also useful when, for whatever reason, one might wish to verify the test output without considering visual differences. Its speed advantages can also be useful if one is linked with utility libraries which seriously impede performance. Possible examples are memory analysis tools, performance profilers, test coverage analyzers.

3.3.4 Interactive mode

Interactive mode is the most useful mode for debugging. In this mode, a DialogBox is popped up whenever a visual difference is encountered. The dialog box shows the recorded image and the current image side-by-side, with differences highlighted in red. The differences may also be overlaid on either the recorded or the current image, in order to determine exactly what pixels have changed between the two runs. When the visual difference results because the recorded and current images differ in size, only the recorded image is shown. Other debugging clients, such as **xmag**, can be used to help determine the exact cause of such size differences.

The interactive dialog also sports two buttons: an **Accept** and a **Failure** button. If the tester is able to immediately determine that some variation is acceptable (for example, new functionality that changes the appearance), he can press the **Accept** button, and a message noting the acceptance will be written out the the output file. If the difference is a definite or possible failure, the tester would use the **Failure** button. OSF testers have found that it is often most useful to run the entire directory in batch mode, then examine any test with visual differences by running it in interactive mode.