$RCSfile: header.mm,v $ $Revision: 1.1.2.1 $ $Date: 1994/02/03 22:41:22 $ # $RCSfile: ch01.sm,v $ $Revision: 1.1.2.1 $ $Date: 1994/02/03 22:42:04 $

Chapter 1. Automation Basics

This section describes what makes a test automatable. In addition, the libraries and file-naming conventions for automated tests are described, and instructions on building the automated test suite are provided.

1.1 Requirements for Automatable Tests

The following restrictions exist in the current implementation of the automation technology. It may be possible in some cases to proceed with limited testing even if not all these requirements are met. We welcome your feedback on which of these restrictions poses the largest impediment, and how you work around them.

1.1.1 XmCreate Convenience Routines

In order to automate events to any widget, that widget must be created with the appropriate **XmCreate...** function rather than a call to **XtCreateWidget**, **XtCreateManagedWidget**, or **XtVaCreateManagedWidget**. The automation code uses "wrappers" for the **XmCreate...** functions and stores essential information about the widgets created. You may, however, continue to use **XtCreateShell** functions.

1.1.2 Unique XtNames

Each widget in the client under test must have a unique name, provided at create time. Automation scripts reference widgets by this "XtName" and the code has no algorithms for resolving duplications. This is likely to be the most difficult restriction for large applications.

1.1.3 CommonTestInit

All test clients must start with a call to **CommonTestInit**. The function sets up signal handlers for fatal conditions, initializes the Xt toolkit, creates an application context, performs automation initialization and optional malloc package initialization, and finally, creates and realizes an application shell called **Shell1**. The presence of a single application shell by this name is also a requirement of automation. You may pop up additional shells that have **Shell1** as a parent.

1.1.4 CommonPause

Currently, all automated clients are required to have one call to **CommonPause()**.

1.1.5 Unsupported OSF/Motif Version 1.2 Functionality

Neither Drag and Drop nor *dragging* TearOff Menus are supported by version 1.2 of the automation libraries.

1.1.6 Private Event Loops

Clients that implement their own event loops are not automatable.

1.1.7 Resolution Independent Units

Test clients that use resolution independent units (for example, **Xm1000TH_INCHES**) should only record and play back on the same screen, or on screens of the same pixel resolution. Simply, a one-inch wide button might be rendered with 97 pixels on one monitor, but require 100 pixels on another monitor. The visual comparison code would report a "wrong size" error in this case.

1.1.8 The uil-Based Application

Applications which use a uil interface are not automatable under the current implementation.

1.1.9 XmTrackingLocate

Any calls to **XmTrackingLocate** during the execution of an automated test will confuse event synthesis and cause an error.

1.1.10 Root Window and Root Menu

There is no target in the scripting language for the root window. Consequently, one cannot, for example, post the root menu.

1.1.11 Icon Box

The icons in the icon box can be manipulated, but the icon box itself cannot.

1.1.12 Multiple Screens

The current implementation contains no facility for record or replaying on multiple screens of a single machine.

1.2 Automation Files and Directories

An automated test must be linked with the automation libraries. In addition, supporting script and instruction files will be required, and a naming convention for these supporting files is described.

1.2.1 Libraries and Supporting Files

There are five libraries that support automation.

./tests/Auto/lib/Create	Creation routines for widgets
./tests/Auto/lib/Script	The automation scripting language code
./tests/Auto/lib/Visual	Visual record and compare code

./tests/Auto/lib/Synth	Input synthesis library
./tests/Auto/lib/Common	Convenience routines for use in tests

In addition, the following files support running the automated test suite.

./tests/General/dirsAuto	List of automated directories
./tests/General/RUN_template	Template RUN script
./tests/General/bin/Genrun	Script that produces customized per-directory RUN scripts

In each automated directory exists a file **RUN.custom**, which contains specifics for running the tests in that directory under automation. For instance, this file defines the environment variable **$AUTOVPATH**, the value of which should be set to the location of the visual and output files to be used for comparison. By issuing the command:

make RUN

You build an executable **RUN** script which includes the directory-specific information. You should always use the **RUN** command to run the automated tests, to ensure the proper flags are passed to each test. In addition to the flags specified, any flags you specify to the **RUN** command itself will be passed through to each client executed within the script.

1.3 File Structure of an Automated Test

Each automated test is made up of the following files:

file	test executable
file.o	test object file
file.c	test source
file.h	test include [optional]
file.dat	instructions for manual testing
file.Dat	pre-processed instructions for manual testing
file.scr	automation script
file.Scr	pre-processed automation script
file.prt	saved output (redirected **stdout** and **stderr**)
file.vis	saved visual images

1.4 Building Automated Tests

By default, tests are built for manual use. In order to build the tests for use under automation, you must first:

- Build the libraries under **$TOP/tests/Auto**.
- Build the Makefiles in Automated test directories with a special flag.
- Build the tests.

1.4.1 Building the Libraries

To build the libraries, execute the following commands, starting out in directory **$TOP**. Note that the files **Motif.rules** and **Motif.tmpl** in **$TOP/config** contain modifications essential to automation. If you do not use OSF's versions of these files, you will need to modify your **config** files to include the necessary definitions.

```
cd tests/Auto
make Makefile
make Makefiles
make
```

1.4.2 Building the Makefiles

Next, build the Makefiles for the automated test directories. You may execute the following commands in each automated directory, or write a script that reads the file **dirsAuto**. (The **-DAUTOMATION** flag is not propogated when Makefiles are built from the top of the **tests** directory.)

```
make IMAKE_DEFINES="-DAUTOMATION" Makefile
```

1.4.3 Building the Tests

Lastly, build the tests themselves. When linked with all the automation libraries, each test is quite large. Test executables of 2 Mb are not uncommon. Be certain to allocate enough disk space, or write a script that will build, run, and then clean each test directory before moving on to the next.

If the scripts for any tests contain Window Manager commands (which start with the word **Window** or **Icon**), it will be necessary to use a OSF/Motif Window Manager (**mwm**) built for automation. The only difference between a default **mwm** and one built for automation is that the latter posts a per-window property with information about the location and size of window manager decorations. To build **mwm**, build the Makefile using the same command listed above, then rebuild **mwm**.

The following restrictions about the window manager commands should be noted:

- For version 1.2, the mwm client and the server should be run on the same machine, or on machines using the same byte order.
- Window commands executed when a non-automated mwm is running will cause failure messages to be printed to the output file. In addition, subsequent visual comparisons may fail because a window was not iconified or resized as expected.