# A DISTINCT APPROACH FOR X/MOTIF APPLICATION GUI TEST AUTOMATION

**K.V. Maruthi Prasad**[1]

[1]*ISRO Satellite centre, HAL Airport road, Bangalore-17, India*

## Abstract

*This paper titled "A distinct approach for X/Motif application GUI test automation" presents the research results of the innovative approach applied on X/Motif applications under test automation. It is the excerpts of the results obtained on X/Motif GUI software test automation without record & playback technique. This approach is based on virtualisation of mouse button and key board key events using "XSendEvent" Xlib routine. It also presents the details about the software that has been developed for X/Motif GUI application testing automated through a tester input file of identified keywords with the necessary input as test cases. The paper identifies the limitations and future plans for the expansion of the work.*

***Keywords:*** *X/Motif, test automation, XSendEvent, record & playback, GUI.*

----------***----------

## 1. INTRODUCTION

ISRO (Indian Space Research Organisation) is the premier government institute involved in space research and development activities. ISRO has been known for it's accomplishments in nation building through science & technological innovations in space field. GEOSCHEMACS (Geostationary Earth Orbit SpaCecraft HEalth Monitoring Analysis and Control Software) is the in-house developed end to end software solution and primary set of ground software elements used for Indian geo mission health monitoring, control and analysis. GEOSCHEMACS is a software package based on client / server architecture with the development environment primarily consisting of C/C++, X/Motif, Oracle on UNIX / LINUX Operating System flavours. The total size of GEOSCHEMACS is around one million lines of source code.

The role of ground software elements has been crucial and critical in meeting the ever expanding space services for users. It is necessary to evolve reliable software for ground elements used for spacecraft health monitoring, analysis and control so that there is no disturbance in supporting space services. Testing sufficient enough is the only way to make any software reliable and worthy of using. GUI based spacecraft health monitoring analysis & presentation software consist the major part under GEOSCHEMACS. X/Motif is the predominant GUI development environment. This set of X/Motif GUI software is highly interactive in nature and also require enhancements / modifications as per spacecraft specific or general requirements. The test cases, test combinations are more in number and it is difficult to test repetitively for regression testing. Hence, X/Motif GUI applications test automation is required.

Test automation can enable some testing tasks to be performed more efficiently than by testing manually. Automation of testing makes the effort involved in performing regression tests at minimal. GUI based applications test automation allows the tester to run more tests in less time and also to execute them more often. Automation of GUI based application testing enables us to execute test cases of input entry with greater accuracy, run difficult or impossible test cases to do manually. GUI applications test automation gives increased confidence on the software under test. Test automation ensures the consistency & repeatability of tests and reuse of tests. Test automation reduces costs and increases the quality of the testing tasks.

### 1.1 What is X/Motif?

Motif refers to both a Graphical User Interface (GUI) specification and the widget toolkit for building applications that follow specification under the X window system on UNIX and other POSIX compliant systems. It is a toolkit or widget set layered on Xlib and Xt. Xt (X Toolkit or Xtoolkit Intrinsics) provides necessary functionality for implementation of graphical user interfaces. Xt provides an object oriented framework for creating reusable, configurable user interface components called widgets. Motif provides widgets for such common user interface elements as labels, push buttons, menus, dialog boxes, scroll bars and test-entry or display areas. The X window system (or simply X) is a hardware and operating system independent windowing system developed by MIT. The system is based on client server architecture. It is a distributed, network transparent, device independent windowing and graphics system. X divides the screen into multiple input and output areas called windows. X takes user input from a pointer (mostly mouse) and also handles keyboard input. X was designed as a network protocol – a

predefined set of requests and replies – between two processes. One of those processes is an application program called a client, and the other, the server (X server) which controls the display hardware, keyboard and pointer.

Motif GUI uses X as the window system and Xt as the platform for the application programming interface. The stable X version 11 (X11) system provides subroutine library, Xlib. Xlib provides functions for connecting to a particular display server, creating windows, drawing graphics, responding to events and so on. X was designed to provide windows on bitmapped terminals. Xlib is the C language interface to X protocol. X event is packet of information that is generated by the X server when certain actions occur (such as moving the pointer or pressing a key).

**Table 1** User Interface Model for Motif

| Motif (Xm library) |
|---|
| Xt Intrinsics (Xt library) |
| Xlib (X window system library) |
| Operating System |

## 1.2 What is GUI Test Automation?

GUI testing to mean that a GUI-based software application is tested solely by performing sequences of events on GUI widgets; and the correctness of the software is determined by examining only the state of the GUI widgets. GUI test automation is difficult. It is technology-dependent. It is usually acknowledged that an automatic process is faster than a manual process. The GUI reacts to various user events like mouse clicks and keystrokes. This allows the user the front end to communicate with the underlying application. GUI in turn communicates with the user via method calls or some kind of messaging system. GUI functional test means validating GUI objects, checking functional flows by operating GUI objects and verifying output data which are generated in back end and then displayed in front page. People have ventured to perform these operations following different models and techniques which can range from fully manual to semi automatic. However the tendency is to automate as much as possible so as to make it very fast and have a huge coverage which would otherwise take a tremendous time for a human. GUIs typically have a large number of potential inputs and input sequences. To reasonably verify the system's functionality requires a large amount of testing. Performing these tests manually is costly and can be practically impossible. Therefore, it is necessary to perform automated testing. Automation testing is a process of writing a computer program, to do testing that would otherwise need to be done manually. The test scripts once written or developed for the application or the software under test can be run repeatedly as per the requirement. Also, it is a quick and efficient process without the manual intervention. As such the test coverage of the application, the maintenance of the scripts, unattended modes of the user were observed to be beneficial in the automation testing process.

Too many repeatable tests are challenge to man. It doesn't matter for machine to do so. That's one reason that automation of GUI testing is required. Another reason is for regression test purpose. The idea of software test automation is to let computer simulate what human do when manually running a test on the target application. A challenge of GUI test automation is how to recognize the GUI objects and the actions of them by machine

## 1.3 Capture and Replay Technique

The most prominent technique that is available in most of the GUI testing automation tools is capture and replay or record and play back. The tester interacts with the system GUI to run the system, thus generating sessions of sequence of mouse clicks, UI and keyboard events; The tool captures and stores the user events and the GUI screen shots; a script is produced per each user session. The tester can automatically replay the execution by running the script. This process is extremely labour intensive and largely relies in the ability of the test designer. It is huge and expensive to manage and quite a lot of work to generate the entire test cases. A captured test is a linear script and it is far from good solution for a number of reasons, including: The test script only stores inputs that have been recorded, not test cases. So it does not know what the expected results are until you program it. Small change introduced in the AUT (Application Under Test) can break most of the scripts. The captured script can only cope with precisely the same conditions as when it was captured.

## 1.4 Keyword Driven Approach

To provide the capacity of testing the GUI automatically for all the possible or required test case combinations without recording, another method called "key-word driven approach" is suitable. This approach has several advantages such as: 1. low maintenance since the test cases are concise, clear, easily readable, easy to modify and easily reusable; 2. Keyword can be reused in multiple test cases; 3. Not dependent on any language;

## 2. THE CONCEPT

The concept behind this research approach is virtualisation of mouse button and key board key input events. Virtualisation of manual mouse clicks & key input is the primary problem requires to be solved. Mouse clicks and key board key inputs are queued as X events to the X server for the required X client software through "XSendEvent" X-lib routine. "XSendEvent" routine facilitates to send X event to the given X window identifier without man in loop.

At first step, it is required to get the top window of the X-client software under test. This is possible by using

"XQueryTree" X-lib routine with the given X-client software name and searching from root window of the screen. It is necessary to get window and widget hierarchy of the given X-client software for sending Xevents through "XSendEvent" for the required X window identifier. This is possible by the utilisation of Editres protocol.

The most important task is allowing the tester to define test case combinations for GUI test automation on the required X-client system. This can be made possible by designing a tester input file of test case combinations to be executed on the system under test with certain keyword list & specification of corresponding input against them. Resources of the X-client under test (attained through editres protocol) can be used as the basic input in deciding required unique X-window identifier for sending X event with reference to each test case.

## 3. THE DESIGN

The research has been focused on the basis of keywords for testing the X/Motif application GUI automatically for the required or possible test case combinations. Just like any GUI test automation, this approach also contains following main components:

1. Application under Test (AUT) or System Under Test (SUT): It is the X/Motif application under testing.
2. Test case generation: Test cases are given through an input file. They are generated manually with the knowledge of allowed keywords and the corresponding input against them for the respective test case.
3. Test case Execution: Execution of test cases is automated fully with the input of test cases through the manually generated tester input file. The automation of GUI testing has been through virtualization of X events for mouse button clicks and text keyboard input.
4. Test results verification: Test results are basically verified with the state and presence of the GUI widgets.
5. Test observations log: The results and observations along with error cases are logged to the given tester results output file in ASCII format.

This approach is useful for all GUI testing automation cases at unit, integration, system, acceptance and regression levels. The inputs are based on GUI components functionality and structure. This approach answers the GUI test automation requirements such as: 1. Able to recognise or identify the components of a GUI of the AUT. 2. Able to exercise / send GUI events (such as mouse clicks and text field input) for the required GUI components. 3. Able to test the functionality underlying a GUI set of components.

### 3.1 How to identify the Components of X/Motif Application GUI?

The components of any X/Motif Application GUI can be identified basically with the associated X window identifier. Various X/Motif GUI components that can be categorised into different widget types are push button, toggle button, label, option menu, radio box, combo box, main window, check box, pull down menu, pull right menu, text widget, scroll test widget, file selection box, dialog selection box, scrolled list, scrolled vertical bar, scrolled horizontal bar, scale widget, up arrow button, down arrow button, left arrow button, right arrow button, drawing area widget. These mostly cover all variety of recognising X/Motif widget components. As our idea is to send X events to the identified GUI component's X window identifier, it is now required to devise a method for getting the X window identifier. The method of identifying window identifiers and generating test cases for automating the GUI testing has been devised based on certain keywords. The widget hierarchy with the associated X window identifiers, positions and geometry of the Application Under Test (AUT) are attained through Editres protocol. Editres protocol allows for getting as well setting supported widget resources of the AUT widgets.

### 3.2 Test Cases Generation

The keywords framed for preparing test cases are: xappName, tstBegin, tstEnd, testStrt, testEnd, testLevl, wdgtName, wdgtType, wdgtData, testNrml, testRptn, seqRptn, testRang, seqRang.

Each keyword expects zero or more input against that. Few of the keywords can be provided optionally (with default input if not given).

**xappName**: This keyword is used for giving the X/Motif application name under test.

**tstBegin**: This keyword is primarily for indicating the start of test cases input.

**tstEnd**: It is for indicating the end of test cases input.

**testStrt**: This keyword is used for test case starting with the corresponding test case number as input against it.

**testEnd**: This keyword is used for test case ending with the corresponding test case number as input against it.

**testLev**l: It is the facility for accommodating test case combinations at a group level also.

**wdgtName**: It is the primary keyword for identifying the associated X window identifier of the required widget or gadget through which the necessary X events can be spooled automatically. The X window identifier is found out using Editres protocol with the input of full path of corresponding Widget resource name and it's sequence number of occurrence for the scenario of test case.

**wdgtType**: It is meant for providing the type of widget with reference to the test case.

**wdgtData**: This keyword can be used for giving necessary or additional data for the test case of interest and for the corresponding widget type (data such as menu item number, option number or textual keying input). It is an optional input. Rest of the keywords can be utilized optionally for various purposes of test case normalisation, required number of test case repetition, test case data range etc..

As it was mentioned, the test cases generation is manual and tester input file of the interested test cases are prepared using the above mentioned keywords.

### 3.3 How to Automate the Test Cases Execution?

The major point under automating the test case execution is the ability of sending the GUI events automatically. GUI events are meant as X events such as mouse clicks and text key input to the application under test. "XsendEvent" Xlib routine (or function) is used for sending the required Xlib events delivered to the necessary X-window of the AUT. "XSendEvent" routine requires five arguments to be passed as 1. Pointer to Display type (which specifies the connection to the X server), 2. Window identifier for which the event is to be sent to, 3. A Boolean type value (True); 4. Event mask of long data type (ButtonPressMask or ButtonReleaseMask or KeyPressMask or KeyReleaseMask), 5. Pointer to variable of XEvent structure.

Once after finding out the top level X-window of the AUT using "XQueryTree" and "XFetchName" routines, the widget hierarchy of that instance of AUT along with geometry details can be attained using Editres protocol and Xmu library functions. For every given test case and for the given number of test cases, depending on the type of widget under that test case, necessary X-events are spooled to the associated X-window of the widget / gadget with the required widget data that has been provided. Since the virtualisation of mouse clicks or buttons and keyboard input is possible, the ability for testing the functionality underlying an AUT GUI component or set of components has been achieved. Hence, GUI test automation requirements have been met.

## 4. THE SYSTEM

A software system for automating the GUI testing of any X/Motif application has been realised with the concept and approach mentioned in the previous sections. It is the software system based on X/Motif for automating the testing through the possible test cases fed through a tester input file. This software system is available on RHEL 5.4 and TRU UNIX 5.1B operating system based environments and is portable to any flavour of UNIX and LINUX operating systems.

Let the software system's name as: 'XmotifAplnAutoTestSoftware'. User can test the GUI of his / her interested X/Motif application in the respective environment by invoking the software system as follows:

**XmotifAplnAutoTestSoftware** <X/Motif application name under test>

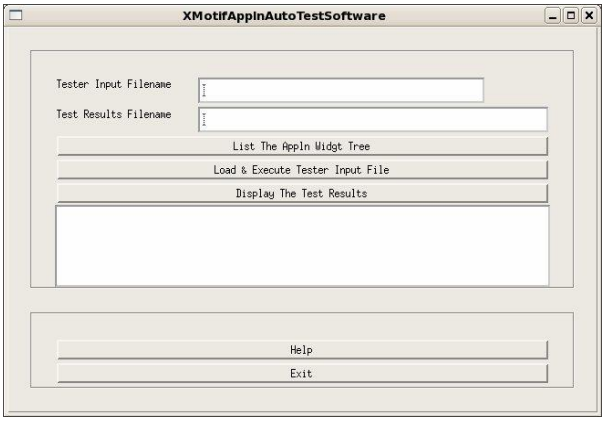


**Fig -1:** Main GUI of X/Motif Application Auto Test Software

User can click over the push button titled "List The Appln Widget Tree" for getting corresponding X/Motif application widget tree. The widget tree will be reported in the current working directory with the name coined as <name of the application under test>XtestAppLst<random six alphabetic chars>. This file can be made use of making tester input files. The ASCII file is with contents presented as number of widgets and widget information (serial number, window, widget id, name, class and path). Path contents shall be used as resource name against wdgtName keyword in the tester input file.

A simple tester input file sample for clicking / testing the push button of the application named as 'MainForPushBtn' is as follows:

```
xappName MainForPushBtn
tstBegin
testStrt 1
testLevl 1
wdgtName 1 MainForPushBtn.formWidget.Push Me
wdgtType 10
testEnd 1
tstEnd
```

The above tester input file contains one test case at single level for clicking the first occurrence of the push button consisting of the widget resource "MainForPushBtn.formWidget.Push Me". The widget type 10 is for push button. "Load & Execute Tester Input File" push button can be clicked for loading, validating and executing test cases over the application under test as per the contents of given tester input file. During the test execution, the software system's GUI is made insensitive. Message window is for displaying the error messages if any and also for displaying the test results.

The system enables for testing up to 1000 test cases through one tester input file. It has the features of repeating any desired test case multiple times and also the facility for giving ranges of data input for the desired test case.

Two phases of the software system can be depicted as follows: User has to invoke the Application Under Test (AUT) in the respective environment.

### 4.1 Phase 1: Tester Input Files Preparation

This phase is manual and the test cases are generated as tester input files for the required test combinations of the AUT. Preparation of test cases are based on the keywords mentioned under test case generation section. For preparing the test cases, the widget resources are important and this can be attained through getting the AUT's widget tree by clicking "List The Application Widget Tree" of **XmotifAplnAutoTestSoftware** system.

### 4.2 Phase 2: Test Cases Auto Execution on Application under Test

Load and execute the prepared and required tester input file by typing the name of the tester input file and by clicking the button meant for it. As per the loaded and validated tester input file, the execution of the test cases on the application under test shall be as per the following flow chart shown in Fig-2. Each test case is executed as per the test case input by sending the required X events on to the specific X window identifier of the X/Motif Application Under Test. The result of the test case is verified against the mentioned X window availability and expected output mentioned as part of the test case.

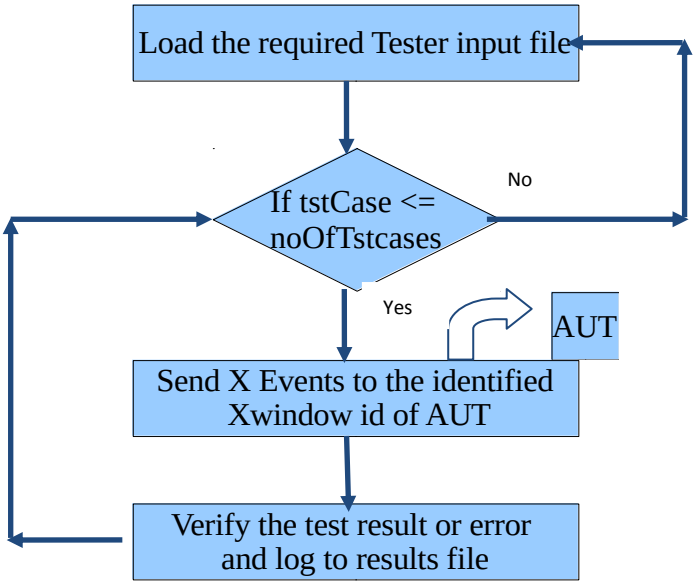


**Fig -2:** Flow Chart for Test Cases Auto Execution

### 4.3 Results

To cover all the GUI components under X/Motif, twenty five widget types have been identified. They are push button, toggle button, label, option menu, radio box, combo box, main window, check box, pull down menu, pull right menu, text widget, scroll text widget, file selection box, dialog selection box, scroll list, scroll vertical bar, scroll horizontal bar, scale widget, up arrow button, down arrow button, left arrow button, right arrow button, drawing area widget etc...

To test the capacity of this approach and to accommodate all possible test combinations, one X/Motif application for each category of the widget has been developed. Further, few applications covering combinations of some or most of the widget types were also tested. The applications were named with the convention such as MainForPushBtn for push button type of widget and so on. Over all fifty testers input files have been generated with one tester input file covering test case combinations of the corresponding AUT. The tester input files contain test cases ranging from 1 to 1000.

The observations have been different with reference to Editres protocol and in getting widget resources for different X servers on TRU-64 UNIX and RHEL 5.4 operating systems. The results are as per expectations and the testing time for each of the tester input file is very fast. Since the delay between two test case executions over AUT is configurable, automation of GUI testing can be adjusted accordingly with the necessary observing or monitoring requirements. The system is very helpful in covering many of the test cases which cannot be done manually. It is helpful in regression testing and enhances the confidence on the AUT reliability. Since many or all combinations of test cases are possible to get tested and observations logged, it elevates the GUI testing and makes easy and error less. Designers and testers are required to prepare the tester input file(s) for any X/Motif application once only. The preparation of tester input file is completely transparent and not based on any scripting or programming language. It is very much possible to repeat the test case multiple times either with the same input or with range of input. This approach and the concept do not require any recording of the test cases.

The verification of the test cases (Test oracle mechanism) are possible by looking at the state of the required widget or window through the corresponding widget resource mentioned against a specific keyword. The call back procedures underlying the GUI components can be tested through the auto click and auto text inputting. The system is helpful in using at all levels of testing of the X/Motif application. This approach and the system are also helpful in generating automatic demonstration of the corresponding X/Motif application.

## 5. CONCLUSIONS

Test case preparation is manual and the tester requires preliminary knowledge about widget resources. The automation of test case generation can be thought of enhancement to this software system. The software under testing shall be compatible to Editres protocol. However, this concept and approach of testing the X/Motif application is distinct from record-playback methods and is the required solution for GUI test automation. This software system is certainly a boost to the X/Motif GUI testing and a bonanza for the improvement of software development life cycle process.

## ACKNOWLEDGEMENTS

The author would like to thank Mr B.Prabakaran, MDPD/MDG, ISRO Satellite Centre for the help and support during the realization of this research.

## REFERENCES

[1] The definitive guides to the X window system (Volume One): Xlib Programming Manual for Version 11 by Adrian Nye; O'Reilly & Associates, Inc.
[2] The definitive guides to the X window system (Volume Four): X Toolkit Intrinsics Programming Manual for OSF / Motif 1.2 Edition by Adrian Nye and Tom O'Reilly; O'Reilly & Associates, Inc.
[3] The definitive guides to the X window system (Volume Six A): Motif Programming Manual for OSF / Motif 1.1 Edition by Dan Heller; O'Reilly & Associates, Inc.
[4] Graphical User Interface with X-windows and Motif By Muralidhar R. Rao & Ganga Prasad G.L; Wiley Eastern Limited.
[5] Software Test Automation: Effective use of test execution tools by Mark Fewster & Dorothy Graham; Addision-Wesley.
[6] "Test Automation" by Macario Polo, Pedro Reales, Mario Piattini and Christof Ebert, IEEE Software, January/February 2013.
[7] http://www.rahul.net/kenton/editres.html
[8] http://www.rahul.net/kenton/events.html
[9] linux.die.net/man/3/xtestfakebuttonevent
[10] Xtstlibrary(cgit.freedesktop.org/xorg/lib/libXtst/libXtst-1.2.2.tar.gz)
[11] http://www.linuxquestions.org/questions/programming-9/simulating-1-mouse-click-594576/
[12] http://www.x.org/archive/X11R6.8.1/doc/editres.1.html
[13] http://www.linuxdocs.org/HOWTOs/XWindow-User-HOWTO-2.html
[14] http://www.lehman.cuny.edu/cgi-bin/man-cgi?editres+1
[15] Xmu library (cgit.freedesktop.org/xorg/lib/libXmulibXmu-1.1.2.tar.gz)
[16] http://homepage3.nifty.com/tsato/xvkbd/xvkbd-3.5.tar.gz
[17] http://www.semicomplete.com/projects/xdotool/xdotool.html
[18] sourceforge.net/projects/xdotool-gui/xdotoolgui-1.2.1.tar.gz
[19] cgit.freedesktop.org/xorg/app/editres/
[20] http://www.rahul.net/kenton/xsites.frames.html
[21] manpages.ubuntu.com/manpages/lucid/man1/xdotool.html
[22] http://www.x.org/released/X11R7.6/doc/libXmu/Xmu.html