An Approach to Testing X Window System Servers at a Protocol Level

Michael Lee Squires

Len Wyatt

Sequent Computer Systems, Inc.
15450 SW Koll Parkway
Beaverton, OR 97006
(503) 626-5700

ABSTRACT

An approach to testing display servers for the X Window System is presented which depends only on an underlying communications mechanism. The server tester is intended to provide coverage for those server specifications which cannot be tested from the Xlib level. It also ensures that the basic facilities needed to run the Xlib tests are operational. All testing is performed by sending messages to the server and evaluating the responses from the server.

1. INTRODUCTION

The purpose of this document is to present a set of requirements for a *server tester* and to outline a software structure which will satisfy those requirements. This document is intended to guide not only the server tester developers, but other X Window System developers and testers also.

1.1 Background

There are a variety of interfaces, services, and protocols specified as part of the X Window System. These are implemented by a set of components which may be produced by different organizations, running on different hardware and using different operating systems.

In order to promote a standard, it is not sufficient to merely publish an interface specification. Examples of software which implement the standards and which illustrate the use of the services must be made available. Test procedures and test suites which verify that specific component implementations adhere to the specifications must be defined and made generally available.

The MIT X effort has addressed the interface specification, software implementation, and example use. It has not, thus far, addressed the issue of testing and specification compliance.

In the interests of promoting the X Window System as an industry standard, a number of organizations have formed a *consortium* for the purpose of defining and generating an appropriate set of test tools and procedures for X. This document addresses a specific facet of X testing which has been identified by the consortium.

2. PROBLEM STATEMENT

2.1 Testing Goals

2.1.1 General

The goals for testing X implementations are to:

— promote *portability* of applications and application libraries by ensuring a consistent set of interfaces across disparate computer systems and operating systems. That is, if we have an application program which compiles and runs on system A, we should be able to transfer the source to system B and, within the constraints of language and operating system portability, compile and run that program satisfactorily. Since most applications are currently written to the procedural interface provided in C by the Xlib library, this interface is the first target of the testing effort for portability.

— promote *interoperability* of clients and servers which may be running on disparate computer systems. That is, if we have a client which runs on system A and uses the display server available on system B, then that client should be able to operate successfully with a server running on system C (ignoring protocol extensions and client device dependencies).

2.1.2 Server Tester Goals

The primary goal of the server tester is to address those areas of interoperability which cannot be adequately covered by tests at the Xlib level. A secondary goal is to facilitate the Xlib testing by ensuring that the basic server functions are operational. Another secondary goal is to provide a low-level diagnostic tool which would be useful to server developers.

2.2 Definitions

The following definitions form a basis for understanding the requirements of the server tester.

We define an **interface** as an implementation dependent functional boundary between two adjacent components. Taking Xlib as an example, the function names, calling sequences, and data types necessary to interact with Xlib are its interface.

We define a **service** as an implementation independent set of capabilities that a particular layer of software provides. Again using Xlib as an example, we could define an equivalent level library for another language which provided the same services via a different interface.

We define a **protocol** as the specification of interaction between two peer processes. This specification consists of:

— a message catalog

— sequencing rules for message exchange

— the specifications of the actions of the individual processes in response to the message sequencing

When we examine the X Window System, we see that the X protocol specification encompasses three components: the client, the server, and the display hardware. Thus a complete test of a server's adherence to the protocol specification would include observing its effect on the display hardware as well as its ability to adhere to the rules for message sequencing.

We also notice that the "XLib Interface Specification" actually consists of both an interface and a service specification. Furthermore, the service specification depends on the adherence of a particular server to the protocol specification.

Because Xlib was designed to provide fairly direct access to the underlying X protocol, many of the messages defined in the protocol specification and their corresponding actions have a direct mapping at the Xlib level. This concept of a **mapping** between the Xlib specification and the protocol specification is a key element in determining the requirements for a server tester.

3. APPROACH

3.1 General

Because many of the actions specified in the protocol specification can be tested at the Xlib level, because the Xlib level will be more heavily used by the majority of X-based programmers, and because DEC is mounting a substantial effort to rigorously test the Xlib interfaces and services, we will defer the bulk of the testing of server actions to the Xlib testing effort.

We will design and implement a server tester which will concentrate on testing those features of the protocol which are not testable at the Xlib level or which are prerequisites to the Xlib level testing. Thus, we will test the ability of the server to accept all legal message types and respond appropriately. We will ensure that server capabilities which the Xlib testing will depend on work within some bounds (e.g ensure that the server can send a pixmap to the client upon request). We will test to ensure adherence to the canonical byte stream of the protocol, independent of the host byte sex or compiler structure packing.

3.2 Determining Specific Requirements

We have developed a decision tree which allows us to determine which server actions should be addressed by the server tester.

The first decision in the tree is "is the mapping of this action (and the stimulus to cause it) to the Xlib level direct enough to allow it to be tested at the Xlib level?" An example of an action which can be tested at the Xlib level is drawing a line within a window. Initializing the server connection with an incorrect protocol version number is an example of a test which cannot be performed at the Xlib level.

If the mapping is direct enough to allow testing at the Xlib level, then we need to ask "is this capability fundamental enough that the Xlib tests cannot be reasonably performed if the capability is deficient?" An example of a fundamental capability is the server's ability to send error responses.

If the mapping is not direct enough to allow the testing at the Xlib level, then we ask "can the action be inferred from other data available at the Xlib level?" An example of something which could be inferred is the initialization of the default color map. By reading the color map immediately following opening the display, an Xlib level test can determine if the color map was properly initialized.

Decision #1		Decision #2		Result
Is the mapping direct enough to test at the Xlib level?	NO	Can the action be inferred?	NO	TEST
			YES	DON'T TEST
	YES	Is the action fundamental?	NO	DON'T TEST
			YES	TEST

We will apply this set of decisions to the protocol specification to generate specific tests to perform in the server tester. Generating this list is a specific deliverable on the development schedule (see section 6).

4. SPECIFICATIONS

4.1 General Requirements

The server tester must satisfy the following requirements:

— It will not use X as a basis for its user interaction. (This provides a new server developer the ability to run the tester using standard character terminals.)

— It will verify that the server can accept all legal message types.

— It will verify that the server will reject invalid message types.

— It will perform all tests which are derived from applying the decision criteria in section 3.2 to the protocol specification. The complete list of tests is presented in section 4.2.

— It will be portable to different computer systems and operating systems. (That is, any machine/operating system dependencies will be isolated. We are not committing to port it to any/all systems.)

— It will adapt the tests to different display hardware. (That is, a mechanism for modifying test parameters based on display characteristics will be provided.)

— It will provide the capability to continue a test suite after errors are encountered in a particular test.

— It will provide the capability to test a server's ability to deal with clients running on different byte sex hosts. (That is, the tester will be capable of 'masquerading' as a host with a different byte sex than it really has. This allows an organization to more completely test a server with a single host system.)

Additionally, we will provide user documentation and internal documentation for the server tester.

4.2 Specific Tests

This subsection is currently **TBD**. The detailed list of tests is the first deliverable discussed in section 6.

5. SOFTWARE STRUCTURE

The basic approach to developing the server tester and the test cases which use it is to provide a layered set of C routines which constitute the bulk of the test mechanism. Individual test cases will be implemented as C programs which make use of this layered set. We will explore development of some sort of front-end tool (e.g. preprocessor, interpreter) which will facilitate test case generation, but given the resource and schedule constraints we are unwilling to commit to that at this time.

We will structure the test cases as a series of C programs, each one designed to test a specific capability or set of capabilities. This allows incremental development of the test cases, selective use of individual tests during development or diagnoses, and ease of test suite configuration via use of command scripts.

The software will consist of 5 software layers:

— Communication Manipulation - lowest level - contains the machine and operating dependent software - responsible for byte swapping, buffering, and ensuring the canonical byte stream.

— Message Handling low level utilities (which look *very* similar to the various internal utilities within the current Xlib implementation) - expose more of their internal data structures than Xlib does to provide test case writers the capability to "damage" messages

— Message Generation - essentially a subset of the current Xlib with many parameters omitted in favor of their defaults - (i.e. Generate_Standard_Create_Window_Request would be a candidate routine)

— Adaptation - all routines necessary to adapt the tests to specific display parameters - (e.g. Generate_Safe_Rectangle_Set)

— specific test cases

Note that this software really consists of two major components - the "tools" (levels 1-4) and the test cases themselves. This division is used to generate the milestones in section 6.

```
+---------------------------------------+
|               TEST CASE               |
+---------------------------------------+
      |         |         |    |
      |         |         |    |
      v         |         |    |
+------------+  |         |    |
| ADAPTATION |  |         |    |
+------------+  |         |    |
                |         |    |
                |         |    |
                v         |    |
+----------------------+  |    |
|  MESSAGE GENERATION  |  |    |
+----------------------+  |    |
                          |    |
                          |    |
                          v    |
+-------------------------+    |
|    MESSAGE HANDLING     |    |
+-------------------------+    |
                               |
                               |
                               v
+---------------------------------------+
|    COMMUNICATION MANIPULATION         |
+---------------------------------------+
```

6. DEVELOPMENT SCHEDULE

Deliverable	Date
Complete list of server tests	7/17/87
Tools + initial set of test cases	10/9/87
Remainder of test cases	12/18/87