X WINDOW SYSTEM

C Library and Protocol Reference

Robert W. Scheifler

James Gettys

Ron Newman

X WINDOW SYSTEM

C Library and Protocol Reference

Robert W. Scheifler James Gettys Ron Newman

With Al Mento and Al Wojtas

digital™ Digital Press

Printed in the United States of America.

9 8 7 6 5 4 3 2 1

Order number EY-6737E-DP

Design: David Ford
Manuscript editor: Christie Williams
Production coordinator: Editorial Inc.
Index: Howard Burrows and Rosemary Simpson
Compositor: Black Dot Graphics
Printer: Murray Printing Company

Library of Congress Cataloging in Publication Data

Scheifler, Robert W., 1954-
X window system.

Includes index.
1. X Window System (Computer system) 2. C (Computer program language) I. Gettys, James, 1953–
II. Newman, Ron, 1957- . III. Title.
QA76.76.W56S34 1988 005.4'3 88-30869
ISBN 1-55558-012-2

Acknowledgments

Xlib—C Library X Interface

The design and implementation of the first ten versions of X were primarily the work of three individuals: Robert Scheifler of the MIT Laboratory for Computer Science, Jim Gettys of Digital Equipment Corporation, and Ron Newman of MIT, while at MIT/Project Athena. X version 11, however, is the result of the efforts of dozens of individuals at almost as many locations and organizations. At the risk of offending some of the players by exclusion, we would like to acknowledge some of the people who deserve special credit and recognition. Our apologies to anyone inadvertently overlooked.

First, our thanks goes to Phil Karlton and Scott McGregor, both of Digital, for their considerable contributions to the specification of the version 11 protocol. Susan Angebranndt, Raymond Drewry, Todd Newman, and Phil Karlton of Digital worked long and hard to produce the sample server implementation.

Next, our thanks goes to Ralph Swick (MIT/Project Athena and Digital) who kept it all together for us. He handled literally thousands of requests from people everywhere and saved the sanity of at least one of us. His calm good cheer was a foundation on which we could build.

Our thanks also go to Todd Brunhoff (Tektronix) who was "loaned" to MIT/Project Athena at exactly the right moment to provide very capable and much-needed assistance during the alpha and beta releases. He was responsible for the successful integration of sources from multiple sites; we would not have had a release without him.

Our thanks also go to Al Mento and Al Wojtas of Digital's ULTRIX Documentation Group. With good humor and cheer, they took a rough draft and made it an infinitely better and more useful document. The work they have done will help many everywhere. We also would like to thank Hal Murray (Digital SRC) and Peter George (Digital VMS) who contributed much by proofreading the early drafts of this document.

Our thanks also go to Jeff Dike (Digital UEG), Tom Benson, Jackie Granfield, and Vince Orgovan (Digital VMS), who helped with the library utilities implementation; to Hania Gajewska (Digital UEG-WSL) who, along with Ellis Cohen (CMU and Siemens), was instrumental in the semantic design of the window manager properties; to Dave Rosenthal (Sun Microsystems) who also contributed to the protocol and provided the sample generic color frame buffer device-dependent code; and to Tim Greenwood (Digital IECG) for his help in understanding international keyboards and for providing the KeySyms in Appendix E.

The alpha and beta test participants deserve special recognition and thanks as well. It is significant that the bug reports (and many fixes) during alpha and beta test came almost exclusively from just a few of the alpha testers, mostly hardware vendors working on product implementations of X. The continued public contribution of vendors and universities is certainly to the benefit of the entire X community.

Our special thanks must go to Sam Fuller, Vice-President of Corporate Research at Digital, who has remained committed to the widest public availability of X and who made it possible to greatly supplement MIT's resources with the Digital staff in order to make version 11 a reality. Many of the people mentioned here are part of the Western Software Laboratory (Digital UEG-WSL) of the ULTRIX Engineering group and work for Smokey Wallace, who has been vital to the project's success. Others not mentioned here worked on the toolkit and are acknowledged in the X Toolkit documentation.

Of course, we must particularly thank Paul Asente, formerly of Stanford University and now of Digital UEG-WSL, who wrote W, the predecessor to X, and Brian Reid, formerly of Stanford University and now of Digital WRL, who had much to do with W's design.

Finally, our thanks go to MIT, Digital Equipment Corporation, and IBM for providing the environment where it could happen.

X Window System Protocol

The primary contributors to the X11 protocol are: Dave Carver (Digital HPW); Branko Gerovac (Digital HPW); Jim Gettys (Digital SRC); Phil Karlton (Digital WSL); Scott McGregor (Digital SSG); Ram Rao (Digital UEG); David Rosenthal (Sun Microsystems); and Dave Winchell (Digital UEG).

The implementors of initial server who provided useful input are: Susan Angebranndt (Digital WSL); Raymond Drewry (Digital); and Todd Newman (Digital).

The invited reviewers who provided useful input are: Andrew Cherenson (Berkeley); Burns Fisher (Digital VMS); Dan Garfinkel (HP); Leo Hourvitz (Next); Brock Krizan (HP); David Laidlaw (Stellar); Dave Mellinger (Interleaf); Ron Newman (MIT); John Ousterhout (Berkeley); Andrew Palay (ITC CMU); Ralph Swick (MIT/Project Athena and Digital); Craig Taylor (Sun Microsystems); and Jeffery Vroom (Stellar).

Thanks go to Al Mento of Digital's UEG Documentation Group for formatting this document.

This document does not attempt to provide the rationale or pragmatics required to fully understand the protocol or to place it in perspective within a complete system.

The protocol contains many management mechanisms that are not intended for normal applications. Not all mechanisms are needed to build a particular user interface. It is important to keep in mind that the protocol is intended to provide mechanism, not policy.

Robert W. Scheifler
Laboratory for Computer Science
Massachusetts Institute of Technology

Jim Gettys
Systems Research Center
Digital Equipment Corporation

Ron Newman
Project Athena
Massachusetts Institute of Technology

September 1988

Introduction

The X Window System, or X, is a network-transparent window system. With X, you can run multiple applications simultaneously in windows, generating text and graphics in monochrome or color on a bitmap display. Network transparency means that you can use application programs that are running on other machines scattered throughout the network, as if they were running on your machine. Because X permits applications to be device independent, applications need not be rewritten, recompiled, or even relinked to work with new display hardware.

X provides facilities for generating multifont text and two-dimensional graphics (such as points, lines, arcs, and polygons) in a hierarchy of rectangular windows. Every window can be thought of as a "virtual screen" and can contain subwindows within it, to an arbitrary depth. Windows can overlap each other like stacks of papers on a desk and can be moved, resized, and restacked dynamically. Windows are inexpensive resources; applications using several hundred subwindows are common. For example, windows are often used to implement individual user interface components such as scroll bars, menus, buttons, and so forth.

Although you may think of yourself as a client of the system, in network terms, the application programs you run are called clients and they use the network services of the window system. A program running on the machine with your display provides these services and so is called the X server. The X server acts as an intermediary between you and the applications, handling output from the clients to the display and forwarding your input (entered with a keyboard or mouse) to the appropriate clients for processing.

Clients and servers use some form of interprocess communication to exchange information. The syntax and semantics of this conversation are defined by a communication protocol. This protocol is the foundation of the X Window System and is presented in Part II of this book. Clients use the protocol to send requests to the server to create and manipulate windows, to generate text and graphics, to control input from the user, and to communicate with other clients. The server uses the protocol to send information back to the client in response to various requests and to forward keyboard and other user input on to the appropriate clients.

Because a network roundtrip is an expensive operation relative to basic request execution, the protocol is primarily asynchronous, and data can be in transit in both directions (client to server and server to client) simultaneously. After generating a request, a client typically does not wait for the server to execute the request before generating a new request. Instead, the client generates a stream of requests that are eventually received by the server and executed. The server does not acknowledge receipt of a request and, in most cases, does not acknowledge execution of a request. (This is possible because the underlying transport being used is reliable.)

The protocol is designed explicitly to minimize the need to query the window system for information. Clients should not depend on the server to obtain information that the clients initially supplied. In addition, clients do not poll for input by sending requests to the server. Instead, clients use requests to register interest in various events, and the server sends event notifications asynchronously. Asynchronous operation may be one of the most significant differences between X and other window systems with which you may be familiar.

For the best performance, when the client and the server reside on the same machine, communication between them often is implemented using shared memory. When the client and the server reside on different machines, communication can take place over any network transport layer that provides reliable, in-order delivery of data in both directions (usually called a reliable duplex byte stream). For example, TCP (in the Internet protocol family) and DECnet streams are two commonly used transport layers. To support distributed computing in a heterogeneous environment, the communication protocol is designed to be independent of the operating system, programming language, and processor hardware. Thus, you can use a single display to run applications written in multiple languages under multiple operating systems on multiple hardware architectures simultaneously.

Although X is fundamentally defined by a network protocol, most application programmers do not want to think about bits, bytes, and message formats. Therefore, X has an interface library. This library provides a familiar procedural interface that masks the details of the protocol encoding and transport interactions and automatically handles the buffering of requests for efficient transport to the server, much as the C standard I/O library buffers output to minimize system calls. The library also provides various utility functions that are not directly related to the protocol but that are nevertheless important in building applications. The exact interface for this library differs for each programming language. Xlib is the library for the C programming language and is presented in Part I of this book.

The accompanying figure shows a block diagram of a complete X environment. Each X server controls one or more screens, a keyboard, and a pointing device (typically a mouse) with one or more buttons on it. There can be many X servers; often there is one for every workstation on the network. Applications can run on any machine, even those without X servers. An application might communicate with multiple servers simultaneously (for example, to support computer conferencing between individuals in different locations). Multiple applications can be active at the same time on a single server.

In X, many facilities that are built into other window systems are provided by client libraries. You will not find specifications for things like menus, scroll bars, and dialog boxes; nor will you find the interpretation of particular key and button sequences in this book. The protocol and Xlib avoid mandating such policy decisions as much as possible. You can view the protocol and Xlib as a construction kit providing a rich set of mechanisms that can implement a variety of user interface policies. Toolkits (providing menus, scroll bars, dialog boxes, and so on), higher-level graphics libraries (which might transform abstract object descriptions into graphics requests, for example), and user interface management systems (UIMS) can all be implemented on top of Xlib. Although Xlib provides the foundation, the expectation is that applications will be written using these higher-level facilities in conjunction with the facilities of Xlib, rather than solely on the "bare bones" of Xlib.

You can think of the total user interface as having two primary compo-

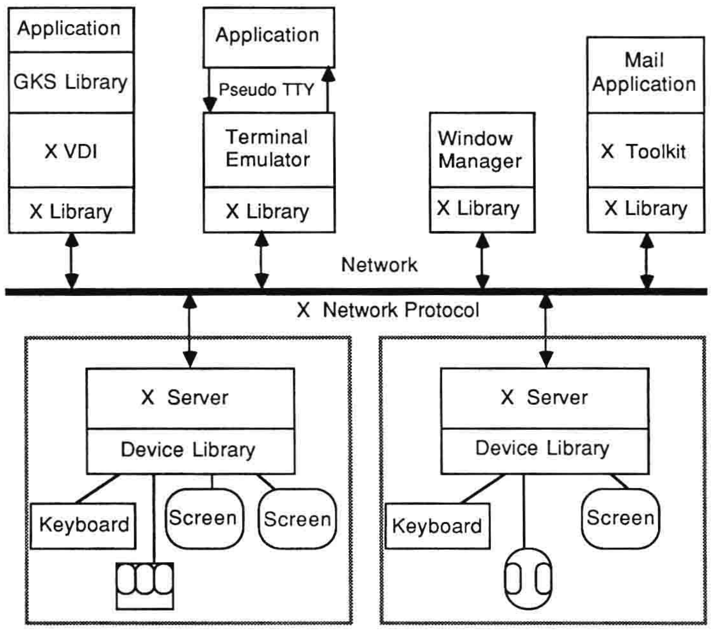

Figure 1. X window system block diagram

nents: the interaction with the user that is logically internal to an application (for example, typing text into a text editor or changing a cell's contents in a spreadsheet) and the interaction that is logically external to an application (for example, moving or resizing an application window or turning an application window into an icon). The external user interface is built into many other window systems, but this is not the case with X. The X protocol does not define an external user interface at all. Rather, the protocol provides mechanisms with which a variety of external user interfaces can be built. These mechanisms are designed so that a single client, called a window manager, can provide the external user interface independent of all of the other clients.

A window manager can automatically:

- Provide title bars, borders, and other window decorations for each application
- Provide a uniform means of moving and resizing windows

- Enforce a strict window layout policy if it desires (for example, "tiling" the screen so that application windows never overlap)
- Provide uniform icons for applications
- Provide a uniform interface for switching the keyboard between applications

With a suitable set of conventions, you can construct applications that are insensitive to the external user interface provided by a window manager but that run unmodified in multiple environments and still behave properly.

Because the protocol can deal with such a broad spectrum of user interfaces, no single program, toolkit, UIMS, or window manager is likely to use all of the facilities the protocol and Xlib provide. Do not be concerned if you do not understand why some facility exists; it may support a user interface style with which you are not familiar.

Principles

Early in the development of X, we argued about what should and should not be implemented in the server. For example, we did not know if menus or terminal emulators could be implemented in the client with adequate performance or whether "rubber banding" (dynamically stretching a simple figure in response to movement of the pointing device) would be acceptable when performed across a network. Experimentation during the first months showed us that more was possible than we had first believed.

These observations hardened into the following principles, which guided us through the early X design:

- Do not add new functionality unless an implementor cannot complete a real application without it.
- It is as important to decide what a system is not, as to decide what it is. Do not serve all the world's needs, but make the system extensible so that additional needs can be met in an upwardly compatible fashion.
- The only thing worse than generalizing from one example is generalizing from no examples at all.
- If a problem is not completely understood, it is probably best to provide no solution at all.
- If you can get 90 percent of the desired effect for 10 percent of the work, use the simpler solution.

- Isolate complexity as much as possible.
- Provide mechanism rather than policy. In particular, place user interface policy in the client's hands.

The first principle kept the wish list under control. Just because someone wanted something in the server, we did not feel obligated to add it. This kept us focused on the important issues that made real applications work. This principle was a somewhat more difficult touchstone to use during the design of the present version of X, given its significantly larger audience. We modified the principle to be "know of some real application that will require it."

At each iteration of the X design, there was always more to do than time allowed. We therefore focused on mechanisms with the broadest applicability and for which consensus in the group could easily be achieved. For example, we focused on two-dimensional graphics, explicitly deferring three-dimensional graphics.

At the same time, to avoid obsolescence, we designed the present version of X to be extensible at both the protocol and library interfaces and without requiring incompatible changes to existing applications. Examples of extensions we had in mind were additional graphics models (such as PHIGS and PostScript), real-time video, and general programmability in the server. (We view programmability as simply one example of an extension, not as the sole mechanism for extensibility; mere programmability does not give you support for video or high-performance support for graphics.)

During the design and implementation process, we generally suspected that any problems were just the tips of large icebergs. Expending effort to solve an immediate problem without first trying to generalize the problem is usually a mistake; a few related examples often make a whole class of problems obvious. This is not to say that we ignored the first instance of a problem; often there were adequate solutions using existing mechanisms.

We attempted to avoid solutions to problems we did not fully understand. For example, the preliminary design for the present version of X supported multiple input devices (more than just a single keyboard and mouse). As we worked through the design, we realized it had flaws that would take significant time and experimentation to correct. As a result, we removed this support from the system, knowing that correct support could be added later through the extension mechanism.

We also tried to avoid winning a complexity merit badge. If we could get most of what we needed with less complexity than a complete solution would require, we were willing to compromise our goals. Only history will decide if these tradeoffs were successful. Much of the existing complexity is a result of providing support for external window management; most programmers need not be concerned with this, particularly those using an X toolkit. We expected that toolkits would hide various forms of tedium from the programmer. For example, a program that displays "Hello World" with configurable colors and font and obeys window management conventions is about 150 lines of code when written using only the facilities of Xlib; an equivalent program written using a toolkit can have fewer than a dozen lines of code. Thus, it is important to keep in mind that Xlib is only one layer in a complete X programming environment.

Isolation of complexity is necessary in large systems. A system in which every component is intimately related to every other becomes difficult to change as circumstances change. We therefore attempted to build as much as possible into client programs, introducing only the minimum mechanisms required in the server.

Deciding what a system is not is as important as deciding what it is. For example, at various times people urged that remote execution and general interclient remote procedure call be integral parts of X. They felt there were no established standards in these areas, and they wanted X to be a self-contained environment. As is often the case, solving the immediate problem by adding to the existing framework rather than by integrating into a larger framework is less work, but the result is not satisfactory for long. The X protocol is correctly viewed as just one component in an overall distributed systems architecture, not as the complete architecture by itself.

User interface design is difficult and currently quite diverse. Although global user interface standards might someday be possible, we believed it prudent to promote the cooperative coexistence of a variety of user interface styles and to support diverse user communities and ongoing research activities. By separating window management functions from the server and from normal applications and by layering user interface policy in higher-level libraries on top of Xlib, we allowed for experimentation without forcing all users to be guinea pigs. As a result, many existing user interfaces have been imported into the X environment. Having a "pick one or roll your own" policy instead of a "love it or leave it" one has drawbacks; the applications developer must choose a user interface style and user community. You should remember, however, that Xlib and the protocol is not an end but a foundation.

History

X was born of necessity in 1984. Bob Scheifler was working at MIT's Laboratory for Computer Science (LCS) on a distributed system called Argus and was in need of a decent display environment for debugging multiple distributed processes. Jim Gettys, a Digital engineer, was assigned to MIT's Project Athena, an undergraduate education program sponsored by Digital and IBM that would ultimately populate the campus with thousands of workstations.

Neither Digital nor IBM had a workstation product with a bitmap display in 1984. The closest thing available from Digital was a VS100 display attached to a VAX. Both Athena and LCS had VAX-11/750s, and Athena was in the process of acquiring about 70 VS100s. VS100s were in field test at the time, and the firmware for them was unreliable. Athena loaned one of the first VS100s to LCS in exchange for cooperative work on the software. Our immediate goal was clear: We needed to build a window system environment running under UNIX on VS100s for ourselves and the groups we worked for. We had little thought of anything beyond these goals, but wondered where to begin. Little software was available elsewhere that was not encumbered by license or portability.

Paul Asente and Brian Reid, then both at Stanford University, had developed a prototype window system called W to run under Stanford's V operating system. W used a network protocol and supported "dumb terminal" windows and "transparent graphics" windows with display lists maintained in the server. In the summer of 1983, Paul Asente and Chris Kent, summer students at Digital's Western Research Laboratory, ported W to the VS100 under UNIX. They were kind enough to give us a copy.

The V system has reasonably fast synchronous remote procedure call, and W in the V environment was designed with a synchronous protocol. The port to UNIX retained the synchronous communication even though communication in UNIX was easily five times slower than in V. The combination of prototype VS100s with unreliable firmware and W using slow communication was not encouraging, to say the least; one could easily type faster than the terminal window could echo characters.

In May of 1984, we received reliable VS100 hardware and firmware. That summer, Bob replaced the synchronous protocol of W with an asynchronous protocol and replaced the display lists with immediate mode graphics. The result was sufficiently different from W that continuing to call it W was inappropriate and would cause confusion, as W was in some limited use at Athena. With no particular thought about the name, and because the familial resemblance to W was still strong at that date, Bob called the result X. Much later, when the name became a serious issue, X had already stuck and was used by too many people to permit a change.

Development was rapid during the next eight months. The first terminal emulator (VT52) and window manager were written in the CLU programming language, the language of choice in the research group where Bob worked. Bob continued development of the server and the protocol, which went from version 1 to version 6 during this period (the version number was incremented each time an incompatible change was made). Mark Vandevoorde at Athena wrote a new VT100 terminal emulator in C, and Jim Gettys worked on Xlib and the UNIX support for starting the window system. Late in 1984, we received faster VS100 firmware, causing the first round of performance analysis and optimization. Within a few weeks, we were again hardware limited, but we had a much better understanding of performance issues.

By early 1985, many people inside Digital were using X, and plans were underway for the first Digital UNIX workstation product, which was based on the MicroVAX-II. At the time, support for UNIX in Digital was limited, and there was no chance of getting any other window system except X on Digital hardware. Other systems were either highly nonportable or were unavailable because of licensing problems (this was the case with Andrew). X was the logical candidate. We had ported X version 6 to the QVSS display on the MicroVAX. Ron Newman joined Project Athena at this time and worked on documenting Xlib, already in its third major revision.

We redesigned X to support color during the second quarter of 1985, with Digital's eventual VAXstation-II/GPX as the intended target. Although MIT had licensed version 6 to a few outside groups for a brief time at nominal charge, a key decision was made in the summer of 1985 not to license future versions of X. Instead, it would be available to anyone at the cost of production. In September of 1985, version 9 of X was made publicly available, and the field test of the VAXstation-II/GPX began. During that fall, Brown University and MIT started porting X to the IBM RT/PC, which was in field test at those universities. A problem with reading unaligned data on the RT forced an incompatible change to the protocol; this was the only difference between version 9 and version 10.

During the fall, the first significant outside contributions of code to X started to appear from several universities and from Digital. In January of 1986, Digital announced the VAXstation-II/GPX, which was the first commercial X implementation. Release 3 of X (X10R3) was available in February and was a major watershed in X development. Although we were happy to see a major corporation incorporate X into its product line, we knew the design was limited to the taste and needs of a small group of people. It could solve just the problems we faced, and its hardware origins were still obvious in key aspects of the design. We knew version 10 had inherent limitations that would force major redesign within a few years, although it was certainly adequate for developing many interesting applications.

Over the next few months, a strange phenomenon occurred. Many other corporations, such as Hewlett-Packard, were basing products on version 10, and groups at universities and elsewhere were porting X to other displays and systems, including Apollo Computer and Sun Microsystems workstations. The server was even ported to the IBM PC/AT. Somewhat later, Hewlett-Packard contributed their toolkit to the MIT distribution.

We tired of hearing comments such as "We like X, but there is this one thing you ought to change." People were already declaring it a "standard," which was, to our thinking, premature. Before long, however, we were confronted with a fundamental decision about X's future. We seriously considered doing nothing; after all, X did almost everything we needed it to, and what it did not do could be added without difficulty. Unfortunately, this would leave many people using an inadequate platform for their work. In the long run, X would either die because of its inadequacies, or it would spawn wildly incompatible variations. Alternatively, based on feedback from users and developers, we could undertake a second major redesign of X.

Although we were willing to do the design work, we knew that the resulting design would be ambitious and would require much more implementation work than our meager resources at MIT would permit. Fortunately, Digital's Western Software Laboratory (DECWSL) was between projects. This group had the required expertise, including people who had contributed to pioneering Xerox window systems. More importantly, these people were intimately familiar with X. Smokey Wallace, DECWSL's manager, and Jim Gettys proposed the implementation of version 11, which would then be given back to MIT for public distribution without a license. Digital management quickly approved the proposal.

We started intensive protocol design in May of 1986. No proprietary information was used in the design process. Key contributors included Phil Karlton and Scott McGregor of Digital. Dave Rosenthal of Sun Microsystems was invited to join Digital engineers in the design team, and Bob Scheifler acted as the chief architect. At the first design meeting, we decided it was not feasible to design a protocol that would be upwardly compatible with version 10 and still provide the functionality essential for the range of display hardware that had to be supported. With some reluctance, we abandoned compatibility with version 10 (although Todd Brunhoff of Tektronix has since shown that one can build a reasonable "compatibility server" to display version 10 applications on a version 11 server).

We carried out most of the actual design work using the electronic mail facilities of the DARPA Internet, which connects hundreds of networks around the country, including MIT's campus network and Digital's engineering network. The entire group held only three day-long meetings during the design process. During these meetings we reached a consensus on issues we could not resolve by mail. Even with group members on opposite coasts, responses to most design issues were only a few minutes away. A printed copy of all the messages exchanged during this time would be a stack of paper several feet high. Without electronic mail, the design simply would not have been possible.

Once we completed a preliminary protocol design, we invited people from other companies and universities to review the specification. By August, we had a design ready for public review, which was again carried out using electronic mail, courtesy of the Internet. Design of the sample server implementation started at this time. Phil Karlton and Susan Angebranndt of DECWSL designed and implemented the device-independent parts of the server, and